JAVA Cheat Sheet

(Cover all Basic Java Syntaxes)

Over 300 Examples

Ray Yao (2nd Edition)

JAVA

Cheat Sheet

**(Cover all Basic Java Syntaxes,
More Than 300 Examples)**

Ray Yao

About This Book

This book covers all basic Java syntaxes, almost each syntax entry lists a program example and result output**.** We can quickly reference the most helpful programming syntaxes, such as common command syntax, string function syntax, collection function syntax, class & object syntax……; all these syntaxes are very useful for programming**.**

We can take this book as a basic syntax manual because its entries are arranged alphabetically so that we can easily reference the important syntax**.**

Nowadays or in the future, the Java Syntax Book can provide great help for coding both in our study and our work**.**

Disclaimer

This book is intended as a basic syntax manual only; it cannot include all entries on this subject**.** Its purpose is as a supplement for a cheat sheet book, not as a whole Java dictionary**.**

Table of Contents

Syntax Chart

& syntax

"&" has two functions.

1. Work as a relational operator.

// Return true if two conditions are true, return false if one of the conditions is false.

// "&" evaluates all conditions, even if the first condition is false.

e.g.

```
public class Main {
public static void main(String[] args){
int x = 100, y = 90, z = 60;
if ((x > y) & (x > z))
System.out.println("The value of x is " + x);
}
}
```

// Output: The value of x is 100

2. Work as a bitwise AND

// Return 1 if two bits are 1, return 0 if one of the bits is 0.

e.g.

```
public class Main {
    public static void main(String[] args){
        int x= 4;
        int y = 6;
        int z = x & y;
        System.out.println(x + " & " + y + " = " + z);
    }
}
```

// Output: 4

| syntax

"|" is used for bitwise Or.

// Return 1 if one of the bits is 1, return 0 if all bits are 0.

e.g.

```
public class Main {
    public static void main(String[] args){
        int x= 4;
        int y = 6;
        int z = x | y;
        System.out.println(x + " | " + y + " = " + z);
    }
}
// Output:  6
```

---

^ syntax:

"^" is used for bitwise XOR

```
// if corresponding bits are different, it returns 1, else it returns 0.
e.g.
public class Main {
    public static void main(String[] args){
        int x= 4;
        int y = 6;
        int z = x ^ y;
        System.out.println(x + " ^ " + y + " = " + z);
    }
}
// Output:  4 ^ 6 = 2
```

<< syntax:

operand << bits

```
// Left shift all bits according to a given number of bits .
e . g .
public class Main {
    public static void main (String[] args) {
        int operand = 2;
        // left shift 2 bits
        int n = operand << 2;
        System.out.println(n);
    }
```

```
}
// Output:  8
```

>> syntax:

```
operand  >>  bits
// Right shift all bits according to a given number of bits.
e.g.
public class Main {
    public static void main (String[] args) {
        int operand = 8;
        // right shift 2 bits
        int n = operand >> 2;
        System.out.println(n);
    }
}
// Output:  2
```

>>> syntax:

```
operand  >>>  bits
// Right shift all bits according to a given number of bits.
// The vacancy in the leftmost position is filled with 0
// Unsigned right shift.
e.g.  Given  -2 = 11111110
-2 >>> 1    // return  01111111
```

// and /* */ syntax:

Single-line comments start with two forward slashes **//.**
Multi-line comments start with **/*** and ends with ***/.**

```
e.g.
public class Main {
  public static void main(String[] args) {
    // Here is a single-line comment
    System.out.println("Hello World");
    /* Here is a multi-line comment.
   This program will print our Hello World
```

```
   */
  }
}
// Output:  Hello World
```

? : syntax:

(bool-expression) ? (if-true-do-this) : (if-false-do-this);

// Run one of the statement according to the result of the boolean expression

e.g.(1)

```
int a=100; int b=300;
String result1 = (a<b) ? "orange" : "pineapple";
System.out.print ( result1);
// Output:  orange
```

e.g.(2)

```
int a=100; int b=300;
String result2 = (a>b) ? "orange" : "pineapple";
System.out.print ( result2 );
// Output:  pineapple
```

+ connect syntax:

str = str1 + str2;

str = str1.concat(str2);

// "+" can join two strings.

// concat() can connect two strings.

e.g.

```
String s1 = "Hello";  String s2 = "Friends";
System.out.print( s1 + s2);   // connect two strings
System.out.print( s1.concat (s2));   // connect two strings
// Output:  Hello Friends
// Output:  Hello Friends
```

---

abs() syntax:

Math.abs(number)

// Returns the absolute value of a number

e.g.

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.abs(-168.88));
    System.out.println(Math.abs(168.88));
  }
}
// Output:
168.88
168.88
```

---

abstract syntax:

abstract class/method

```
// "abstract" is used in class or method
// Abstract class works as a parent class, which will be extended by its subclass. The method of the subclass will override the abstract method of the abstract class.
// "abstract class" cannot create any object.
// "abstract method" is used in abstract class only
// "abstract method" is an empty method without a body.
```

e.g.

```
abstract class Book{     // create an abstract class
abstract void reading();     // create an abstract method
}
class eBook extends Book{  //  extend the parent class
void reading(){     // override the abstract method
System.out.println("I override that abstract method.");
}}
public class AbstractDemo {
public static void main(String[] args){
eBook obj =new eBook();
obj.reading();
}}
// Output:  I override that abstract method.
```

---

acos() syntax:

Math.acos(number)

```
// Return the arc cosine value of a number
e.g.
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.acos(0.88));
    System.out.println(Math.acos(-0.88));
    System.out.println(Math.acos(0));
}}
// Output:
0.4949341263408955
2.646658527248898
1.5707963267948966
```

---

add() syntax:

collection.add(element);

```
// Add an element to a collection
// Collection refers to LinkedList, ArrayList, HashSet and List.
e.g.
import java.util.HashSet;
import java.util.Set;
public class MyClass {
    public static void main(String[] args) {
        Set<Integer> mySet = new HashSet<>();
        //add integer values in this collection
        mySet.add(10);
        mySet.add(20);
        mySet.add(30);
        System.out.print("Collection elements : ");
        System.out.println(mySet);
}}
// Output:  Collection elements : [20, 10, 30]
```

---

addAll() syntax:

Collections.addAll(mySet, 10, 20, 30, 40, 50);

```
// Add all elements to a collection
```

```
// Collection refers to LinkedList, ArrayList, HashSet, List
e.g.
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
public class MyClass {
  public static void main(String[] args) {
    Set<Integer> mySet = new HashSet<>();
      Collections.addAll(mySet, 10, 20, 30, 40, 50);
      // add all elements to mySet
      System.out.println("Collection elements : "+mySet);
}}
// Output:  Collection elements : [50, 20, 40, 10, 30]
```

---

addExact() syntax:

Math.addExact(num1, num2)

```
// Returns the sum of two numbers
e.g.
class Main {
  public static void main(String[] args) {
    int x = 36;
    int y = 52;
    System.out.println(Math.addExact(x, y));
}}
// Output:  88
```

---

anonymous function syntax:

(parameter1, parameter2) -> expression

```
/*  Lambda expression is an anonymous function; actually it is a short code
block that can return a value. */
e.g.
import java.lang.FunctionalInterface;
interface MyClass{    // interface
    double piValue();   // abstract method
}
public class Main {
```

```
    public static void main( String[] args ) {
    MyClass refer;   // declare a reference to MyClass
    refer = ()  ->  3.1415;   // anonymous function
    System.out.println("The Pi value is " + refer.piValue());
}}
// Output:  The Pi value is 3.1415
```

---

arithmetic operator syntax:

Arithmetic operators mean +, - *, /, %, ++, --

```
// "%" gets a remainder.
// "++" increases 1
// "--" decreases 1.
e.g.
public class Main {
  public static void main(String[] args) {
    int x1 = 10; int y1 = 2;
    System.out.println(x1 + y1);     // 10+2=12
    int x2 = 10; int y2 = 2;
    System.out.println(x2 - y2);     // 10-2=8
    int x3 = 10; int y3 = 2;
    System.out.println(x3 * y3);     // 10*2=20
    int x4 = 10; int y4 = 2;
    System.out.println(x4 / y4);     // 10/2=5
    int x5 = 10; int y5 = 2;
    System.out.println(x5 % y5);     // 10%2=0
    int x6 = 10;
    System.out.println(++x6);     // 10+1=11
    int x7 = 10;
    System.out.println(--x7);     // 10-1=9
}}
```

---

array syntax:

int arrayName[ ] = {"value0", "value1", "value2",…};

```
// Create an array
e.g.
int arr[ ] = { 10,11,12,13 };     // create an array
```

// The above code creates an array named "arr", which has four elements: Number 0 element is arr[0] with value 10. Key is 0. Number 1 element is arr[1] with value 11. Key is 1. Number 2 element is arr[2] with value 12. Key is 2. Number 3 element is arr[3] with value 13. Key is 3. In the array, Key's alias is "index". Index's alias is "key". Note that index begins from 0.

---

array syntax:

```
int arrayName[ ] = new int [ number of elements ];
int arrayName[index0] = "value0";
int arrayName[index1] = "value1";
int arrayName[index2] = "value2";
```

// Create an array

e.g.

```
int arr[ ] = new int [3];   // create an array
arr [0] = 10;
arr [1] = 20;
arr [2] = 30;
```

// Above codes create an array, the array name is "arr", and it has three elements: arr [0], arr [1], arr [2]. Their indexes are 0, 1, and 2. Their values are 10, 20, and 30. Note that index number begins from 0.

---

array element syntax:

```
int value = array[index];
```

// Get a value from an element.

```
int array[index] = value;
```

// Set a value to an element.

e.g.

```
int myArr[ ] = {10, 20, 300, 40, 50};
int val = myArr[2];   // Get a value from an element.
System.out.println ( val );
```

// Output: 300

e.g.

```
int myNum[ ] = new int [5];
myNum[2] = 80;     // Set a value to an element.
System.out.println ( myNum[2]);
//  Output:  80
```

---

array length syntax:

```
arrayName.length;
// Return the total number of elements of an array.
e.g.
public class ArrLenClass{
public static void main (String [ ] args){
int myArr[ ] = {100,200,300};   // create an array "myArr"
int len = myArr.length;   // get the length of the array
System.out.print ( len );
}}
//  Output:  3
```

---

array sorting syntax:

```
Arrays.sort(array_name);
// Sort the array element, and display the result in order.
// Before using Arrays.sort(), you must "import java.util.Arrays";
e.g.
package tryPackage;
import java.util.Arrays;
public class SortArrClass{
public static void main (String [ ] args){
String arr[ ] = { "Cod", "App", "Dot", "Bay"};
Arrays.sort(arr);    // sort the array
for(int n=0; n<arr.length; n++){    // run 4 times
System.out.print (arr[n]+" ");     // n is a key
}}}
//  Output: App Bay Cod Dot
```

ArrayList syntax:

```
ArrayList myList = new ArrayList();
// Create an ArrayList object.
```

// The ArrayList class is a resizable array, the size of the ArrayList can be modified.
// An ArrayList is a dynamic data structure, it can iterate or search specified elements very fast in the list.
e.g.

```
import java.util.*;
public class MyArrayList{
public static void main(String[ ] args) {
ArrayList myList = new ArrayList();
// create an object of ArrayList
myList.add(new Integer(10));     // adds an element to myList
myList.add(new Integer(20));
myList.add(new Integer(30));
System.out.print(myList);
}}
```

// Output:
[10, 20, 30]
// Note:
The difference of LinkedList and ArrayList:
LinkedList runs fast in adding, inserting and removing elements.
ArrayList runs fast in iteration and searching element.

ArrayList functions syntax:

```
obj.add()      // add an element
obj.get()      // get an element value
obj.set()      // set an element value
obj.size()      // return the size of elements
obj.remove()    // remove an element
obj.clear()      // clear all elements
```

e.g.

```
import java.util.ArrayList;
public class Main {
  public static void main(String[] args) {
    ArrayList<String> colors = new ArrayList<String>();
    colors.add("Red");     // add an item
    colors.add("Yellow");
```

```
    colors.add("Green");
    System.out.println(colors);    // show item values
    for (int i = 0; i < colors.size(); i++) {    // loop through
    System.out.print(colors.get(i) + "  ");
     }
     System.out.println(colors.get(0));    // get an item value
     colors.set(1, "White");    // set an item value
     System.out.println(colors);
     colors.remove(2);    // remove an item value
     System.out.println(colors);
    System.out.println(colors.size());    // get ArrayList size
    colors.clear();    // clear all items
    System.out.println(colors);
}}
// Output:
[Red, Yellow, Green]
Red  Yellow  Green  Red
[Red, White, Green]
[Red, White]
2
[]
```

---

asin() syntax:

Math.asin(number)

```
// Return the arc sine value of a number
e.g.
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.asin(0.88));
    System.out.println(Math.asin(-0.88));
    System.out.println(Math.asin(0));
}}
// Output:
1.075862200454001
-1.075862200454001
0.0
```

---

assertion syntax:

assert condition: "message";

// Assertions help to detect any issues by checking code that we assume true. If the condition returns false, JVM will throw a message and stop running the program.

// When running the file with asserts code, we need to use command "java –ea filename" in command line to enable the assertions.

e.g.

```
import java.util.Scanner;
public class Assertion{
  public static void main( String args[] ){
      int value = 100;
      assert value > 100 : "The value is not greater than 100";
      System.out.println("The value is " + value);
}}
```

// Output: Exception in thread "main" java.lang.AssertionError: The value is not greater than 100

---

assignment operators

Assignment operators means =, +=, %=, >>=, ……

e.g.

```
public class Main {
  public static void main(String[] args) {
    int n1 = 6; n1 += 2;
    System.out.println(n1);     // n1 = 6+2, return 8
    int n2 = 6; n2 -= 2;
    System.out.println(n2);     // n2 = 6-2, return 4
    int n3 = 6; n3 *= 2;
    System.out.println(n3);     // n3 = 6*2, return 12
    int n4 = 6; n4 /= 2;
    System.out.println(n4);     // n4 = 6/2, return 3
    int n5 = 6; n5 %= 2;
    System.out.println(n5);     // n5 = 6%2, return 0
    int n6 = 6; n6 &= 2;
    System.out.println(n6);     // n6 = 6&2, return 2
```

```
    int n7 = 6; n7 |= 2;
    System.out.println(n7);     // n7 = 6|2, return 6
    int n8 = 6; n8 ^= 2;
    System.out.println(n8);     // n8 = 6^2, return 4
    int n9 = 6; n9 >>= 2;
    System.out.println(n9);     // n9 = 6>>2, return 1
    int n10 = 6; n10 <<= 2;
    System.out.println(n10);     // n10 = 6<<2, return 24
}}
```

---

atan() syntax:

Math.atan(number)

```
// Returns the arctangent value of a number between -PI/2 and PI/2 radians
e.g.
public class Main {
  public static void main(String[] args) {
    double x = 0.88;
    double y = 0.0;
    double z = 1.0;
    System.out.println(Math.atan(x));
    System.out.println(Math.atan(y));
    System.out.println(Math.atan(z));
}}
// Output:
0.7216548508647612
0.0
0.7853981633974483
```

---

atan2() syntax:

Math.atan2(y, x)

```
// Return the arc tangent of y / x in radians
e.g.
class Main {
  public static void main(String[] args) {
    double x = 4.2;
    double y = 5.9;
```

```
    double result = Math.atan2(y, x);
    System.out.println(result);     // the result is a radian
    System.out.println(Math.toDegrees(result));
    // convert to degree
}}
// Output:
0.9521519758701495
54.554289672401765
```

---

boolean syntax:

boolean variable

// A data type with true or false values

e.g.

```
    boolean good = true;
    boolean bad = false;
    System.out.println(good);
    System.out.println(bad);
// Output:  true,   false
```

BorderJFrame & BorderLayout Syntax:

BorderJFrame myFrame=new BorderJFrame() // create BorderJFrame

setLayout(new BorderLayout()) // set to use BorderLayout

add(new JButton(), BorderLayout.LOCATION) // add a button

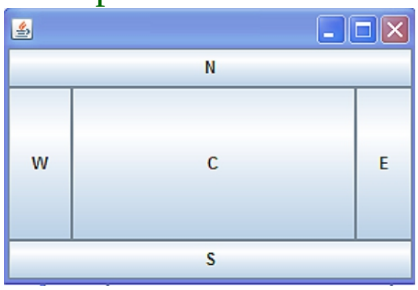
// Add a button to BorderJFrame. "LOCATION" means five locations: "East", "South", "West", "North", and "Center".

// The BorderLayout means that all components are placed in the five specified locations.

e.g.

```
import java.awt.* ;
import javax.swing.*  ;
public class BorderJFrame extends JFrame{
public BorderJFrame(){
setLayout(new BorderLayout());     // set layout
add(new JButton("N"), BorderLayout.NORTH );
```

```
add(new JButton("S"), BorderLayout.SOUTH );
add(new JButton("E"), BorderLayout.EAST );
add(new JButton("W"), BorderLayout.WEST );
add(new JButton("C"), BorderLayout.CENTER );
}     // add buttons
public static void main(String[] args){
BorderJFrame myFrame = new BorderJFrame();
// Create a BorderJFrame object
myFrame.setSize(300,200);
myFrame.setVisible(true);
}}
// Output:
```



---

break syntax:

```
break;
// Terminate a loop or a switch block
e.g.
for (int n = 1; n < 8; n++) {
  if (n == 6) {
    break;
  }
  System.out.print(n);
}
// Output:  12345
```

Button Event syntax:

```
implements ActionListener     // create an interface for listener
addActionListener(this)       // add a listener to a component
actionPerformed(ActionEvent e){…}    // run if an event occurs
```

// By implementing ActionListener, an interface can be created for a listener and listening for event.
e.g.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class ButtonFrame extends JFrame implements ActionListener{
// create an interface for listener
public ButtonFrame(){
JButton myButton = new JButton("Click Me");
myButton.addActionListener(this);     // add a listener to button
add(myButton);
}
public static void main(String[ ] args){
ButtonFrame myFrame =new ButtonFrame();
myFrame.setSize(300, 200);
myFrame.setVisible(true);
}
public void actionPerformed(ActionEvent e){
// actionPerformed() runs when an event occurs
System.out.println("Button Event Occurs!");
}}
```

Output: **Click Me** Button Event Occurs!

---

byte syntax:

byte variable

// A data type of the numbers from -128 to 127

e.g.

```
byte myNum = 88;
System.out.println(myNum);
// Output:  88
```

---

canRead() syntax:

file.canRead()

// Return true if the file can be read.

e.g.

```
File f = new File ( aFile.txt );    // create a file object
if (f.exists()) {
System.out.println("Is existing?"+ f.exists());
System.out.println("Can read?"+ f.canRead());
System.out.println("Can write?"+ f.canWrite());
System.out.println("Is a file?"+ f.isFile());
}
// Output:
Is existing? true
Can read? true
Can write? false
Is a file? true
```

---

canWrite() syntax:

file.canWrite()

// Return true if the file can be written.

e.g.

```
File f = new File ( aFile.txt );    // create a file object
if (f.exists()) {
System.out.println("Is existing?"+ f.exists());
System.out.println("Can read?"+ f.canRead());
System.out.println("Can write?"+ f.canWrite());
System.out.println("Is a file?"+ f.isFile());
}
// Output:
Is existing? true
Can read? true
Can write? false
Is a file? true
```

CardLayout Event syntax:

implements ActionListener // create an interface for listener

addActionListener(this) // add a listener to a component

actionPerformed(ActionEvent e){…} // run if an event occurs

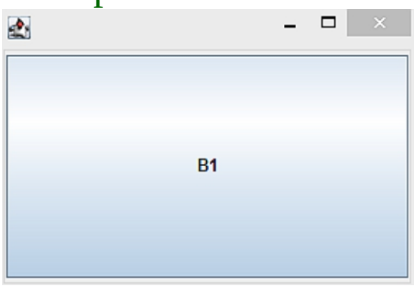
```
// By implementing ItemListener, an interface can be created for a listener and listening for event.
e.g.
import java.awt.*;
import java.awt.event.*;
class CardLayoutClass extends Frame implements ActionListener{
// create an interface for listener
    CardLayout myCard;
    Button button1, button3, button2;
    Panel panel1;
    CardLayoutClass(){
      myCard=new CardLayout();
      panel1=new Panel();
      panel1.setLayout(myCard);      // use CardLayout
      button1=new Button("first");
      button3=new Button("last");
      button2=new Button("next");
      for(int n = 1; n <= 3; n++){
      panel1.add(new Button("I am the Number "+ n + " Card"));
      }       // add button to panel1
      button1.addActionListener(this);      // add listeners
      button3.addActionListener(this);
      button2.addActionListener(this);
      Panel panel2=new Panel();
      panel2.add(button1);      // add button to panel2
      panel2.add(button2);
      panel2.add(button3);
      add(panel1,BorderLayout.CENTER);      // use BorderLayout
      add(panel2,BorderLayout.SOUTH);
      setBounds(10,10,200,190);
      setVisible(true);
}
public static void main(String args[]){
 new CardLayoutClass();
```

```
}
public void actionPerformed(ActionEvent e){
// actionPerformed() runs when an event occurs
     if(e.getSource()==button1){      // if click button1
       myCard.first(panel1);
        }
     else if(e.getSource()==button2){      // if click button2
      myCard.next(panel1);
      }
     else if (e.getSource()==button3){      // if click button3
     myCard.last(panel1);
}}}
// Output:     I am the Number 1 Card
```

CardJFrame & CardLayout Syntax:

```
CardJFrame myFrame=new CardJFrame();   // create a CardJFrame
setLayout(new CardLayout());     // set to use CardLayout
add(new JButton())    // add a button to CardJFrame
// The CardLayout means that a component works as a card; only one component is visible every time. CardLayout needs an event listener to show all components one by one.
e.g.
import java.awt.* ;
import javax.swing.*  ;
public class CardJFrame extends JFrame{
public CardJFrame(){
setLayout( new CardLayout(1,3));     // set layout
add(new JButton("B1"));     // add buttons
add(new JButton("B2"));
add(new JButton("B3"));
}
public static void main(String[] args){
CardJFrame myFrame = new CardJFrame();
// Create a CardJFrame object "myFrame"
myFrame.setSize(300,200);
myFrame.setVisible(true);
```

```
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```



---

case syntax:

case value:

```
// Marks code blocks in switch statements
e.g.
    int number = 2;
    switch (number) {
      case 1:
        System.out.println("Number 1");
        break;
      case 2:
        System.out.println("Number 2");
        break;
    }
// Output:  Number 2
```

---

cbrt() syntax:

Math.cbrt(number)

```
// Returns the cube root value of a number
e.g.
class Main {
  public static void main(String[] args) {
    double a = 125.0;
    double b = -27;
    double c = 0.0;
    System.out.println(Math.cbrt(a));
    System.out.println(Math.cbrt(b));
```

```
    System.out.println(Math.cbrt(c));
}}
// Output:
5.0
-3.0
0.0
```

---

ceil() syntax:

Math.ceil(number)

// Return a nearest integer that is greater than or equal to its argument.

e.g.

```
public class Main {
  public static void main(String[] args) {
    double num = 3.14;
    System.out.println(Math.ceil(num));
}}
// Output:  4.0
```

---

char syntax:

char variable

// A data type for a single character

e.g.

```
    char ch = 'C';
    System.out.println(ch);
// Output:  C
```

---

charAt() syntax:

string.charAt(index)

// Return a character at the specified index

e.g.

```
public class Main{
      public static void main(String []args){
          String str = "kindle";
          char c = str.charAt(2);
          System.out.println(c);
}}
```

// Output: n

Check Box Event

implements ItemListener // create an interface for listener
addItemListener(this) // add a listener to a component
itemStateChanged(ItemEvent e){…} // run if an event occurs
// By implementing ItemListener, an interface can be created for a listener and listening for event.
e.g.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
public class CheckBoxFrame extends JFrame implements
ItemListener{     // create an interface for listener
public CheckBoxFrame(){
JCheckBox myCheckBox = new JCheckBox("Check Me");
myCheckBox.addItemListener(this);     // add a listener
add(myCheckBox);
}
public static void main(String[ ] args){
CheckBoxFrame myFrame =new CheckBoxFrame();
myFrame.setSize(300, 200);
myFrame.setVisible(true);
}
public void itemStateChanged(ItemEvent e){
// itemStateChanged() runs when an event occurs
System.out.println("Check Box Event Occurs!");
}}
```

Output: □ **Check Me** Check Box Event Occurs!

---

class syntax:

modifier class ClassName {
type variable;

type methodName() { }…

```
}
// A class is a template for an object, and creates an object.
// "class ClassName" defines a class.
e.g.
public class MyClass {   // define a class
  int num = 100;
  public static void main(String[] args) {
    MyClass myObj = new MyClass();  // create an object
    System.out.println(myObj.num);
  }
}
// Output:  100
```

---

class & object syntax:

```
public class ClassName {     // define a class
}
ClassName obj = new ClassName(  );   // create an object
e.g.
public class Flower {     // define a class
String co1, co2;      // define two variables
void beautify() {      // define a method
System.out.println("This flower is " + co1);
System.out.println("That flower is " + co2);
}
public static void main(String[] args) {
Flower obj = new Flower(  );     // create an object
obj.co1= "red.";   obj.co2="blue.";     // obj references co1, co2
obj.beautify();      // obj references beautify()
}}
// Output:
This flower is red.
That flower is blue.
```

---

class attribute syntax:

```
public class MyClass {
  int variable1 = value1;
  int variable2 = value2;
}
// Create a class called "MyClass" with two attributes: variable1
and variable2
e.g.
public class Main {
  int num = 10;   // define a class property
  public static void main(String[] args) {
    Main obj = new Main();
    System.out.println(obj.num);
  }
}
// Output:  10
```

---

close() & AutoCloseable syntax:

```
public void close() throws Exception{
}
// "close()" is used to clean up the resource, it will be called automatically
when the object is closed or the block exits.
public class ClassName implements AutoCloseable{
}
// AutoCloseable interface provides the close() method.
System.gc()
// "gc()" suggests JVM to make a garbage collection.
// Note: It's not recommended to explicitly call System.gc().
e.g.
public class MyClass implements AutoCloseable {
    public static void main(String[] args) {
        try (MyClass obj = new MyClass()) {
            System.out.print("The object is no longer used!  ");
        } catch (Exception e) {
            System.out.println( e.getMessage() );
}}
    public void close() throws Exception {
```

```
        System.out.print("close() is executed!");
}}
// Output:     The object is no longer used!  close() is executed!
```

---

codePointAt() syntax:

string.codePointAt(index)

```
// Return a Unicode of a character at the specified index.
e.g.
public class Main{
      public static void main(String []args){
         String str = "kindle";
         int u = str.codePointAt(2);
         System.out.println(u);
      }
}
// Output:  110
```

---

codePointBefore() syntax:

string.codePointBefore(index)

```
// Return a Unicode of a character before the specified index.
e.g.
public class Main{
      public static void main(String []args){
         String str = "kindle";
         int u = str.codePointBefore(2);
         System.out.println(u);
}}
// Output:   105
```

---

codePointCount() syntax:

string.codePointCount(beginIndex, endIndex)

```
// Returns the number of Unicode values in the specified range of a String
e.g.
public class Main {
  public static void main(String[] args) {
    String str = "Hello World";
```

```
    int num = str.codePointCount(0, 8);
    System.out.println(num);
}}
// Output:  8
```

---

color object syntax:

```
Color myColor = new Color(r, g, b);     // create a color
// "RGB" means "red, green, blue", its value is from 0 to 255.
// "0,0,0" means black, "255,255,255" means white.
myButton.setBackground(myColor);     // set background color
myButton.setForeground(Color.color);     // set  foreground color
e.g.
import java.awt.* ;
import javax.swing.*  ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout(new FlowLayout());
JButton myButton = new JButton("R in 8 Hours");     // create a button
Font myFont = new Font("Arial", Font.ITALIC, 16);     // create a font
myButton.setFont(myFont);
Color myColor = new Color(0,0,0);     // create a color
myButton.setBackground(myColor);     // set background color
myButton.setForeground(Color.white);     // set  foreground color
add(myButton);      // add a button
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```

Combo Box Event syntax:

implements ItemListener // create an interface for listener
addItemListener(this) // add a listener to a component
itemStateChanged(ItemEvent e){…} // run if an event occurs
// By implementing ItemListener, an interface can be created for a listener and listening for event**.**
e.g**.**

```
import javax.swing.*;
import java.awt.FlowLayout;
import java.awt.event.*;
public class ComboBoxFrame extends JFrame implements
ItemListener{     // create an interface for listener
public ComboBoxFrame(){
JComboBox myComboBox = new JComboBox();
myComboBox.addItem(null);
myComboBox.addItem("Selected");
myComboBox.addItemListener(this);     // add a listener
setLayout( new FlowLayout());
add(myComboBox);
}
public static void main(String[ ] args){
ComboBoxFrame myFrame =new ComboBoxFrame();
myFrame.setSize(300, 200);
myFrame.setVisible(true);
}
public void itemStateChanged(ItemEvent e){
// itemStateChanged() runs when an event occurs
System.out.println("Combo Box Event Occurs!");
}}
// Output:
Combo Box Event Occurs!
```

---

comment syntax:

Single-line comments start with two forward slashes **//.**
Multi-line comments start with **/*** and ends with ***/.**
e.g**.**

```
public class Main {
```

```
  public static void main(String[] args) {
    // Here is a single-line comment
    System.out.println("Hello World");
    /* Here is a multi-line comment.
   This program will print our Hello World
   */
}}
// Output:  Hello World
```

---

compareTo() syntax:

str1.compareTo(str2)

```
// Compare two strings alphabetically based on Unicode value.
// Return zero if two strings are equal, return non-zero if two strings are not equal.
// If the first string is greater than the second string in Unicode value, returns a positive value.
// If the first string is less than the second string in Unicode value, returns a negative value.
e.g.
public class Main {
  public static void main(String[] args) {
    String str1 = "vacation";
    String str2 = "vocation";
    System.out.println(str1.compareTo(str2));
}}
// Output:  -14
```

---

compareToIgnoreCase() syntax:

str1.compareToIgnoreCase(str2)

```
// Compare two strings alphabetically based on Unicode value, the differences of the cases.
// Return zero if two strings are equal, return non-zero if two strings are not equal.
// If the first string is greater than the second string in Unicode value, returns a positive value.
```

// If the first string is less than the second string in Unicode value, returns a negative value.
e.g.

```
public class Main {
  public static void main(String[] args) {
    String str1 = "JQuery in 8 Hours";
    String str2 = "jQuery in 8 hours";
    System.out.println(str1.compareToIgnoreCase(str2));
  }
```

// Output: 0

---

concat() syntax:

str1.concat(str2)

// concatenate two strings

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str1 = "Python ";
    String str2 = "Syntax Book";
    System.out.println(str1.concat(str2));
}}
```

// Output: Python Syntax Book

---

conditional operator syntax:

(bool-expression) ? (if-true-do-this) : (if-false-do-this);

// Run one of the statement according to the result of the boolean expression

e.g.(1)

```
int a=100; int b=300;
String result1 = (a<b) ? "orange" : "pineapple";
System.out.print ( result1);
```

// Output: orange

e.g.(2)

```
int a=100; int b=300;
String result2 = (a>b) ? "orange" : "pineapple";
System.out.print ( result2 );
```

// Output: pineapple

---

constructor syntax:

```
modifier class ClassName{
ClassName(){…}      // define a constructor
}
```

// The constructor is used to initialize variable. Constructor name is the same as class name.

// When creating an object, the constructor will be called automatically

e.g.

```
public class Flower{   // define a class
String co1, co2;
public Flower(){co1="red";  co2="blue";}   // constructor
void beautify() {
System.out.println("This flower is " + co1);
System.out.println("That flower is " + co2);
}
public static void main(String[] args) {
Flower obj = new Flower(  );  // create an object, call constructor
obj.beautify();
}}
```

// Output:

This flower is red.

That flower is blue.

---

contains() syntax:

str1.contains(str2)

// Return true if a string contains a specified character or a sequence of characters, return false if not.

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "RayYao";
    System.out.println(str.contains("Ray"));
   System.out.println(str.contains("Yao"));
   System.out.println(str.contains("R"));
```

```
    System.out.println(str.contains("Hay"));
}}
// Output: true,  true,  true,  false
```

---

containsAll() syntax:

collection1.containAll(collection2);

// Return true if collection1 contains all elements of collection2, return false if conllection1 does not contain all elements of collection2.

e.g.

```
import java.util.ArrayList;
public class MyClass {
public static void main(String[] args) {
ArrayList<String> list1 = new ArrayList<>();
list1.add("A");
list1.add("B");
list1.add("C");
System.out.println("List1: " + list1);
ArrayList<String> list2 = new ArrayList<>();
list2.add("A");
list2.add("B");
System.out.println("List2: " + list2);
boolean bool1 = list1.containsAll(list2);
// check if List1 contains all element of List2
System.out.println("List1 contains all elements of List2? " + bool1);
boolean bool2 = list2.containsAll(list1);
// check if List2 contains all elements of List1
System.out.println("List2 contains all elements of List1? " + bool2);
}}
// Output:
List1: [A, B, C]
List2: [A, B]
List1 contains all elements of List2? true
List2 contains all elements of List1? false
```

---

contentEquals() syntax:

str1.contentEquals(str2)

// Return true if a string is equal to another string exactly, return false if not.
e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "RayYao";
    System.out.println(str.contentEquals("RayYao"));
    System.out.println(str.contentEquals("Yao"));
    System.out.println(str.contentEquals("Hay"));
}}
```

// Output: true, false, false

---

continue syntax:

continue;

// Continues to the next iteration of a loop
e.g.

```
for (int n = 1; n < 8; n++) {
  if (n == 6) {
    continue;
  }
  System.out.print(n);
}
```

// Output: 123457

---

copySign() syntax:

Math.copySign(x, y)

// Copy the sign of the second argument and assign the sign to the first argument.
e.g.

```
class Main {
  public static void main(String[] args) {
    float x = 1.68f;
    float y = -3.14f;
    System.out.println(Math.copySign(x, y));
}}
```

// Output:
-1.68

---

copyValueOf() syntax:

str1.copyValueOf(str2, beginIndex, endIndex)

// Return a String by copying the elements of an array.

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str1 = "";    // define an empty string
    char[] str2 = {'R', 'a', 'y', 'Y', 'a', 'o'};
    System.out.print(str1.copyValueOf(str2, 0, 6));
}}
```

// Output: RayYao

---

cos() syntax:

Math.cos(radians)

// Return the cosine value of a radian

e.g.

```
class Main {
public static void main(String[] args) {
double degr = 30;   // degree
double radi = Math.toRadians(degr);   // radian
System.out.println(Math.cos(radi));
}}
```

// Output:

0.8660254037844387

---

cosh() syntax:

Math.cosh(radian)

// Return a hyperbolic cosine value of the radian

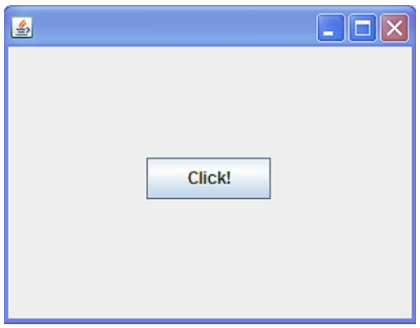
e.g.

```
class Main {
  public static void main(String[] args) {
    double degree1 = 60.0;   // degrees
    double degree2 = 0.0;
    double radian1 = Math.toRadians(degree1);   // radians
    double radian2 = Math.toRadians(degree2);
```

```
    System.out.println(Math.cosh(radian1));
    System.out.println(Math.cosh(radian2));
}}
// Output:
1.600286857702386
1.0
```

Customized Layout syntax:

```
JPanel myPanel = new JPanel();      // create a JPanel
myPanel.setLayout(null);      // use customized layout by setting "null"
add(myPanel);      // add myPanel to myFrame
myPanel.add(myButton);      // add button to myPanel
setBounds(x, y, width, height);
// set the bounds of JFrame or JPanel according to x, y coordinates.
// Customized Layout means that we can use Customized Layout by setting
the layout "null" instead of Java built-in Layout.
e.g.
import javax.swing.*;
import java.awt.*;
class myFrame extends JFrame{
myFrame(){    // constructor
JPanel myPanel = new JPanel();    // create a JPanel
myPanel.setLayout(null);     // use customized layout by setting "null"
add(myPanel);      // add myPanel to myFrame
myPanel.setBounds(50, 30, 150, 130);   // set bounds of myPanel
setBounds(200, 100, 300, 230);    // set bounds of myFrame
setVisible(true);     // set myFrame visible
JButton myButton = new JButton("Click!");
myPanel.add(myButton);      // add a button to myPanel
myButton.setBounds(50, 50, 90, 30);   // set bounds of myButton
}}
public class CustomizedLayout{
public static void main (String args[]){
new myFrame();
}}
// Output:
```



---

date object syntax:

LocalDate obj = LocalDate.now()

// Create a date object

e.g.

```
import java.time.LocalDate;   // import LocalDate class
public class Main {
  public static void main(String[] args) {
    LocalDate dateObj = LocalDate.now();   // create a date object
    System.out.println(dateObj);    // show the current date
}}
// Output:  2022-03-15
```

---

date & time syntax:

import java.time.LocalDateTime;

// import LocalDateTime class

LocalDateTime obj = LocalDateTime.now();

// create an object of local date and time

e.g.

```
import java.time.LocalDateTime;  // import LocalDateTime class
public class Main {
  public static void main(String[] args) {
    LocalDateTime obj = LocalDateTime.now();
    System.out.println(obj);
}}
// Output:  2022-03-15T10:38:06.933
```

---

decrementExact() syntax:

Math.decrementExact(num)

```
// Return the value after subtracting 1 to the num
e.g.
class Main {
  public static void main(String[] args) {
    int x = 100;
    System.out.println(Math.decrementExact(x));
    long y = 100000L;
    System.out.println(Math.decrementExact(y));
}}
// Output:
99
99999
```

---

default syntax:

```
default:
// Marks default code block in switch statements
e.g.
   int number = 100;
    switch (number) {
      case 1:
        System.out.println("Number 1");
        break;
      case 2:
        System.out.println("Number 2");
        break;
      default:
        System.out.println("Another Number");
   }
   // Output:  Another Number
```

---

default modifier syntax:

```
class MyClass
// There is no any modifier for the default class
// The default class is only accessible in the same package.
e.g.
class Member {     // default class
```

```
  String firstName = "Ray";
  String lastName = "Yao";
    public static void main(String[] args) {
    Member obj = new Member();
    System.out.println("Name: " + obj.firstName + " " + obj.lastName);
     }
}
// Output:  Name: Ray Yao
```

---

delete() syntax:

fileObj.delete()

```
// delete a file
e.g.
import java.io.File;
public class DeleteFile {
  public static void main(String[] args) {
    File obj= new File("myFile.txt");
    if (obj.delete()) {
      System.out.println("The file deleted is " + obj.getName());
    } else {
      System.out.println("Failed to delete the file.");
}}}
// Output:  The file deleted is myFile.txt
```

---

do / while syntax:

do{

……

}while(condition)

```
// "do" works with "while" composing a do/while loop.
e.g.
int n = 1;
```

do{

```
  System.out.print(n);
  n++;
```

}

while(n <= 5);

```
// Output: 12345
```

---

double syntax:

```
double
// A data type of the numbers from 1.7e−308 to 1.7e+308
e.g.
double num = 166.88d;
System.out.println(num);
// Output: 166.88
```

---

else syntax:

```
else{ }
// Used in conditional statements if the condition is false
e.g.
int score = 60;
if (score < 60) {
  System.out.println("Failed!");
} else {
  System.out.println("Passed!");
}
// Output: Passed!
```

---

encapsulation syntax:

```
public class Encapsulation {
  private String value;     // private attribute
  public String getValue() {   // Getter
    return value;
  }
  public void setValue(String newValue) {     // Setter
    this.value = newValue;
  }
}
// The meaning of encapsulation is to hide the sensitive data from users.
// Declare class variables/attributes as private
// Provide public getter and setter methods to access
e.g.
```

```
class Encap{
    private String name;
    public String getName(){     // getter
        return name;
    }
    public void setName(String newValue){     // setter
        name = newValue;
    }
}
public class Main{
    public static void main(String args[]){
        Encap obj = new Encap();
        obj.setName("Ray Yao");
        System.out.println("My name is " + obj.getName());
    }
}
// Output:  My name is Ray Yao
```

---

endswith() syntax:

str1.endswith(str2)

// Return true if a string ends with some specified characters, return false if not.

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "RayYao";
    System.out.println(str.endsWith("Ray"));
    System.out.println(str.endsWith("Yao"));
    System.out.println(str.endsWith("o"));
}}
// Output:  false,   true,   true
```

---

enum syntax:

enum EnumName{CONSTANTS}

// Declares an enumerated type class with multiple constants

// An enum class contains a group of constants

EnumName.CONSTANT // an Enum references a constant

e.g.

```
enum Flower {
  RED,
  YELLOW,
  GREEN
}
public class Main {
  public static void main(String[] args) {
    Flower flower = Flower.RED;
    System.out.println(flower);
 }}
// Output: RED
```

---

equals() syntax:

str1.equals(str2)

// Compare two strings, returns true if two strings are equal, returns false if not.

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str1 = "RayYao";
    String str2 = "RayYao";
    String str3 = "Java Programming";
    System.out.println(str1.equals(str2));
    System.out.println(str1.equals(str3));
}}
// Output: true,  false
```

---

equalsIgnoreCase() syntax:

str1.equalsIgnoreCase(str2)

// Compare two strings ignoring the case difference, returns true if two strings are equal, returns false if not.

e.g.

```
public class Main {
  public static void main(String[] args) {
```

```
    String str1 = "JQuery in 8 Hours";
    String str2 = "jQuery in 8 hours";
    System.out.println(str1.equalsIgnoreCase(str2));
  }
}
// Output:  true
```

---

exists() syntax:

file.exists()

```
// Return true if the file exists
e.g.
File f = new File ( aFile.txt );    // create a file object
if (f.exists()) {
System.out.println("Is existing?"+ f.exists());
System.out.println("Can read?"+ f.canRead());
System.out.println("Can write?"+ f.canWrite());
System.out.println("Is a file?"+ f.isFile());
}
// Output:
Is existing? true
Can read? true
Can write? false
Is a file? true
```

---

exp() syntax:

Math.exp(num)

// Returns the value of e^n

// Namely: Math.exp(n) = e^n

```
e.g.
class Main {
  public static void main(String[] args) {
    double n = 3.14d;
    System.out.println(Math.exp(n));
  }
```

```
}
// Output:  23.103866858722185
```

---

expm1() syntax:

Math.expm1(num)

// Return the value of e^n -1

// Namely Math.expm1(n) = e^n-1.

e.g.

```
class Main {
  public static void main(String[] args) {
    double n = 3.14d;
    System.out.println(Math.expm1(n));
  }
}
// Output:
22.103866858722185
```

---

exports syntax:

```
module a.module{
  exports a.package to b.module;
}
```

// Export a package to one or more specific modules so that the package can be accessed by the specified modules.

e.g.(1)

```
module java.base {    // from the module 'java.base'
      exports jdk.internal to jdk.jfr
}
// Export the package 'jdk.internal' to the module 'jdk.jfr'
```

e.g.(2)

```
module myModule {
    exports myPackage;
}
// Export myPackage so that it can be used outside of the myModule
```

---

extends syntax:

```
class ChildClass extends ParentClass
// A child class extends a parent class
// A child class inherits all features of a parent class.
e.g.
class Book {      // parent class
  protected String title = "R";
  public void fun() {
    System.out.print("Recommended Book: ");
}
}
class Ebook extends Book{     // child extends parent
  private String name = "in 8 Hours";
  public static void main(String[] args) {
    Ebook obj = new Ebook();
    obj.fun();
    System.out.print(obj.title + " " + obj.name);
}
}
// Output:  Recommended Book:  R in 8 Hours
```

---

file class syntax:

```
File myFile= new File(fileName);
// create a file object
e.g.
File f = new File ( aFile.txt );    // create a file object
if (f.exists()) {
System.out.println("Is existing?"+ f.exists());
System.out.println("Can read?"+ f.canRead());
System.out.println("Can write?"+ f.canWrite());
System.out.println("Is a file?"+ f.isFile());
}
// Output:
Is existing? true
Can read? true
Can write? false
```

Is a file? true

---

file creating syntax:

File obj= new File("myFile.txt");

// Create a file object

e.g.

```
import java.io.File;
import java.io.IOException;
public class Main {
  public static void main(String[] args) {
    try {
      File obj= new File("myFile.txt");
      if (obj.createNewFile()) {
        System.out.println("The file created is " + obj.getName());
      } else {
        System.out.println("The file already exists.");
      }
    } catch (IOException e) {
      System.out.println("Error occurred.");
      e.printStackTrace();
}}}
```

// Output: The file created is myFile.txt

---

file deleting syntax:

fileObj.delete()

// delete a file

e.g.

```
import java.io.File;
public class DeleteFile {
  public static void main(String[] args) {
    File obj= new File("myFile.txt");
    if (obj.delete()) {
      System.out.println("The file deleted is " + obj.getName());
    } else {
      System.out.println("Failed to delete the file.");
}}}
```

```
// Output:  The file deleted is myFile.txt
```

---

file reading syntax:

```
Scanner obj = new Scanner(fileObj);
// Create a file reading object
String content = obj.nextLine();
// Read each line of a file
e.g.
// Assume that the content in the myFile.txt is "Java in 8 Hours".
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    try {
      File fileObj = new File("myFile.txt");
      Scanner obj = new Scanner(fileObj);
      while (obj.hasNextLine()) {
        String content = obj.nextLine();    // file reading
        System.out.println(content);
      }
      obj.close();
    } catch (FileNotFoundException e) {
      System.out.println("Error occurred.");
      e.printStackTrace();
}}}
// Output:  Java in 8 Hours
```

---

file reading syntax:

```
FileReader obj = new FileReader(f);
// Create a file reading object
obj.read()
// Read a file
e.g.
import java.io.FileReader;
import java.io.IOException;
```

```
public class MyClass{
public static void main(String[] args) throws IOException{
String f = "D:\\myFile.txt";
// myFile.txt contents is: Hello World!
FileReader obj = new FileReader(f);
try {
int i;    // return -1 when reading the end of file
while((i = obj.read()) != -1) {     // read the file
System.out.print((char)i);
}} finally {
obj.close();
}}}
// Output:   Hello World!
```

---

file writing syntax:

```
FileWriter obj = new FileWriter("myFile.txt");
// Create a file writing object
obj.write("contents");
// Write a file
e.g.
import java.io.FileWriter;
import java.io.IOException;
public class Main {
  public static void main(String[] args) {
    try {
      FileWriter obj = new FileWriter("myFile.txt");
      obj.write("Java in 8 Hours");     // file writing
      obj.close();
      System.out.println("Write file successfully!");
    } catch (IOException e) {
      System.out.println("Error occurred.");
      e.printStackTrace();
}}}
// Output:  Write file successfully!
```

FileInputStream syntax:

```
FileInputStream obj = new FileInputStream(fileName);
obj.read();      //  reads the content of a file
// FileInputStream class is used to process input stream.
// "new FileInputStream (fileName)" creates an input stream.
// "obj.read();" reads the content of a file.
e.g.
(Given myFile.txt content is "PHP in 8 Hours!" in the D:\ disk)
import java.io.*;
public class InputFile {
public static void main(String[] args) {
int num;
try{
FileInputStream obj = new FileInputStream("D:\\myFile.txt");
while((num=obj.read())!= -1)  // read the file
System.out.print((char)num);  // change to characters
obj.close();
}
catch(FileNotFoundException e){
System.err.println(e);
}
catch(IOException e){
System.err.println(e);
}}}
// Output:  PHP in 8 Hours
// "obj" is an input stream object.
// "obj.read()" reads the content of myFile.txt
// "-1" indicates the end of file.
```

FileOutputStream syntax:

```
FileOutputStream obj = new FileOutputStream(fileName);
System.in.read();   // read the inputted data
obj.write();      // write data to the file
// FileOutputStream class is used to process Output stream.
// "new FileOutputStream (fileName)" creates an Output stream.
// "System.in.read();" reads the inputted data from keyboard.
// "obj.write();" writes data to the file.
```

e.g.
(Assume that we will write the file "myFile.txt" in the D:\ disk)

```
import java.io.*;
public class OutputFile {
public static void main(String[] args) {
char ch;
try{
FileOutputStream obj = new FileOutputStream("D:\\myFile.txt");
System.out.println("Please type a sentence, finish by Enter:");
// assume that we input "PHP in 8 Hours!" by keyboard
while((ch = (char)System.in.read())!= '\n')  /*read the input characters from keyboard*/
obj.write(ch);  // write characters to file
obj.close();
}
catch(FileNotFoundException e){
System.err.println(e);
}
catch(IOException e){
System.err.println(e);
}}}
// Output:  Please type a sentence, finish by Enter:
// Please input "PHP in 8 Hours!" by keyboard.
// Please check the myFile.txt in the D:\ disk, you will find that the content is "PHP in 8 Hours!".
```

FileReader syntax:

```
FileReader obj = new FileReader(f);
// Create a file reading object
obj.read()
// Read a file
```

e.g.

```
import java.io.FileReader;
import java.io.IOException;
public class MyClass{
public static void main(String[] args) throws IOException{
```

```
String f = "D:\\myFile.txt";
// myFile.txt contents is: Hello World!
FileReader obj = new FileReader(f);
try {
int i;    // return -1 when reading the end of file
while((i = obj.read()) != -1) {     // read the file
System.out.print((char)i);
}} finally {
obj.close();
}}}
// Output:   Hello World!
```

FileWriter syntax:

```
FileWriter obj = new FileWriter("myFile.txt");
// Create a file writing object
obj.write("contents")
// Write a file
e.g.
import java.io.FileWriter;
import java.io.IOException;
public class Main {
  public static void main(String[] args) {
    try {
      FileWriter obj = new FileWriter("myFile.txt");
      obj.write("Java in 8 Hours");     // file writing
      obj.close();
      System.out.println("Write file successfully!");
    } catch (IOException e) {
      System.out.println("Error occurred.");
      e.printStackTrace();
}}}
// Output:  Write file successfully!
```

---

final syntax:

```
final class/variable/method
```

// A modifier makes class/variable/method unchangeable, being inherited, and being overridden.
e.g.

```
public class Main {
  final int n = 100;     // make the variable n unchangeable
  public static void main(String[] args) {
    Main obj = new Main();
    obj.n = 200;   // error!
    System.out.println(obj.n);
  }
}
// Output:  ( error message… )
```

---

final class syntax:

final class
// "final" is a keyword.
// "final class" means that the class cannot be extended.
e.g.

```
final class FnlClass {   // declare a final class
……
}
class SubClass extends FnlClass {   // Error! Cannot extends
……
}
// Output:  Error message
```

---

final method syntax:

final method
// "final" is a keyword.
// "final method()" means that method cannot be overridden.
e.g.

```
class Animal{
final void study (){   // declare a final method
System.out.print("Animal");
}}
class Tiger extends Animal {
```

```
void study (){    // Error! Because study() want to override…
System.out.print("Tiger");
}}
// Output:  Error message
```

---

final variable syntax():

```
final variable
// "final" is a keyword.
// "final+ variable" means that variable value cannot be modified.
e.g.
public class FnlClass{
final int n =120;   // declare a final variable
static void study(  ){ n = 500;}   // Error! For "n" is a final variable
public static void main (String args[ ]){
System.out.print(  FnlClass.study() );
}}
// Output:  Error message
```

---

finally syntax:

```
finally{}
// Used in exception statement, must be executed in any condition
e.g.
public class Main {
  public static void main(String[] args) {
    try {
      int[] arr = {10, 20, 30};
      System.out.println(arr[3]);
    } catch (Exception e) {
      System.out.println("Error! Out of range!");
    } finally {
      System.out.println("Why? Learn Java in 8 Hours again!");
 }}}
// Output:
// Error! Out of range!
// Why? Learn Java in 8 Hours again!
```

---

float syntax:

float variable

// A data type of the numbers from 3.4e−038 to 3.4e+038

e.g.

```
float num = 1.68f;
System.out.println(num);
//  Output:  1.68
```

---

floor() syntax:

Math.floor(number)

// Return a nearest integer that is less than or equal to its argument.

e.g.

```
public class Main {
  public static void main(String[] args) {
    double num = 3.14;
    System.out.println(Math.floor(num));
  }
}
// Output:  3.0
```

---

font object syntax:

```
Font myFont = new Font("fontName", fontStyle, fontSize);
// Create a customized font
// "fontName" means the font such as "Arial", "Georgia", "Verdana"…
// "fontStyle" means the style such as "BOLD", "ITAIC", "PLAIN"…
myLabel.setFont(myFont);     // set the font of myLabel
```

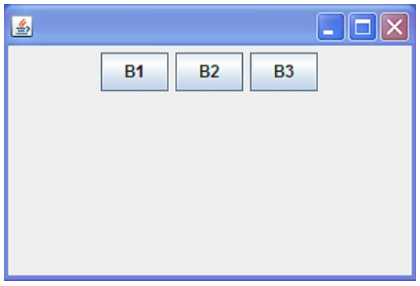
e.g.

```
import java.awt.* ;
import javax.swing.*  ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
JLabel myLabel = new JLabel("R in 8 Hours");     // create a label
Font myFont = new Font("Arial", Font.ITALIC, 16);    // create a Font
myLabel.setFont(myFont);     // set the font of myLabel
add(myLabel);      // add a label to JFrame
```

```
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```

FlowJFrame & FlowLayout Syntax:

```
FlowJFrame myFrame=new FlowJFrame();    // create a FlowJFrame
setLayout(new FlowLayout());     // set to use FlowLayout
add(new JButton())    // add a button to FlowJFrame
// The FlowLayout means that all components are placed from left to right,
and then to next line.
e.g.
import java.awt.* ;
import javax.swing.*  ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());      // set layout
add(new JButton("B1"));     // add buttons
add(new JButton("B2"));
add(new JButton("B3"));
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
// Create a FlowJFrame object "myFrame"
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```



---

for loop syntax:

for (initialization; condition; increment)

// Create a for loop

e.g.

```
for (int n = 1; n < 6; n++) {
        System.out.print(n);
}
// Output:  12345
```

---

format() syntax:

format("string %arg", value)

// Returns a formatted string by using the specified arguments.

// Argument 'd' specifies returning a digital value, 's' a string value, 'f' a float value, 'x' a hexadecimal value, 'c' a character value……

// About argument, please reference "String Format Chart".

e.g.

```
public class Main {
public static void main(String[] args) {
String s1 = String.format("%d", 100); // Digital value
String s2 = String.format("%s", "R in 8 Hours"); // String value
String s3 = String.format("%f", 2.99); // Float value
String s4 = String.format("%x", 100); // Hexadecimal value
String s5 = String.format("%c", 'W'); // Char value
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
System.out.println(s4);
System.out.println(s5);
}}
```

```
// Output:
100
R in 8 Hours
2.990000
64
W
```

---

gc() & finalize() syntax:

```
System.gc()     // Suggest JVM to make a garbage collection. However, it doesn't guarantee that the garbage collector will run immediately.
// Note: It's not recommended to explicitly call System.gc().
type void finalize(){…}     // finalize() was deprecated.
```

---

getbytes() syntax:

```
str.getBytes(Charset.forName("charset"));
// Encode a String by using a charset, return a new byte array.
e.g.
import java.util.Arrays;
import java.nio.charset.Charset;
public class Main {
  public static void main(String[] args) {
    String myStr = "Good";
    byte[] myArray;
    // using UTF-8 charset for encoding
    myArray = myStr.getBytes(Charset.forName("UTF-8"));
    System.out.println(Arrays.toString(myArray));
    // using UTF-16 charset for encoding
    myArray = myStr.getBytes(Charset.forName("UTF-16"));
    System.out.println(Arrays.toString(myArray));
  }
}
// Output:
[71, 111, 111, 100]
[-2, -1, 0, 71, 0, 111, 0, 111, 0, 100]
```

---

getChars() syntax:

str. getChars(int srcBeginIndex, int srcEndIndex, char[] dest, int destBeginIndex)

// Copy characters from a string into a character array.
// srcBeginIndex: the index of the first character in the string to copy
// srcEndIndex: the index of the last character in the string to copy
// char[] dest: the destination char array to copy to.
// destBeginIndex: the index in the array from where the character is copied into the array

e.g.

```
public class StringGetCharsExample{
public static void main(String args[]){
String str = new String("Shell Scripting in 8 Hours");
        char[] arr = new char[10];
        try{
           str.getChars(6, 15, arr, 0);
           System.out.println(arr);
        }catch(Exception ex){System.out.println(ex);}
}}
```

// Output: Scripting

---

getContentPane() syntax:

Container myContainer=myFrame.getContentPane();

// Create a container, to add some components to the new JFrame, we must create a container first.

myContainer.add(myButton); // add a button to myContainer

e.g.

```
import java.awt.* ;       // import Awt class for a container
import javax.swing.*  ;     // import Swing class for JButton
public class MyClass{
public static void main( String[ ] args){
JFrame myFrame = new JFrame("JFrame Title");
JButton myButton = new JButton("Button");     // create a JButton
Container myContainer = myFrame . getContentPane();   //
```

create a container

```
myContainer.add(myButton);    // add myButton to myContainer
```

```
myFrame.setSize(300,200);
myFrame.setVisible(true);
}}
// Output:
```


```
// Note:
Container myContainer = myFrame.getContentPane();
myContainer.add(myButton);
The above two lines of code can be substituted by one command:
add(myButton);      // add a  button to JFrame
```

---

getExponent() syntax:

getExponent(number)

```
//  Return the unbiased exponent from a number
e.g.
public class Main {
  public static void main(String[] args) {
    float x = 60.3f;
    System.out.println(Math.getExponent(x));
    double y = 82.9d;
    System.out.println(Math.getExponent(y));
  }
}
// Output:
5
6
```

---

getName() syntax:

file.getName()

```
// Get file name
```

e.g.

```
import java.io.*;
public class MyClass{
public static void main(String args[]) {
try {
File f = new File("D:\\hello.txt");   // create a file object
String name = f.getName();    // get file name
System.out.println("File name : " + name);  // show file name
}
catch (Exception e) {
System.err.println(e.getMessage());
}}}
// Output:   File name : hello.txt
```

---

getter syntax:

```
public class Encapsulation {
  private String value;     // private attribute
  public String getValue() {   // Getter
    return value;
  }
  public void setValue(String newValue) {     // Setter
    this.value = newValue;
  }
}
// The meaning of encapsulation is to hide the sensitive data from users.
// Declare class variables/attributes as private
// Provide public getter and setter methods to access
```

e.g.

```
class Encap{
    private String name;
    public String getName(){     // getter
        return name;
    }
    public void setName(String newValue){     // setter
        name = newValue;
    }
```

```
}
public class Main{
    public static void main(String args[]){
        Encap obj = new Encap();
        obj.setName("Ray Yao");
        System.out.println("My name is " + obj.getName());
    }
}
// Output:  My name is Ray Yao
```

---

global variable syntax:

```
type globalVariable =  value;
myMethod(){
}
// The global variable is defined outside a method. It works outside the current method.
e.g.
class MyClass {
int m= 1000, n = 2000;   //  global variable
void meth(){
int m = 10, n = 20;     //  local variable
}
}
// "int m= 1000, n = 2000;" defines two global variables outside the method.
```

GridBagJFrame & GridBagLayout syntax:

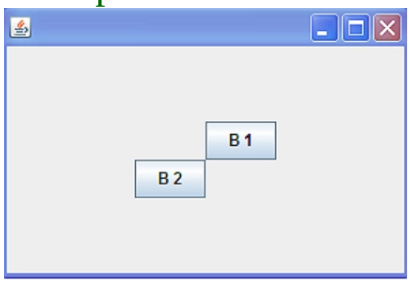
```
GridBagJFrame myFrame=new GridBagJFrame();   // create GridBagJFrame
setLayout(new GridBagLayout())     // set to use GridBagLayout
GridBagConstraints location = new GridBagConstraints();
// "location.gridx = row" puts a component in specified cell by row.
// "location.gridy = col" puts a component in specified cell by column.
add(new JButton(button), location)    // add button to GridBagJFrame
```

// The GridBagLayout means that all components are placed to the specified location according to row and column.

e.g.

```
import java.awt.* ;
import javax.swing.*  ;
public class GridBagJFrame extends JFrame{
public GridBagJFrame(){
setLayout(new GridBagLayout());    // set to use GridBagLayout
GridBagConstraints location = new GridBagConstraints();
// create a Constraints object to set the location of the buttons
location.gridx = 2;  location.gridy = 0;     // put the button by row & col
add(new JButton("B1"), location);    // add the button to GridBagJFrame
location.gridx = 0;  location.gridy = 2;
add(new JButton("B2"), location);
}
public static void main(String[] args){
GridBagJFrame myFrame=new GridBagJFrame();  // create GridBagJFrame
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
```

// Output:



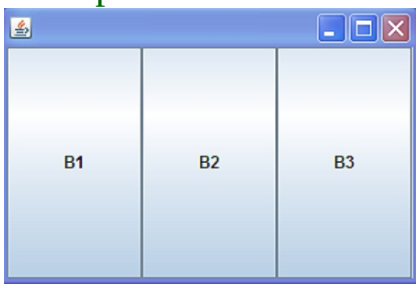
GridJFrame & GridLayout Syntax:

```
GridJFrame myFrame=new GridJFrame();    // create a GridJFrame
setLayout(new GridLayout(row, col))     // set to use GridLayout
add(new JButton())    // add a button to GridJFrame
```

// The GridLayout means that all components are placed with an arrangement of rows and columns.
e.g.

```
import java.awt.* ;
import javax.swing.*  ;
public class GridJFrame extends JFrame{
public GridJFrame(){
setLayout( new GridLayout(1,3));     // set layout
add(new JButton("B1"));     // add buttons
add(new JButton("B2"));
add(new JButton("B3"));
}
public static void main(String[] args){
GridJFrame myFrame = new GridJFrame();
// Create a GridJFrame object "myFrame"
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
```

// Output:



---

hashCode() syntax:

str.hashCode()

// Return a hash code of a String

e.g. (1)

```
public class Main {
  public static void main(String[] args) {
    String str = "good";
    System.out.println(str.hashCode());
  }
```

```
}
// Output:  3178685

object.hashCode();
// Return a hashcode of an object
// The hashCode() can create a number to represent an object.
// Note: Two equal objects should return the same hashcode.
e.g. (2)
public class MyHashCode{
public static void main(String[ ] args)  {
Object obj1 = new Object();
Object obj2 = obj1;
System.out.println(obj1.equals(obj2) );
System.out.println(obj1.hashCode());
// return the hashcode of object "obj1".
System.out.println(obj2.hashCode());
// return the hashcode of object "obj2".
}}
// Output:
true
31168322
31168322
--------------------------------------------------------------------------
                                HashMap syntax:
HashMap<Type, Type> obj = new HashMap<Type, Type>();
// Create a HashMap object
// HashMap is an array that contains key/value element
obj.keySet()     // return a Set of the HashMap keys
e.g.
import java.util.HashMap;
public class Main {
  public static void main(String[] args) {
 HashMap<String, String> colors = new HashMap<String, String>();
  // create a HashMap object
     colors.put("Apple", "Red");      // add a HashMap item
     colors.put("Banana", "Yellow");
```

```
    colors.put("Cashew", "Brown");
    System.out.println(colors);
    for (String fruit : colors.keySet()) {   // loop through HashMap
    System.out.print(fruit + " ");   // print all keys
   }
   System.out.println(" ");
   System.out.println(colors.get("Cashew"));   // get item value
   colors.remove("Banana");   // remove an item
   System.out.println(colors);
   System.out.println(colors.size());    // get item size
   colors.clear();    // clear all items
   System.out.println(colors);
  }
}
// Output:
{Apple=Red, Cashew=Brown, Banana=Yellow}
Apple Cashew Banana
Brown
{Apple=Red, Cashew=Brown}
2
{}
```

---

hasNext() syntax:

obj.hasNext()

```
// check if has next element in a collection
// Return true if the object has next element.
e.g.
import java.util.LinkedList;
import java.util.Iterator;
public class Main {
  public static void main(String[] args) {
    LinkedList<String> colors = new LinkedList<String>();
    colors.add("Red");     // add an item
    colors.add("Yellow");
    colors.add("Green");
    System.out.println(colors);    // show item values
```

```
    Iterator<String> it = colors.iterator();
    // create a LinkedList object
    while(it.hasNext()) {   // while it has next element, keep running
    System.out.println(it.next());
}}}
// Output:
[Red, Yellow, Green]
Red
Yellow
Green
```

HashSet syntax:

```
HashSet<String> colors = new HashSet<String>();
// Create a HashSet object
// The item of the HashSet is unique.
e.g.
import java.util.HashSet;
public class Main {
  public static void main(String[] args) {
  HashSet<String> colors = new HashSet<String>();
  colors.add("Red");     // add an item
  colors.add("Yellow");
  colors.add("Green");
  System.out.println(colors);    // show item values
  for (String v : colors) {  // loop through
  System.out.println(v);
  }
  System.out.println(colors.size());  // get size
  System.out.println(colors.contains("Yellow")); // check contain
  colors.remove("Green");   // remove the item
  System.out.println(colors);
  colors.clear();   // clear all items
  System.out.println(colors);
}}
// Output:
[Red, Yellow, Green]
```

Red
Yellow
Green
3
true
[Red, Yellow]
[ ]

HashSet mySet = new HashSet()
// HashSet is a collection which uses a hash table for storage unsorted elements. Hash table stores information by using a method called hashing. HashSet doesn't allow duplicate elements.
// "new HashSet()" creates an object of HashSet.
e.g.

```
import java.util.*;
public class MyHashSet{
public static void main(String args[ ]){
HashSet mySet = new HashSet();    // create an object of HashSet
mySet.add("Very");     // add an element "very" to mySet
mySet.add("Very");
mySet.add("Perfect");
mySet.add("Book");
System.out.println( mySet );
}}
```

// Output: [Book, Very, Perfect]
// Note:
TreeSet outputs sorted and unique values.
HashSet outputs unsorted and unique values.

---

hypot() syntax:

Math.hypot(x, y)

// Return the value of the sqrt(x^2 +y^2) without overflow.
e.g.

```
public class Main {
  public static void main(String[] args) {
    double a = 4.0;
```

```
    double b = 3.0;
    System.out.println(Math.hypot(a, b));
  }
}
// Output:  5.0
```

IEEEremainder() syntax:

Math.IEEEremainder(x,y)

```
// Return the remainder according to IEEE 754 standard.
e.g.
public class Main {
  public static void main(String[] args) {
    double x = 21.0;
    double y = 2.0;
    System.out.println(Math.IEEEremainder(x,y));
  }
}
// Output:  1.0
```

---

if syntax:

if (condition)

```
// Create a condition statement
e.g.
if (100 < 200) {
      System.out.println("100 is less than 200");
}
// Output:  100 is less than 200
```

---

if-else syntax:

if (bool-expression) { // if true do this; **}**

else { // if false do this; **}**

```
// "if...else statement" runs some code if a condition is true and another code
if the condition is false
e.g.
public class IfElseClass{
```

```
public static void main (String [ ] args){
int x=120; int y=500;
if (x>y) {    // if true do this
System.out.print ( "x is greater than y.");
}
else {    // if false do this
System.out.print ( "x is less than y");
}}}
//  x is less than y.
```

---

implement syntax:

```
class MyClass implement Interface
// "implement" is used in "interface"
// Interface is a special class, which will be implemented by a class.
// The interface contains one or more empty method that will be
implemented by a method of the class.
e.g.
interface Book{     // define an interface
public void myBook();   // define an empty method
}
class eBook implements Book{   // implement interface
public void myBook(){   // implement empty method
System.out.println("Implement an empty method.");
}}
public class InterfaceExmp {
public static void main(String[] args) {
eBook obj = new eBook();
obj.myBook();
}}
// Output:  Implement an empty method.
```

---

import syntax:

```
import Package/Class/Interface
// Used to import package, class, interface to current file
e.g.
import java.util.Scanner;   // import Scanner class
```

```
public class Main {
  public static void main(String[] args) {
   Scanner obj = new Scanner(System.in);
   // "System.in" is used to read the user input
     System.out.println("Please enter your name:");
     String name = obj.nextLine();
     System.out.println("My name is: " + name);
}}
// Output:
Please enter your name:
My name is: Ray Yao
```

---

incrementExact() syntax:

Math.incrementExact(num)

```
// Return the value after adding 1 to the num
e.g.
class Main {
  public static void main(String[] args) {
     int x = 100;
     System.out.println(Math.incrementExact(x));
     long y = 100000L;
     System.out.println(Math.incrementExact(y));
  }
}
// Output:
101
100001
```

---

indexOf() syntax:

str.indexOf('string') or str.indexOf('character')

```
// Return an index of a specified string or a character that is the first
occurrence in the string
e.g.
public class Main {
  public static void main(String[] args) {
     String str = "This book is a good book";
```

```
    System.out.println(str.indexOf("book"));
  }
}
// Output:  5
```

---

inheritance syntax:

```
class ChildClass extends ParentClass
// A child class extends a parent class
// A child class inherits all features of a parent class.
e.g.
class Book {      // parent class
  protected String title = "R";
  public void fun() {
    System.out.print("Recommended Book: ");
}
}
class Ebook extends Book{     // child extends parent
  private String name = "in 8 Hours";
  public static void main(String[] args) {
    Ebook obj = new Ebook();
    obj.fun();
    System.out.print(obj.title + " " + obj.name);
}
}
// Output:  Recommended Book: R in 8 Hours
```

---

input by user syntax:

```
Scanner obj = new Scanner(System.in);
String variable = obj.nextLine();
// Create a Scanner object
// "System.in" is used to read the user input
// Before creating a Scanner object, you need to import "java.util.Scanner"
class first.
e.g.
import java.util.Scanner;
public class Example{
```

```
  public static void main(String args[]){
      String myStr;
      Scanner obj = new Scanner(System.in);
      // "System.in" is used to read the user input
      System.out.println("Please input your name: ");
      myStr = obj.nextLine();    // read the input
      System.out.println("My name is: " + myStr);
  }
}
// Output:
Please input your name:
My name is: Ray Yao
```

---

instanceof syntax:

object instanceof class

// Return true if an object is an instance of a specific class

e.g.

```
public class Main {
  public static void main(String[] args) {
    Main obj = new Main();
    System.out.println(obj instanceof Main);
}}
// Output:  true
```

---

int syntax:

int variable

// A data type of the numbers from -2147483648 to 2147483647

e.g.

```
int num = 168168;
System.out.println(num);
// Output:  168168
```

---

interface syntax:

```
interface MyInterface(
emptyMethod();
}
```

class MyClass implement Interface

```
// Interface is a special class, which will be implemented by a class.
// The interface contains one or more empty method that will be
implemented by a method of the class.
e.g.
interface Book{     // define an interface
public void myBook();   // define an empty method
}
class eBook implements Book{   // implement the interface
public void myBook(){   // implement empty method
System.out.println("Implement an empty method.");
}}
public class InterfaceExmp {
public static void main(String[] args) {
eBook obj = new eBook();
obj.myBook();
}}
// Output:  Implement an empty method.
```

---

intern()

str.intern()

```
/* Used for obtaining the string from memory, which ensures that all the
same strings share the same memory.*/
public class InternExample2 {
    public static void main(String[] args) {
        String s1 = "RayYao";
        String s2 = s1.intern();
        String s3 = new String("RayYao");
        System.out.println(s1==s2);   // true
        System.out.println(s1==s3);   // false
        System.out.println(s2==s3);   // false
    }
}
// Output:  true,   false,   false
```

---

isEmpty() syntax:

str.isEmpty()

// Return true is a string is empty, return false if not.

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str1 = "";
    String str2 = "Good";
    System.out.println(str1.isEmpty());
    System.out.println(str2.isEmpty());
  }
}
```

// Output: true, false

---

isFile() syntax:

file.isFile()

// Return true if the file is a file

e.g.

```
File f = new File ( aFile.txt );    // create a file object
if (f.exists()) {
System.out.println("Is existing?"+ f.exists());
System.out.println("Can read?"+ f.canRead());
System.out.println("Can write?"+ f.canWrite());
System.out.println("Is a file?"+ f.isFile());
}
```

// Output:

Is existing? true

Can read? true

Can write? false

Is a file? true

---

iterator syntax:

Iterator<type> it = collection.iterator();

// Create a collection object

// Iterator object is used to loop through collection

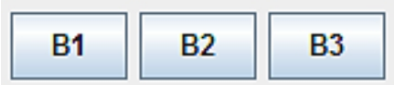
e.g.

```
import java.util.LinkedList;
import java.util.Iterator;
public class Main {
  public static void main(String[] args) {
    LinkedList<String> colors = new LinkedList<String>();
    colors.add("Red");     // add an item
    colors.add("Yellow");
    colors.add("Green");
    System.out.println(colors);    // show item values
    Iterator<String> it = colors.iterator();
    // create a LinkedList object
    while(it.hasNext()) {   // loop through
    System.out.println(it.next());
}}}
// Output:
[Red, Yellow, Green]
Red
Yellow
Green
```

JButton syntax:

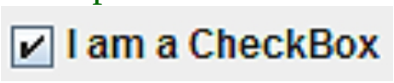
```
JButton myButton=new JButton("name");
// Create a JButton
e.g.
import java.awt.* ;
import javax.swing.*  ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JButton("B1"));      // create a button
add(new JButton("B2"));      // create a button
add(new JButton("B3"));      // create a button
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
```

```
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```



JCheckBox syntax:

```
JCheckBox myCheckBox = new JCheckBox("text");
// Create a JCheckBox with specified text
e.g.
import java.awt.* ;
import javax.swing.* ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JCheckBox("I am a CheckBox"));
// create a checkbox
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```



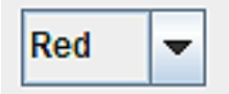
JComboBox syntax:

```
JComboBox myCombox = new JComboBox();   // create a JComboBox
addItem("text")     // addItem("text")
// The JComboBox provides a drop-down list to select.
// The drop-down list contains multiple options.
```

e.g.

```
import java.awt.* ;
import javax.swing.* ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
JComboBox color = new JComboBox();
color.addItem("Red");
color.addItem("Yellow");
color.addItem("Green");
add(color);    // add JComboBox to frame.
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
```

// Output:



JFrame syntax:

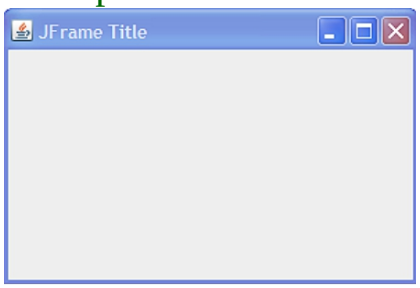
```
JFrame myFrame=new JFrame("JFrameName")   // create a JFrame
```

// To build a Graphical User Interface, we need to create a JFrame first.

e.g.

```
import javax.swing.*;       // import Swing class for JFrame
public class MyClass{
public static void main(String[]  args){
JFrame myFrame = new JFrame("JFrame Title");
// Create a JFrame object "myFrame"
myFrame.setSize(300,200);     // set myFrame size
myFrame.setVisible(true);     // set myFrame visible
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}      // set how to close myFrame
```

// Output:



JLabel syntax:

JLabel myLabel=new JLabel("Text");

// Create a JLabel

e.g.

```
import java.awt.* ;
import javax.swing.*  ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JLabel("Shell Scripting"));      // add a label
add(new JLabel("in"));      // add a label
add(new JLabel("8 Hours"));      // add a label
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
```

// Output:

Shell Scripting in 8 Hours

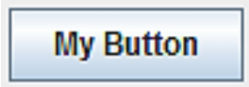
JPanel syntax:

JPanel myPanel = new JPanel();   // create a JPanel

// JPanel is a container. It can contain various components. But JPanel is also a component, it need to add to JFrame.

e.g.

```
import java.awt.* ;
import javax.swing.* ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JPanel());      // create  a panel and add to frame
add(new JButton("My Button"));      // add button to panel
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```



JPasswordField syntax:

```
JPasswordField myPassword=new JPasswordField(width);
// Create a JPasswordField. "width" specifies the max length.
import java.awt.* ;
import javax.swing.* ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JPasswordField (16));     // create a JPasswordField
}
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```

JRadioButton syntax:

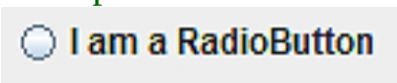
JRadioButton myRadio = new JRadioButton("text");

// Create a JRadioButton with specified text

e.g.

```
import java.awt.* ;
import javax.swing.* ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JRadioButton("I am a RadioButton"));
}     // create a JRadioButton
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
```

// Output:



JTextArea syntax:

JTextArea myTextArea = new JTextArea("text");

// Create a JTextArea with specified text

// JTextArea is used to accept multiple lines data inputted by user.

e.g.

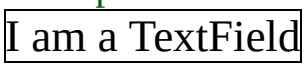
```
import java.awt.* ;
import javax.swing.* ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JTextArea("  I am a JTextArea!  "));
}     // create a textarea
```

```
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```

JTextField syntax:

JTextField myTextField = new JTextField("text");

```
// Create a JTextField with specified text.
// JTextField is used to accept one line data inputted by user.
e.g.
import java.awt.* ;
import javax.swing.*  ;
public class FlowJFrame extends JFrame{
public FlowJFrame(){
setLayout( new FlowLayout());
add(new JTextField(" I am a TextField "));
}      // create a textfield
public static void main(String[] args){
FlowJFrame myFrame = new FlowJFrame();
myFrame.setSize(300,200);
myFrame.setVisible(true);
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
// Output:
```



---

keySet() syntax:

obj.keySet()

```
// Return a set of the HashMap keys
e.g.
import java.util.HashMap;
```

```
public class MyClass {
  public static void main(String[] args) {
    HashMap<String, Integer> scores = new HashMap<>();
    scores.put("A", 100);
    scores.put("B", 60);
    scores.put("C", 0);
    System.out.println("HashMap: " + scores);
   System.out.println("HashMap Keys: " + scores.keySet());
   // return a set of hashmap keys
}}
// Output:
HashMap: {A=100, B=60, C=0}
HashMap Keys: [A, B, C]
```

---

lambda expression syntax:

(parameter1, parameter2) -> expression

```
/*  Lambda expression is an anonymous function; actually it is a short code
block that can return a value. */
e.g.
import java.lang.FunctionalInterface;
interface MyClass{   // interface
    double piValue();   // abstract method
}
public class Main {
    public static void main( String[] args ) {
    MyClass refer;  // declare a reference to MyClass
    refer = ()  ->  3.1415;   // lambda expression
    System.out.println("The Pi value is " + refer.piValue());
    }
}
// Output:  The Pi value is 3.1415
```

---

lastIndexOf() syntax:

str.lastIndexOf('string') or str.lastIndexOf('character')

```
// Return an index of a specified string or a character that is the last
occurrence in the string
```

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "This book is a good book";
    System.out.println(str.lastIndexOf("book"));
  }
}
// Output:  20
```

---

length() syntax:

str.length()

// Return a length of a string

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "Shell Scripting in 8 Hours";
    System.out.println(str.length());
  }
}
// Output:  26
```

LinkedList syntax:

LinkedList myList = new LinkedList();

// Create a LinkedList object

// The LinkedList class is a resizable array, the size of the LinkedList can be modified.

// A LinkedList is list in which each elements reference the next element in the list. Namely a LinkedList is ordered by index position; and allow adding or removing the element.

e.g.

```
import java.util.*;
public class MyLinkedList{
public static void main(String args[ ]){
LinkedList myList = new LinkedList();
// create an LinkedList object
myList.add("Hello! ");    // add an element to myList
```

```
myList.add("My ");
myList.add("Friend. ");
showList(myList);   // call function
}
static void showList(LinkedList myList){
Iterator i = myList.iterator();    // create an iterator object of "myList"
while(i.hasNext()){    // check if myList has next element
Object obj = i.next();
System.out.print(obj.toString());
}}}
// Output:  Hello! My Friend.
// Note:
The difference of LinkedList and ArrayList:
LinkedList runs fast in adding, inserting and removing elements.
ArrayList runs fast in iteration and searching element.
```

LinkedList functions syntax:

```
obj.addFirst("value");   // add first item
obj.addLast("value");   // add last item
obj.getFirst();   // get first item
obj.getLast());   // get last item
obj.removeFirst();   // remove first item
obj.removeLast();    // remove last item
e.g.
import java.util.LinkedList;
public class Main {
  public static void main(String[] args) {
  LinkedList<String> colors = new LinkedList<String>();
  colors.add("Red");     // add an item
  colors.add("Yellow");
  colors.add("Green");
  System.out.println(colors);    // show item values
  colors.addFirst("White");   // add first item
  colors.addLast("Black");   // add last item
  System.out.println(colors);
  System.out.println(colors.getFirst()); // get first item
```

```
  System.out.println(colors.getLast());  // get last item
  colors.removeFirst();  // remove first item
  colors.removeLast();   // remove last item
  System.out.println(colors);
}}
// Output:
[Red, Yellow, Green]
[White, Red, Yellow, Green, Black]
White
Black
[Red, Yellow, Green]
```

LocalDateTime syntax:

```
import java.time.LocalDateTime;
// import LocalDateTime class
LocalDateTime obj = LocalDateTime.now();
// create an object of local date and time
e.g.
import java.time.LocalDateTime;  // import LocalDateTime class
public class Main {
  public static void main(String[] args) {
    LocalDateTime obj = LocalDateTime.now();
    System.out.println(obj);
  }
}
// Output:  2022-03-15T10:38:06.933
```

---

local date syntax:

```
LocalDate obj = LocalDate.now()
// Create a date object
e.g.
import java.time.LocalDate;   // import LocalDate class
public class Main {
  public static void main(String[] args) {
    LocalDate dateObj = LocalDate.now();   // create a date object
    System.out.println(dateObj);    // show the current date
```

```
  }
}
// Output:  2022-03-15
```

LocalTime syntax:

LocalTime obj = LocalTime.now();

// create a time object

e.g.

```
import java.time.LocalTime;  // import LocalTime class
public class Main {
  public static void main(String[] args) {
    LocalTime obj = LocalTime.now();  // create a time object
    System.out.println(obj);   // show current time
  }
}
// Output:  13:56:28.939
```

---

local variable syntax:

myMethod(){

type localVariable = value;

}

// The local variable is defined inside a method. It works inside the current method.

e.g.

```
class MyClass {
int m= 1000, n = 2000;   //  global variable
void meth(){
int m = 10, n = 20;     //  local variable
}
}
// "int m = 10, n = 20;" defines two local variables inside a method
```

LocalDate syntax:

LocalDate obj = LocalDate.now()

// Create a date object

e.g.

```
import java.time.LocalDate;   // import LocalDate class
public class Main {
  public static void main(String[] args) {
    LocalDate myObj = LocalDate.now();   // create a date object
    System.out.println(myObj);    // show the current date
  }
}
// Output:  2022-03-15
```

---

log() syntax:

Math.log(number)

```
// Return the natural logarithm (base e) of a number
e.g.
public class Main {
  public static void main(String[] args) {
    // compute log() of 8.0
    System.out.println(Math.log(8.0));
}}
// Output:   2.0794415416798357
```

---

log10() syntax:

Math.log10(number)

```
// Return the base 10 logarithm of a number
e.g.
public class Main {
  public static void main(String[] args) {
    // compute log10() for a double value
    System.out.println(Math.log10(8.0));
    // compute log10() for zero value
    System.out.println(Math.log10(0.0));
}}
// Output:
0.9030899869919435
-Infinity
```

---

log1p() syntax:

Math.log1p(number)

```
// Return the natural logarithm (base e) of the sum of the number and 1
e.g.
public class Main {
  public static void main(String[] args) {
    // compute log1p() for a double value
    System.out.println(Math.log1p(8.0));
    // compute log1p() for zero value
    System.out.println(Math.log1p(0.0));
  }
}
// Output:
2.1972245773362196
0.0
```

---

long syntax:

long variable

```
// A data type of the numbers from -9223372036854775808 to
9223372036854775808
e.g.
long num = 16816800000L;
System.out.println(num);
// Output:  16816800000
```

---

matches() syntax:

str.matches('string with regular expression')

```
// Check whether or not a string matches the specified regular expression.
Return true if match, return false if not.
e.g.
import java.io.*;
public class Main {
public static void main(String args[]) {
String str = new String("Shell Scripting in 8 Hours");
System.out.println(str.matches("(.*)Scripting(.*)"));
System.out.println(str.matches("Scripting"));
System.out.println(str.matches("Shell(.*)"));
```

```
System.out.println(str.matches("(.*)Hours"));
}}    // (.*) means all characters and spaces
// Output: true,  false,  true,  false
```

---

max() syntax:

Math.max(num1, num2)

// Return the largest value between two numbers

e.g.

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.max(102, 168));
     // return the max value between 102 and 168
}}
// Output:  168
```

---

method syntax:

type methodName(){…..}; // declares a method.

methodName(); // calls a method.

// The Method is a code block that can be repeated using.

e.g.

```
public class MethClass{
public static void main(String args[ ]){
meth(); }    // call "meth"
static void meth(){   // declare a method
System.out.print( "Here is a method" );
}}
// Output: Here is a method
```

---

method & argument syntax:

type methodName (argument) {……};

methodName (arg);

// When calling a method, the arg can be passed to the method at the same time.

e.g.

```
public class MethArg{
```

```
public static void main ( String [ ] args ) {
meth( "with param" );   // call meth & pass an argument
}
static void meth(String argument){ // declare method
System.out.print( "Here is a method " + argument );
}}
// Output:  Here is a method with param
```

---

min() syntax:

```
Math.min(num1, num2)
// Return the smallest value between two numbers
e.g.
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.min(102, 168));
     // return the min value between 102 and 168
}}
// Output:   102
```

---

module syntax:

```
module myModuleName{
}
// Declares a module
// A Java module is the package of a Java application or Java API
e.g.(1)
module myModule{
   exports myPackage;
}
// Export myPackage so that it can be used outside of the myModule

e.g.(2)
module java.base{    // the module 'java.base'
      exports jdk.internal to jdk.jfr
}
// Export the package 'jdk.internal' to the module 'jdk.jfr'
```

---

multiple inheritance syntax:

```
class Parent{   }   // parent class
class Child1 extends Parent{  }   // child1 class
class Child2 extends Parent{  }   // child2 class
class Child3 extends Parent{  }   // child3 class
obj = new Child1();    // create a child1 object
obj = new Child2();   // create a child2 object
obj = new Child3();   // create a child3 object
// Multiple Inheritance means that a parent class can be extended by
multiple child classes.
// multiple Inheritance describes the ability to perform different method for
different objects.
e.g.
class Animals{    // parent class
void yell(){
System.out.println("yelling.....");;
}}
class Dogs extends Animals{   // child class
void yell(){     // overriding
System.out.println("Woww, Woww.....");
}}
class Cats extends Animals{   // child class
void yell(){    // overriding
System.out.println("Meaw, Meaw.....");
}}
public class PolymorphismClass{
public static void main(String args[ ]){   // main function
Animals animalObj;
animalObj = new Dogs();   // create a dog object
animalObj.yell();              // dog object calls yell()
animalObj = new Cats();    // create a cat object
animalObj.yell();              // cat object calls yell()
}}
// Output:
Woww, Woww……
Meaw, Meaw……
```

---

multiplyExact() syntax:

Math.multiplyExact(num1, num2)

// Return the product of num1 and num2

e.g.

```
class Main {
  public static void main(String[] args) {
    int x = 2;
    int y = 9;
    System.out.println(Math.multiplyExact(x, y));
  }
}
```

// Output: 18

---

native syntax:

```
public native void show();    // declare a native method
static{
System.loadLibrary("another_language");  // load another language
}
```

// A native method is a Java interface whose implementation program is written in another language such as C/ C ++.

// loadLibrary() loads the library written in another language.

e.g.

```
class NativeClass{
public native void show();    // declare a native method
static{     // load the library "nativeclass" written in C language
System.loadLibrary("nativeclass");
}
public static void main (String[] args){
NativeClass obj = new NativeClass();
obj.show();     // call the native method show()
}}
```

// A native method is an interface for java to reference non-Java code.

// A native method is implemented by a non-Java language code

---

negateExact() syntax:

negateExact(num)
// Return the reversed sign of the num
e.g.
```
class Main {
  public static void main(String[] args) {
    int x = 100;
    int y = -200;
    System.out.println(Math.negateExact(x));
    System.out.println(Math.negateExact(y));
}}
```
// Output:
-100
200

---

new syntax:

Class object = new Class();
// Create a new object
e.g.
```
public class Main {
  int num = 10;
  public static void main(String[] args) {
    Main obj = new Main();
    System.out.println(obj.num);
}}
```
// Output: 10

int arrayName[ ] = new int [number of elements];

// Create an new array
e.g.
```
int arr[ ] = new int [3];   // create an array
arr [0] = 10;
arr [1] = 20;
arr [2] = 30;
```
// Above codes create an array, the array name is "arr", and it has three elements: arr [0], arr [1], arr [2]. Their indexes are 0, 1, and 2. Their values

are 10, 20, and 30. Note that index number begins from 0.

---

next() syntax:

obj.next()

```
// access the next element in a collection
// Return the value of next element
e.g.
import java.util.LinkedList;
import java.util.Iterator;
public class Main {
  public static void main(String[] args) {
    LinkedList<String> colors = new LinkedList<String>();
    colors.add("Red");     // add an item
    colors.add("Yellow");
    colors.add("Green");
    System.out.println(colors);    // show item values
    Iterator<String> it = colors.iterator();
    // create a LinkedList object
    while(it.hasNext()) {   // while it has next element, keep running
    System.out.println(it.next());   // access the next element
}}}
// Output:
[Red, Yellow, Green]
Red
Yellow
Green
```

---

nextAfter() syntax:

Math.nextAfter(x, y)

```
// Return a number adjacent to x in the direction of y
e.g.
public class Main {
  public static void main(String[] args) {
    double x = 6.5f;
    double y = 9.8f;
    System.out.println(Math.nextAfter(x, y));
```

```
}}
// Output:  6.500000000000001
```

---

nextDown() syntax:

Math.nextDown(num)

```
// Return a number adjacent to the num in the direction to negative infinity.
e.g.
public class Main {
  public static void main(String[] args) {
    float num1 = 6.8f;
    System.out.println(Math.nextDown(num1));
    double num2 = 6.8d;
    System.out.println(Math.nextDown(num2));
}}
// Output:
6.7999997
6.799999999999999
```

---

nextLine() syntax:

```
Scanner obj = new Scanner(fileObj);
// Create a file reading object
String content = obj.nextLine();
// Read each line of a file
e.g.
// Assume that the content in the myFile.txt is "Java in 8 Hours".

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
   try {
      File fileObj = new File("myFile.txt");
      Scanner obj = new Scanner(fileObj);
      while (obj.hasNextLine()) {
        String content = obj.nextLine();    // file reading
```

```
        System.out.println(content);
      }
      obj.close();
    } catch (FileNotFoundException e) {
      System.out.println("Error occurred.");
      e.printStackTrace();
}}}
// Output: Java in 8 Hours
```

---

nextUp() syntax:

Math.nextUp(num)

```
// Return a number adjacent to the num in the direction to positive infinity.
e.g.
public class Main {
  public static void main(String[] args) {
    float num1 = 6.8f;
    System.out.println(Math.nextUp(num1));
    double num2 = 6.8d;
    System.out.println(Math.nextUp(num2));
}}
// Output:
6.8000007
6.800000000000001
```

---

now() syntax:

LocalDate obj = LocalDate.now()

```
// Create a date object
e.g.
import java.time.LocalDate;   // import LocalDate class
public class Main {
  public static void main(String[] args) {
    LocalDate dateObj = LocalDate.now();   // create a date object
    System.out.println(dateObj);    // show the current date
}}
// Output: 2022-03-15
```

---

object syntax:

obj = new ClassName(); // Create an object, an instance of a class.

obj.variable // the obj references a variable

obj.method() // the obj references a method()

e.g.

```
public class Flower {     // define a class
String co1, co2;      // define two variables
void beautify() {      // define a method
System.out.println("This flower is " + co1);
System.out.println("That flower is " + co2);
}
public static void main(String[] args) {
Flower obj = new Flower(  );     // create an object "obj"
obj.co1= "red.";   obj.co2="blue.";     // obj references co1, co2
obj.beautify();      // obj references a method beautify()
}}
// Output:
This flower is red.
That flower is blue.
```

---

offsetByCodePoints() syntax:

str.offsetByCodePoints(int index, int codePointOffset)

// Returns an index within a String, this index is the offset from the given index

// The offset is based on the codePointOffset's code points

// index: The index to be offset.

// codePointOffset: The offset in code points.

e.g.

```
import java.lang.*;
public class Main {
public static void main(String[] args) {
String str = "JavaScript";
System.out.println("The String is " + str);
int offset = str.offsetByCodePoints(3, 5);
```

```
System.out.println("The index is " + offset);
}}
// Output:
The String is JavaScript
The index is 8
```

---

overloading syntax:

```
public methodName(arg1){ }
public methodName(arg2){ }
// Overloading means that there are two or more same-name methods in a class, and their arguments are different.
e.g.
public class Flower{  String x, y;   // declare a class
public Flower ( String a, String b) { x=a; y=b; }   // overloading
public Flower () {  x="red ";  y="blue ";}    // overloading
public static void main (String args[ ]) {
Flower obj1=new Flower("pink ", "white ");
Flower obj2=new Flower(  );
System.out.println(obj1.x +  obj1.y);
System.out.println(obj2.x +  obj2.y);
}}
// Output:
pink  white
red  blue
```

---

overriding syntax:

```
public methodName(arg){ }     // this method in a parent class
public methodName(arg){ }     // this method in a child class
// The method of child class can override the method of the parent class, if the method name and argument are the same.
e.g.
class ParentdClass{
int meth( int n ) { return n; }   // overriding
}
class ChildClass extends ParentdClass{
int meth( int n ) { return 100 + n; }   // overriding
```

```
}
public class OverrideClass {
public static void main (String args[ ]){
ChildClass obj=new ChildClass();
System.out.print( obj.meth( 200 ));
}}
// Output:  300
```

---

package syntax:

```
package packageName
// Create a package
e.g.
package myPackage;
public class MyClass {
  public static void main(String[] args) {
    System.out.println("This is a package!");
  }
}
// Output:  This is a package!
```

---

pi syntax:

Math.PI

```
// Math.PI is a constant that stores the value of pi.
e.g.
public class PIRandom{
public static void main (String [ ] args){
double p = Math.PI;
System.out.println("The PI value is" + p);
}}
//  Output:  The PI value is 3.14159265358979
```

---

polymorphism syntax:

```
class Parent{   }    // parent class
class Child1 extends Parent{  }    // child1 class
class Child2 extends Parent{  }   // child2 class
```

```
obj = new Child1();    // create a child1 object
obj = new Child2();   // create a child2 object
// Polymorphism means that a parent class can be extended by multiple child classes.
// There are the same name methods in the different child class, which is known polymorphism.
e.g.
class Animals{    // parent class
void yell(){
System.out.println("yelling.....");;
}}
class Dogs extends Animals{   // child class
void yell(){     // overriding
System.out.println("Woww, Woww.....");
}}
class Cats extends Animals{   // child class
void yell(){    // overriding
System.out.println("Meaw, Meaw.....");
}}
public class PolymorphismClass{    // main class
public static void main(String args[ ]){   // main function
Animals animalObj;
animalObj = new Dogs();   // create a dog object
animalObj.yell();             // dog object calls yell()
animalObj = new Cats();    // create a cat object
animalObj.yell();             // cat object calls yell()
}}
// Output:
Woww, Woww……
Meaw, Meaw……
```

---

pow() syntax:

Math.pow(x,y)

// Return a value of the x raised to the power of the y.

e.g.

```
public class Main {
```

```
  public static void main(String[] args) {
    // computes 2 raised to the power 3
    System.out.println(Math.pow(2, 3));
}}
// Output:
8.0
```

---

print() syntax:

System.out.print (); // print the result in the same line.

System.out.println (); // print the result in the different line.

e.g.

```
System.out.print( "A " );
System.out.print( "B " );
System.out.print( "C"  );
// Output:  A B C
System.out.println( "A " );
System.out.println( "B " );
System.out.println( "C"  );
// Output:
A
B
C
```

---

private syntax:

private variable/method/constructor

// An access modifier. The private variables, methods or constructors can only be accessed within the declared class

e.g.

```
public class Student {
  private String name = "Andy";
  private String id = "007hero";
  private int phone = 12345678;
```

```
  public static void main(String[] args) {
    Student obj = new Student();
    System.out.println("Name: " + obj.name);
    System.out.println("ID: " + obj.id);
    System.out.println("Phone: " + obj.phone);
}}
// Output:
Name: Andy
ID: 007hero
Phone: 12345678
```

---

protected syntax:

protected variable/method/constructor

```
// An access modifier. The protected variables, methods or constructors can only be accessed  within the same package and subclasses
e.g.
class Student {
  protected String name = "Andy";
  protected String id = "007hero";
  protected int phone = 12345678;
}
public class Classmate extends Student {
  private int age = 18;
  public static void main(String[] args) {
    Classmate obj = new Classmate();
    System.out.println("Name: " + obj.name );
    System.out.println("ID: " + obj.id);
    System.out.println("Phone: " + obj.phone);
    System.out.println("Age: " + obj.age);
}}
// Output:
Name: Andy
ID: 007hero
Phone: 12345678
Age: 18
```

---

public syntax:

public variable/method/constructor

// An access modifier. The public variables, methods or constructors can be accessed by any classes

e.g.

```
class Student {
  public String name = "Andy";
  public String id = "007hero";
  public int phone = 12345678;
}
public class Classmate extends Student {
  private int age = 18;
  public static void main(String[] args) {
    Classmate obj = new Classmate();
    System.out.println("Name: " + obj.name );
    System.out.println("ID: " + obj.id);
    System.out.println("Phone: " + obj.phone);
    System.out.println("Age: " + obj.age);
}}
```

// Output:

Name: Andy

ID: 007hero

Phone: 12345678

Age: 18

Radio Button Event syntax:

implements ItemListener // create an interface for listener

addItemListener(this) // add a listener to a component

itemStateChanged(ItemEvent e){…} // run if an event occurs

// By implementing ItemListener, an interface can be created for a listener and listening for event.

e.g.

```
import javax.swing.*;
import java.awt.event.*;
public class RadioFrame extends JFrame implements ItemListener{     //
```

create an interface for listener

```
public RadioFrame(){
JRadioButton myRadio = new JRadioButton("Select Me");
myRadio.addItemListener(this);     // add a listener to radio
add(myRadio);
}
public static void main(String[ ] args){
RadioFrame myFrame =new RadioFrame();
myFrame.setSize(300, 200);
myFrame.setVisible(true);
}
public void itemStateChanged(ItemEvent e){
// itemStateChanged() runs when an event occurs
System.out.println("Radio Button Event Occurs!");
}}
// Output:      ⊙ Select Me          Radio Button Event Occurs!
```

---

random() syntax:

Math.random()

// Returns a random number between 0.0 and 1.0

e.g.

```
public class Main {
  public static void main(String[] args) {
    // generates a random number between 0.0 to 1.0
    System.out.println(Math.random());
  }
}
// Output:  0.5400266630143229
```

---

read() syntax:

FileReader obj = new FileReader(f);

// Create a file reading object

obj.read()

// Read a file

e.g.

```
import java.io.FileReader;
import java.io.IOException;
```

```
public class MyClass{
public static void main(String[] args) throws IOException{
String f = "D:\\myFile.txt";
// myFile.txt contents is: Hello World!
FileReader obj = new FileReader(f);
try {
int i;    // return -1 when reading the end of file
while((i = obj.read()) != -1) {     // read the file
System.out.print((char)i);
}} finally {
obj.close();
}}}
// Output:   Hello World!
```

---

readObject() syntax:

```
ObjectInputStream obj = new ObjectInputStream(streamObj);
// Create an ObjectInputStream obj
obj.readObject()  // Read data from obj stream
// readObject() is used to read data from an object stream.
e.g.
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
class MyClass {
public static void main(String[] args) {
String data = "Write to object stream. Read from object stream.";
try {
FileOutputStream file = new FileOutputStream("file.txt");
ObjectOutputStream obj1 = new ObjectOutputStream(file);
// Creates an ObjectOutputStream obj1
obj1.writeObject(data);  // write data to obj1 stream

FileInputStream streamObj = new FileInputStream("file.txt");
ObjectInputStream obj2 = new ObjectInputStream(streamObj);
// Create an ObjectInputStream obj2
```

```
System.out.print(obj2.readObject());  // Read data from obj2 stream
obj1.close(); obj2.close();
}
catch (Exception e){ e.getStackTrace();}
}}
// Output:  Write to object stream. Read from object stream.
```

---

recursion syntax:

```
myFunction(){
     myFunction();     // call myFunction()
}
// Recursion means that a function is called inside itself.
e.g.
public class Factorial {
    static int fact( int n ) {
        if (n != 0)     // stop condition
            return n * fact(n-1);    // recursive call
        else
            return 1;
    }
public static void main(String[] args) {
        int num = 5, result;
        result = fact(num);
        System.out.println(num + " factorial is " + result);
}}
// Output:  5 factorial is 120
```

---

regionMatches()

str1.regionMatches(int offset1, String str2, int offset2, int len)

```
// Check if two string regions are equal, return true if their regions are
equal, return false if not.
// offset1: the beginning offset of the str1.
// offset2: the beginning offset of the str2.
// len: the length for comparing
e.g.
import java.io.*;
```

```
public class Main {
    public static void main(String args[]) {
        String Str1 = new String("Shell Scripting in 8 Hours");
        String Str2 = new String("Scripting");
        String Str3 = new String("SCRIPTING");
        System.out.print("Return: " );
        System.out.println(Str1.regionMatches(6, Str2, 0, 9));
        System.out.print("Return: " );
        System.out.println(Str1.regionMatches(6, Str3, 0, 9));
}}
// Output:
Return: true
Return: false
```

Regular Expressions Syntax:

```
patt = Pattern.compile("pattern", Pattern.CASE_INSENSITIVE);
// Create a pattern, specify CASE_SENSITIVE or CASE_INSENSITIVE.
patt.matcher("string");
// Define a string that the pattern tries to match.
match.find()
// returns true if the string matches the patter, return false if not.
e.g.
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class MyClass{
public static void main(String[] args) {
Pattern patt = Pattern.compile("Shell", Pattern.CASE_INSENSITIVE);
Matcher match = patt.matcher("Shell Scripting in 8 Hours");
boolean found = match.find();
if(found) {
System.out.println("Found a match");
} else {
System.out.println("Not found a match");
}}}
// Output:   Found a match
```

---

remove() syntax:

```
collection.remove(index);
// Remove an element at specified index in a collection
e.g.
import java.util.ArrayList;
public class MyClass {
public static void main(String[] args) {
ArrayList<Integer> myList = new ArrayList<>();
myList.add(10);
myList.add(11);
myList.add(12);
System.out.println("MyList: " + myList);
int del = myList.remove(2);     // remove the element at index 2
System.out.println("Removed Element: " + del);
}}
// Output:
MyList: [10, 11, 12]
Removed Element: 12
```

---

removeAll() syntax:

```
collction.removeAll(collection);
// Remove all elements from a collection
e.g.
import java.util.ArrayList;
public class Main {
  public static void main(String[] args){
      ArrayList<String> myList = new ArrayList<>();
      myList.add("A");
      myList.add("B");
      myList.add("C");
      System.out.println("Original ArrayList: " + myList);
    // remove all elements from arraylist
      myList.removeAll(myList);    // remove all elements
      System.out.println("ArrayList after removeAll(): " + myList);
}}
// Output:
```

Original ArrayList: [A, B, C]
ArrayList after removeAll(): []
---

replace() syntax:

str.replace('old', 'new')
// Return a new string where a character is replaced by another character
e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "JavaScript";
    System.out.println(str.replace('a', 'e'));
}}
```

// Output: JeveScript
---

replaceFirst() syntax:

str.replaceFirst('regex', 'new')
// Replaces the first occurrence's substring which matches the specified regular expression with the new substring
e.g.

```
public class Main{
    public static void main(String[] args) {
        String str = "This book is brilliant!";
        String result = str.replaceFirst("[b]", "c");
        System.out.println(result);
}}
```

// Output: This cook is brilliant!
---

replaceAll() syntax:

str.replaceAll('regex', 'new')
// Replaces each substring of a string which matches the specified regular expression with the new substring.
e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "He sings 18 songs every 3 days";
    String result = "\\d+";
```

```
    System.out.println(str.replaceAll(result, " "));
  }   // replace the numbers with an empty string
}
// Output:  He sings   songs every   days
```

---

requires syntax:

requires module

```
// "requires" keyword is used to specify dependencies on other modules.
e.g.
module myModule {
    requires javafx.graphics;
}
// Require a standard Java module named javafx.graphics.
```

---

retainAll() syntax:

collection1.retainAll(collection2);

```
// Retain all common elements in two collections
e.g.
import java.util.ArrayList;
public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> list1 = new ArrayList<>();     // list1
        list1.add("A");
        list1.add("B");
        list1.add("C");
        System.out.println("ArrayList1: " + list1);
        ArrayList<String> list2 = new ArrayList<>();     // list2
        list2.add("B");
        list2.add("M");
        list2.add("W");
        System.out.println("ArrayList2: " + list2);
        list1.retainAll(list2);     // retain all common elements
        System.out.println("Common Elements: " + list1);
}}
// Output:
ArrayList1: [A, B, C]
```

ArrayList2: [B, M, W]
Common Elements: [B]

---

return syntax:

return value;
// return a value to the function caller
e.g.

```
public class Main {
  static int fun(int num) {
    return 10 + num;
  }
  public static void main(String[] args) {
    System.out.println(fun(20));    // fun(20) is a caller
}}
```

// Output: 30

---

rint() syntax:

Math.rint(num)
// Return a value that is nearest to the num and is equal to the mathematical integer
e.g.

```
class Main {
  public static void main(String[] args) {
    System.out.print(Math.rint(3.14) + " ");      // 3.0
    System.out.print(Math.rint(-1.8) + " ");      // -2.0
}}
```

// Output: 3.0 -2.0

---

round() syntax:

Math.round(num)
// Return the value of the num rounded to its nearest integer
e.g.

```
public class Main {
  public static void main(String[] args) {
    double num = 3.14;
    System.out.println(Math.round(num));
```

```
}}
// Output:  3
```

---

setter syntax:

```
public class Encapsulation {
  private String value;     // private attribute
  public String getValue() {   // Getter
    return value;
  }
  public void setValue(String newValue) {     // Setter
    this.value = newValue;
  }
}
// The meaning of encapsulation is to hide the sensitive data from users.
// Declare class variables/attributes as private
// Provide public getter and setter methods to access
e.g.
class Encap{
    private String name;
    public String getName(){     // getter
        return name;
    }
    public void setName(String newValue){     // setter
        name = newValue;
    }
}
public class Main{
    public static void main(String args[]){
        Encap obj = new Encap();
        obj.setName("Ray Yao");
        System.out.println("My name is " + obj.getName());
    }
}
// Output:  My name is Ray Yao
```

---

short syntax:

short variable

```
// A data type of the numbers from -32768 to 32767
e.g.
short num = 1000;
System.out.println(num);
// Output:  1000
```

---

signum() syntax:

Math.signum(num)

```
// Return the sign of the num
// Return -1.0 or 1.0 only
e.g.
public class Main {
  public static void main(String[] args) {
    double num1 = -3.14;
    System.out.println(Math.signum(num1));
    double num2 = 678;
    System.out.println(Math.signum(num2));
  }
}
// Output:
-1.0
1.0
```

---

sin() syntax:

Math.sin(radian)

```
// Return a sine value of a radian
e.g.
class Main {
public static void main(String[] args) {
double degr = 30;   // degree
double radi = Math.toRadians(degr);   // radian
System.out.println(Math.sin(radi));
}
}
// Output:   0.49999999999999994
```

---

sinh() syntax:

Math.sinh(radian)

// Return a hyperbolic sine value of the radian

e.g.

```
class Main {
  public static void main(String[] args) {
    double degree1 = 60.0;   // degrees
    double degree2 = 0.0;
    double radian1 = Math.toRadians(degree1);   // radians
    double radian2 = Math.toRadians(degree2);
    System.out.println(Math.sinh(radian1));
    System.out.println(Math.sinh(radian2));
  }
}
```

// Output:

1.2493670505239751

0.0

---

split() syntax:

str.split('delimiter/regex')

// Split a string into some substrings according to the specified delimiter or regular expression, return an array.

e.g.

```
public class Main{
public static void main(String []args){
String str = "ant, bat, cat, dog";
String[] arr = str.split(", ");    // ' , ' is a delimiter
for (int i=0; i < arr.length; i++){
System.out.println(arr[i]);   // print all elements of the arr
}}}
```

// Output: ant bat cat dog

---

sqrt() syntax:

Math.sqrt(num)

// Return a square root of the num

e.g.

```
class Main {
  public static void main(String[] args) {
    // compute square root of 9
    System.out.println(Math.sqrt(9));
  }
}
// Output: 3.0
```

---

startsWith() syntax:

str.startsWith("characters")

// Check if a string starts with the specified characters

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "JavaScript";
    System.out.println(str.startsWith("Java"));
    System.out.println(str.startsWith("Script"));
   }
}
// Output:  true    false
```

---

static variable syntax:

static variable

// A data type modifying variable or method, make them can be referenced by class directly, instead of by object.

// "static variable" is known as a class variable, it can be referenced by the class or object.

e.g.

```
public class StaClass {
static int num=10;    // declare a static variable
public static void main (String args[ ]){
System.out.print( StaClass.num );   /* StaClass references a static variable "num" directly  */
}}
// Output: 10
```

---

static method syntax:

static type method(){ }

// A data type modifying variable or method, make them can be referenced by class directly, instead of by object.

// "static method" is known as a class method, it can be called by the class or object.

e.g.

```
public class StaClass {
static int calculate() { return 10 ;}    // declare a static method
public static void main (String args[ ]){
System.out.print(  StaClass.calculate() );  /* StaClass calls a static method "calculate()" directly  */
}}
```

// Output: 10

---

strictfp syntax:

strictfp class/method

// "strictfp" is a modifier to restrict floating-point calculations in a class or method. It ensures that all floating-point operations across different Java platforms and provides consistent and identical result.

e.g.

```
public strictfp class Main {
    public static void main(String[] args){
        System.out.println(Double.MAX_VALUE);
    }
}
```

// Output: 1.7976931348623157E308

---

string syntax:

String variable = "value";

// String values must be enclosed by double quotes

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "Hello World";
```

```
    System.out.println(str);
  }
}
// Output:  Hello World
```

---

string length syntax:

int len = String.length();

// String.length() can get the length of a string.

e.g.

```
public class StrLen{
public static void main (String [ ] args){
String myStr = "Here is a string";
int len = myStr.length();   // get the string length
System.out.println( "The length of the string is " + len);
}}
// Output:  The length of the string is 16
```

---

string connection syntax:

str = str1 + str2;

str = str1.concat(str2);

// "+" can join two strings.

// concat() can connect two strings.

e.g.

```
String s1 = "Hello";  String s2 = "Friends";
System.out.print(s1 + s2);   // connect two strings
System.out.print(s1.concat (s2));   // connect two strings
// Output:  Hello Friends
// Output:  Hello Friends
```

---

stringBuffer syntax:

StringBuffer str = new StringBuffer();

// Create a StringBuffer object.

// StringBuffer represents a string whose value can be dynamically changed.

e.g.

```
StringBuffer str = new StringBuffer ( "abcde" );
```

str.insert(2, "1234"); // Output**:** abc1234de
str.append("5678"); // Output**:** abc1234de5678
str.reverse(); // Output**:** 8765ed4321cba
// "StringBuffer str = new StringBuffer ("abcde")" creates an object "str", its value is "abcde"**.**

---

subSequence() syntax:

str.subSequence(startIndex, endIndex)
// Returns a character sequence from a string
e.g**.**

```
public class Main {
  public static void main(String[] args) {
    String str = "Shell Scripting in 8 Hours";
    System.out.println(str.subSequence(6, 15));
  }
}
```

// Output**:** Scripting

---

subString() syntax:

str.substring(startIndex)
str.substring(startIndex, int endIndex)
// Return a substring of a string
e.g**.**

```
public class Main{
public static void main(String args[]){
String str="Shell Scripting in 8 Hours";
System.out.println(str.substring(6));
System.out.println(str.substring(6, 15));
}
}
```

// Output**:**
Scripting in 8 Hours
Scripting

---

subtractExact() syntax:

Math.subtractExact(num1, num2)

```
// Return the difference between num1 and num2
e.g.
class Main {
  public static void main(String[] args) {
    int x = 100;
    int y = 20;
    System.out.println(Math.subtractExact(x, y));
  }
}
// Output:  80
```

---

super syntax;

super.variable
super.method()

```
// "super" represents the object of the parent class
// "super" can reference parent class's variable or method.
e.g.
class Book {   // parent class
String str = "Ray Yao's Book:";
public void title() {
System.out.println("C# in 8 Hours");
}}
class Ebook extends Book {   // child class
public void title() {
System.out.println(super.str);  // reference parent class's variable
super.title();   // reference parent class's method
}}
public class Main {
public static void main(String[] args) {
Book myEbook = new Ebook();
myEbook.title();    // call the child class's method title()
}}
// Output:
Ray Yao's Book:
C# in 8 Hours
```

---

switch / case syntax:

```
switch (condition) {
case value1: statement; break;
case value2: statement; break;
case value3: statement; break;
}
// "switch" statement executes one of the "case" statements if the condition is satisfied
e.g.
int weekday = 2;
switch (weekday) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
}
// Output:  Tuesday
```

---

synchronized syntax:

```
synchronized(object)
// "synchronized(object)" can monitor a thread object so that only one thread can run at a time, other threads cannot interfere this thread running.
// Synchronized object prevents other thread interference
e.g.
class Number {
void printValue(String myThread) {
  synchronized(this) {     // synchronized the thread object
   for (int n = 1; n <= 3; n++) {
    System.out.println(myThread +  " of number " + n);
    try {
     Thread.sleep(300);
```

```
      }
      catch (InterruptedException e) {
       System.out.println("Thread has been interrupted.");
}}}}}
public class SynchronizedDemo {
public static void main(String[] args) {
  final Number number = new Number();
  Thread thr1 = new Thread("Thread 1") {     // thread 1
   public void run() {
    number.printValue(Thread.currentThread().getName());
   }};
  Thread thr2 = new Thread("Thread 2") {     // thread 2
   public void run() {
    number.printValue(Thread.currentThread().getName());
}};
  thr1.start();
  thr2.start();
}}
// Output:
Thread 1 of number 1
Thread 1 of number 2
Thread 1 of number 3
Thread 2 of number 1
Thread 2 of number 2
Thread 2 of number 3
```

---

system.in syntax:

```
System.in.read();   // read the inputted data
obj.write();      // write data to the file
// "System.in.read();" reads the inputted data from keyboard.
// "obj.write();" writes data to the file.
e.g.
(Assume that we will write the file "myFile.txt" in the D:\ disk)
import java.io.*;
public class OutputFile {
public static void main(String[] args) {
```

```
char ch;
try{
FileOutputStream obj = new FileOutputStream("D:\\myFile.txt");
System.out.println("Please type a sentence, finish by Enter:");
// assume that we input "PHP in 8 Hours!" by keyboard
while((ch = (char)System.in.read())!= '\n')  /*read the input characters from
keyboard*/
obj.write(ch);  // write characters to file
obj.close();
}
catch(FileNotFoundException e){
System.err.println(e);
}
catch(IOException e){
System.err.println(e);
}}}
// Output:   Please type a sentence, finish by Enter:
// Please input "PHP in 8 Hours!" by keyboard.
// Please check the myFile.txt in the D:\ disk, you will find that the content
is "PHP in 8 Hours!".
```

---

tan() syntax:

Math.tan(radian)

```
// Return a tangent value of the radian
e.g.
class Main {
  public static void main(String[] args) {
    double degree1 = 30;
    double degree2 = 45;
    double radian1 = Math.toRadians(degree1);
    double radian2 = Math.toRadians(degree2);
    System.out.println(Math.tan(radian1));
    System.out.println(Math.tan(radian2));
  }
}
// Output:
```

0.5773502691896257
0.9999999999999999

---

tanh() syntax:

Math.tanh(radian)

// Return a hyperbolic tan value of the radian

e.g.

```
class Main {
  public static void main(String[] args) {
    double degree1 = 60.0;   // degrees
    double degree2 = 0.0;
    double radian1 = Math.toRadians(degree1);   // radians
    double radian2 = Math.toRadians(degree2);
    System.out.println(Math.tanh(radian1));
    System.out.println(Math.tanh(radian2));
  }
}
```

// Output:

0.7807144353592677
0.0

Text Field Event syntax:

implements ActionListener // create an interface for listener

addActionListener(this) // add a listener to a component

actionPerformed(ActionEvent e){…} // run if an event occurs

// By implementing ActionListener, an interface can be created for a listener and listening for event.

e.g.

```
import javax.swing.*;
import java.awt.event.*;
public class TextFieldFrame extends JFrame implements
ActionListener{      // create an interface for listener
public TextFieldFrame(){
JTextField myTextField = new JTextField("Enter Data");
myTextField.addActionListener(this);     // add a listener
add(myTextField);
```

```
}
public static void main(String[ ] args){
TextFieldFrame myFrame =new TextFieldFrame();
myFrame.setSize(300, 200);
myFrame.setVisible(true);
}
public void actionPerformed(ActionEvent e){
// actionPerformed() runs when an event occurs
System.out.println("Text Field Event Occurs!");
}}
// Output:     Enter Data     Text Field Event Occurs!
```

---

this syntax:

this.variable / this.method()

```
// "this" represents the current object of its class
e.g.
public class Main {
  int num;
  public Main(int num) {     // constructor
    this.num = num;   // "this" represents "obj"
  }
  public static void main(String[] args)  {
    Main obj = new Main(100);
    System.out.println("The num is: " + obj.num);
}}
// Output:  The num is: 100
```

---

thread isAlive() syntax:

thread.isAlive()

```
// Check if the thread is alive
e.g.
public class Main extends Thread{
public void run(){
try{
Thread.sleep(1000);      // sleep 1000 milliseconds
```

```
System.out.println("Is the current thread alive after sleeping 1 second?"
+Thread.currentThread().isAlive());
}      // check if the current thread is alive
catch (InterruptedException ie) {
}}
public static void main(String[] args) {
Main thr = new Main();
System.out.println("Is the thread alive before starting? "+thr.isAlive());     //
check if the thread is alive before start()
thr.start();    // start the thread
System.out.println("Is the thread alive after starting? "+thr.isAlive());     //
check if the thread is alive after start()
}}
// Output:
Is the thread alive before starting?  false
Is the thread alive after starting?  true
Is the current thread alive after sleeping 1 second?  true
```

---

thread creating syntax:

```
ThreadClass threadObj = new ThreadClass();
// Create a thread object
threadObj.start();     // start running the thread
// Threads can be used to perform multiple tasks simultaneously.
// Each thread represents a subtask.
e.g. (1)
public class ThreadDemo extends Thread{  // extends
public void run(){
// the thread body
}}
public static void main(String[ ] args){
ThreadDemo threadObj = new ThreadDemo();    // create a thread
threadObj().start();    // start a thread
}

e.g. (2)
class ThreadDemo implements Runnable{  // implements
```

```
public void run(){
// the thread body
}}
public static void main(String[ ] args){
Thread threadObj = new Thread(new ThreadDemo());
threadObj.start();    // start a thread
}
// "run()" automatically runs when thread start.
// "threadObj = new ThreadDemo();" creates a thread object "threadObj".
// "start()" starts the thread.
```

---

thread extends Thread syntax:

```
public class MyClass extends Thread{  }
// MyClass needs to extend the Thread class before creating a thread object
and running thread tasks
e.g.
public class ThreadClass extends Thread{  // extends
public void run(){     // thread body
for(int n = 0; n<3; n++){
System.out.println("Thread!");
}}
public static void main(String[ ] args){
ThreadClass threadObj = new ThreadClass();
threadObj.start();      // start thread
}}
// Output:
Thread!
Thread!
Thread!
```

---

thread implements Runnable syntax:

```
public class MyClass implements Runnable{ }
// MyClass needs to implement Runnable before creating a thread object
and running thread tasks
e.g.
public class ThreadClass implements Runnable{ // implement
```

```
// implements
public void run(){     // thread body
for(int n = 0; n<3; n++){
System.out.println("Thread!");
}}
public static void main(String[ ] args){
Thread threadObj = new Thread(new ThreadClass());
threadObj.start();    // start thread
}}
// Output:
Thread!
Thread!
Thread!
```

---

throw syntax:

```
throw new Exception("message");
// Generate an error and error message.
e.g.
public class Main{
public static void main(String[] args)        {
System.out.println(100/0);
throw new ArithmeticException("/ by zero");
}}
// Output:
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at Main.main(Main.java:5)
```

---

throw syntax;

```
catch(ArithmeticException e){
    throw e;
}
// Catch an error from "try" and throw an exception
e.g.
public class Main{
    static void fun(){
        try{
```

```
            System.out.println(100/0);
        }
        catch(ArithmeticException e){
            throw e;    // throw an exception
        }}
    public static void main(String[] args){
            fun();
}}
// Output:
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at Main.fun(Main.java:4)
            at Main.main(Main.java:11)
```

---

throws syntax:

type method() throws exception{ }

// An exception may be thrown by a method

e.g.

```
public class Main{
static void fun() throws ArithmeticException {
System.out.println(100/0);
}
public static void main(String[] args){
fun();
}}
// Output:
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at Main.fun(Main.java:3)
            at Main.main(Main.java:6)
```

---

time syntax:

LocalTime obj = LocalTime.now();

// create a time object

e.g.

```
import java.time.LocalTime;  // import LocalTime class
public class Main {
  public static void main(String[] args) {
```

```
    LocalTime obj = LocalTime.now();  // create a time object
    System.out.println(obj);   // show current time
  }
}
// Output:  13:56:28.939
```

---

toArray() syntax:

collection.toArray()

```
// Convert a collection to an array
e.g.
import java.util.*;
public class MyClass{
public static void main(String[] args){
ArrayList myList=new ArrayList();
myList.add(10);
myList.add(20);
myList.add(30);
Object arr[]=myList.toArray();    // convert myList to array
for(Object item:arr){     // iterate all elements of array
System.out.print(item + " ");
}}}
// Output:   10  20  30
```

---

toCharArray() syntax:

str.toCharArray()

```
// Convert a string into a sequence of characters or a character array.
e.g.
public class Main {
public static void main(String args[]){
String str = "RayYao";
char[] c = str.toCharArray();
for (int i = 0; i < c.length; i++){
System.out.print(" " + c[i] + " ");
}}}
// Output:     R  a  y  Y  a  o
```

---

toDegrees() syntax:

Math.toDegrees(radian)

```
// Covert a radian to a degree
e.g.
class Main {
  public static void main(String[] args) {
    double radian1 = 0.5235987755982989;
    double radian2 = 1.0471975511965978;
    System.out.println(Math.toDegrees(radian1));
    System.out.println(Math.toDegrees(radian2));
  }
}
// Output:
30.0
60.0
```

---

toIntExact() syntax:

Math.toIntExact(long num)

```
// Convert a Long number to an Int number.
class Main {
  public static void main(String[] args) {
    long x = 2000000L;
    long y = -1000000L;
    int num1 = Math.toIntExact(x);
    int num2 = Math.toIntExact(y);
    System.out.println(num1);
    System.out.println(num2);
  }
}
// Output:
2000000
-1000000
```

---

toLowCase() syntax:

str.toLowCase()

```
// Convert the letters of a string to lower case.
```

e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "Shell Scripting in 8 Hours";
    System.out.println(str.toLowerCase());
  }
}
// Output:  shell scripting in 8 hours
```

---

toRadians() syntax:

Math.toRadians(degree)

// Covert a degree to a radian

e.g.

```
public class Main {
  public static void main(String[] args) {
    double degree1 = 30;
    double degree2 = 60;
    System.out.println(Math.toRadians(degree1));
    System.out.println(Math.toRadians(degree2));
  }
}
// Output:
0.5235987755982988
1.0471975511965976
```

---

toString() syntax:

num.toString() or Integer.toString(num)

// Return a String representing the value of the Number Object.

e.g.

```
public class Main {
   public static void main(String args[]) {
      Integer num = 10;
      System.out.println(num.toString());
      System.out.println(Integer.toString(20));
   }
}
```

```
// Output:  10    20
```
---

toUpperCase() syntax:

str.toUpperCase()

```
// Convert the letters of a string to upper case.
e.g.
public class Main {
  public static void main(String[] args) {
    String str = "Shell Scripting in 8 Hours";
    System.out.println(str.toUpperCase());
}}
// Output:  SHELL SCRIPTING IN 8 HOURS
```
---

transient syntax:

transient type variable;

```
// "transient" is a modifier which makes any variable not be serialized.
// "transient" can keep the confidential information of variable value.
// "transient" variable value will not be written to the specified file.
// Serialization is used to convert a variable to a byte stream.
e.g.
import java.io.*;
class Engineer implements Serializable{
int id;
String name;
transient int age;    //  it will not be serialized
public Engineer(int id, String name, int age) {
this.id = id;
this.name = name;
this.age = age;    // "age" will not be written to the file
}}
public class MyClass{
public static void main(String args[])throws Exception{
Engineer man =new Engineer(888, "Smith", 30);   // create an object
FileOutputStream myFile=new FileOutputStream("myFile.txt");
ObjectOutputStream out=new ObjectOutputStream(myFile);
out.writeObject(man);     // write to file except "age"
```

```
out.flush();
out.close();
myFile.close();
System.out.println("Very Good!");
}}
// Output:  Very Good!
```

TreeSet syntax:

TreeSet mySet = new TreeSet()

// TreeSet is a sorted collection, which provides a total sequence of its elements. TreeSet stores its elements in a tree structure and they are automatically arranged in a sorted order. TreeSet doesn't allow duplicate element.

// "new TreeSet()" creates an object of TreeSet.

e.g.

```
import java.util.*;
public class MyTreeSet{
public static void main(String args[ ]){
TreeSet mySet = new TreeSet();
mySet.add("Hello");    // add an element "Hello" to mySet.
mySet.add("Dear");
mySet.add("My");
mySet.add("Friend");
Iterator i = mySet.iterator();    // create an iterator object of "mySet"
while(i.hasNext())     // check if mySet has next element
System.out.print(" " + i.next());
}}
// Output:  Dear  Friend  Hello  My
```

// Note:

The output text is sorted alphabetically.

The sorted order is "Dear, Friend, Hello, My".

TreeSet does not allow the duplicated elements.

TreeSet outputs sorted and unique values.

HashSet outputs unsorted and unique values.

---

trim() syntax:

str.trim()
// Remove all spaces before and after a string
e.g.

```
public class Main {
  public static void main(String[] args) {
    String str = "       Ray Yao       ";
    System.out.println(str);
    System.out.println(str.trim());
  }
}
```

// Output:

```
            Ray Yao
Ray Yao
```

---

try / catch / finally syntax:

try {…}
catch(Exception e) {…}
finally {…}
// Used in the exception statement of the "try…catch…finally"
// "try" block may include some error codes
e.g.

```
    try {
      int[] arr = {0, 1, 2};
      System.out.println(arr[3]);
    } catch (Exception e) {
      System.out.print("Exception occurs! ");
  } finally {
     System.out.println(" Must run finally! ");
  }
```

// Output: Exception occurs! Exception occurs! Must run finally!

---

type casting / type converting syntax:

(type) variable
// Convert an old type to a specified new type.
e.g.

```
public class Main {
```

```
  public static void main(String[] args) {
    float myFloat = 3.14f;
    int myInt = (int) myFloat;
    System.out.println(myFloat);
    System.out.println(myInt);
}}
// Output:
3.14
3
```

---

ulp() syntax:

Math.ulp(num)

```
// Returns the size of the unit of least precision of the num
// "ulp" means: the unit of least precision
e.g.
public class Main{
    public static void main(String[] args){
        double num = 6.8;
        System.out.println(Math.ulp(num));
    }
}
// Output:  8.881784197001252E-16
```

---

user input syntax:

Scanner obj = new Scanner(System.in);

String variable = obj.nextLine();

```
// Create a Scanner object
// "System.in" is used to read the user input
// Before creating a Scanner object, you need to import "java.util.Scanner"
class first.
e.g.
import java.util.Scanner;
public class Example{
  public static void main(String args[]){
      String myStr;
      Scanner obj = new Scanner(System.in);
```

```
      // "System.in" is used to read the user input
      System.out.println("Please input your name: ");
      myStr = obj.nextLine();    // read the user input
      System.out.println("My name is: " + myStr);
  }
}
// Output:
Please input your name:
My name is: Ray Yao
```

---

valueOf() syntax:

type.valueOf(value)

```
// Convert different types of values into strings
e.g.
public class Main {
    public static void main(String args[]) {
        Integer i =Integer.valueOf(10);
        Double d = Double.valueOf(20);
        Float f = Float.valueOf(30);
        System.out.print(i + " " );
        System.out.print(d + " ");
        System.out.print(f + " ");
    }
}
// Output:  10    20.0    30.0
```

---

var syntax:

var variable = value;

```
// Define a variable of any data type.
// "var" can automatically detect the data type
e.g.
var v1 = 10;  // int
var v2 = 3.14;  // double
var v3 = 'C';  // char
var v4 = "R in 8 Horus";  // string
var v5 = true;   // boolean
```

```
        System.out.println(v1);
        System.out.println(v2);
        System.out.println(v3);
        System.out.println(v4);
        System.out.println(v5);
// Output:
10
3.14
C
R in 8 Hours
true
```

---

variable syntax:

```
type variableName = value;
// To define a variable, we must specify a type and assign a value to the variable.
e.g.
String book = "Python Syntax Book";   // 'book' is a variable
System.out.println(book);
// Output:
Python Syntax Book
```

---

variable syntax:

```
type variableName;
variable = value;
// To define a variable, we must specify a type and assign a value to the variable.
e.g.
int num;     // declare a variable 'num'
num = 100;    // assign a value to the variable
System.out.println(num);
// Output:  100
```

---

void syntax:

```
void method(){…}
// Specify a method that has no any return type
```

e.g.

```
public class Main {
  static void fun() {
    System.out.println("Scala in 8 Hours");
  }
  public static void main(String[] args) {
    fun();
  }
}
// Output:  Scala in 8 Hours
```

---

volatile syntax:

```
volatile type variable = value;
// "volatile" keyword allows multiple threads write & read in one variable.
// "volatile variable" is written & read directly by main memory, not cache.
// "volatile" keyword can ensure the security of the multiple threads.
```

e.g.

```
public class VolatileClass extends Thread{
volatile boolean isRunning = true;    // define a volatile variable
public void run() {    // run threads
long num=0;
while (isRunning) {
num++;
}
System.out.println("Thread is stopped." + num);
}
public static void main(String[] args) throws InterruptedException{
VolatileClass th = new VolatileClass();
th.start();   // start a thread
Thread.sleep(1800);
th.isRunning = false;
// variable "isRunning" can be written and read by other threads
th.join();
System.out.println("Because isRunning is " + th.isRunning);
}}
// Output:
```

Thread is stopped.1652909048
Because isRunning is false

---

while syntax:

```
while (condition){
……
}
```

// The while loop makes the code keeping running when the condition is true.

e.g.

```
int n = 1;
    while(n < 6) {
      System.out.print(n);
      n++;
   }
```

// Output: 12345

---

wrapper class syntax:

```
WrapperClass variable = value;   // create a wrapper object
```

// Wrapper class means Byte, Short, Integer, Long, Float, Double, Boolean, Character, ArrayList, LinkedList, HashMap, HashSet classes.

// We must use the wrapper class to create a wrapper object.

e.g.

```
public class Main {
  public static void main(String[] args) {
    Integer i = 100;       // Integer is a wrapper class
    Double d = 100.88;    // Double is a wrapper class
    String s = "good";    // String is a wrapper class
    System.out.println(i);
    System.out.println(d);
    System.out.println(s);
}}
```

// Output: 100 100.88 good

---

write() syntax:

```
FileWriter obj = new FileWriter("myFile.txt");
```

```
// Create a file writing object
obj.write("contents");
// Write a file.
e.g.
import java.io.FileWriter;
import java.io.IOException;
public class Main {
  public static void main(String[] args) {
    try {
      FileWriter obj = new FileWriter("myFile.txt");
      obj.write("Java in 8 Hours");     // file writing
      obj.close();
      System.out.println("Write file successfully!");
    } catch (IOException e) {
      System.out.println("Error occurred.");
      e.printStackTrace();
}}}
// Output:  Write file successfully!
```

---

writeObject() syntax:

```
ObjectOutputStream obj = new ObjectOutputStream(file);
// Creates an ObjectOutputStream obj
obj.writeObject(data);  // write data to obj stream
// writeObject() is used to write data to an object stream
e.g.
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
class MyClass {
public static void main(String[] args) {
String data = "Write to object stream. Read from object stream.";
try {
FileOutputStream file = new FileOutputStream("file.txt");
ObjectOutputStream obj1 = new ObjectOutputStream(file);
// Creates an ObjectOutputStream obj1
```

```
obj1.writeObject(data);  // write data to obj1 stream
FileInputStream streamObj = new FileInputStream("file.txt");
ObjectInputStream obj2 = new ObjectInputStream(streamObj);
// Create an ObjectInputStream obj2
System.out.print(obj2.readObject());  // Read data from obj2 stream
obj1.close(); obj2.close();
}
catch (Exception e){ e.getStackTrace();}
}}
// Output:   Write to object stream. Read from object stream.
```

Appendix

Java Keywords Chart

abstract	boolean	break
byte	case	catch
char	class	const
continue	default	do
double	else	extends
false	final	finally
float	for	if
implements	import	int
interface	new	null
package	private	protected
public	return	static
String	super	switch
this	throw(s)	true
try	void	while

Data Type Chart

Data Type	**Size**	**Description**
byte	1 byte	from -128 to 127
short	2 bytes	from -32,768 to 32,767
int	4 bytes	from -2,147,483,648 to 2,147,483,647
long	8 bytes	+ or - integer exceeding 2.14 billion
float	4 bytes	fractional numbers with decimal digits
double	8 bytes	fractional numbers with 15 decimal digits
boolean	1 bit	true or false values
char	2 bytes	a single character/letter or ascii values

Arithmetic Operators Chart

Operator	Name	Example
+	addition	x + y
-	subtraction	x - y
*	multiplication	x * y
/	division	x / y
%	modulus	x % y
++	increment	++x // increase 1
--	decrement	--x // decrease 1

Assignment Operators Chart

Operator	Example	Same As
=	x = 8	x = 8
+=	x += 8	x = x + 8
-=	x -= 8	x = x - 8
*=	x *= 8	x = x * 8
/=	x /= 8	x = x / 8
%=	x %= 8	x = x % 8
&=	x &= 8	x = x & 8
|=	x |= 8	x = x | 8
^=	x ^= 8	x = x ^ 8
>>=	x >>= 8	x = x >> 8
<<=	x <<= 8	x = x << 8

Comparison Operators Chart

Operators	Running
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
==	equal
!=	not equal

Logical Operators Chart

Operators	Equivalent
&&	and
\|\| | or |
! | not |

Logical Result Chart

true && true; returns true;	true && false; returns false;	false &&false; returns false;
true II true; returns true;	true II false; returns true;	false II false; return false;
! false; returns true;	! true; returns false;	

Other Operators Chart

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise OR
? :	test ? true-do-this : false-do-this

Overloading & Overriding Chart

overloading	same method name, different argument, overloading in the same class.
overriding	same method name, same argument, overriding in between super class and sub class.

Math Methods

Method	Returns
abs()	a number's absolute value
ceil()	an integer greater than or equal to its argument
cos()	an angle's trigonometric cosine
exp()	a number's Math.E to the power
floor()	an integer less than or equal to its argument.
log()	a number's natural logarithm
random()	a random positive value from 0 to 1
max()	a greater between two numbers
min()	a smaller between two numbers
pow()	the first argument to the power of the second
round()	the closest long or int
sin()	an angle's trigonometric sine
sqrt()	a number's square root
tan()	an angle's trigonometric tangent

Collection Function Chart

Functions	Descriptions
add()	add element
addAll()	add all elements
remove()	remove element
removeAll()	remove all elements
retainAll()	retain all elements
contains()	check if contain element
containsAll()	check if contain all elements
toArray()	convert element to array
keySet()	return a Set of the HashMap keys
next()	get next element value
hasNext()	check if data has next element
equals()	check if two objects are equal
hashCode()	create a hashcode of an object

Default, Public, Private, Protected Chart

Java has four permission modifiers to define a member.
The access permissions of public, protected, default, private are:

accessed by:	public	protect	default	private
same class in same package	ok	ok	ok	ok
parent & child class in same package	ok	ok	ok	
parent & child class in different packages	ok	ok		
non-parent & non-child class in different packages	ok			

"public" member can be accessed in any packages & classes.

"protected" member cannot be accessed in non-parent class or non-child class.

"default" member cannot be accessed in different packages.

"private" member cannot be accessed in different classes.

Abstract Chart

Abstract Structure:

```
abstract class books{    //define an abstract class
abstract int read();   //define an abstract method
}
class eBooks extends books{    //extend the parent class
int read (){ return …; }    //override the abstract method
}
```

Interface Chart

Interface Structure:

```
interface books {    // define an interface
int title();    // define an empty method
}
class eBooks implement books {    // implement the interface
public int title() {    // implement the empty method
return …;
}
}
```

Thread Methods Chart

Method	Description
start()	starts the thread.
stop()	stops the thread.
sleep()	makes the thread sleep for some time.
wait()	makes the thread in waiting status.
notify()	wakes up the thread that is waiting.
yield()	lets other threads with same priority run.
isAlive()	tests the current thread alive.
currentThread()	gets the current thread.
interrupt()	stops the "blocked" thread.
suspend()	makes the thread pause.

resume()	starts the thread again**.**
join()	runs this thread after another thread ended**.**

Modifiers Chart

Modifier	Description
final	cannot be overridden/modified
static	belongs to the class, rather than an object
abstract	The method does not contain a body.
transient	Skipped when serializing the object containing them
synchronized	Can only be accessed by one thread at a time
volatile	Not cached thread-locally, read from "chief memory"

Date & Time Class Chart

Class	Description
LocalDate	a date (yyyy-MM-dd)
LocalTime	a time (HH-mm-ss-ns))
LocalDateTime	datetime (yyyy-MM-dd-HH-mm-ss-ns)
DateTimeFormatter	a formatter for date-time objects

Event, Listener, Methods Chart

Event	Event Listener	Handling Method
JButton JTextField JPasswordField	ActionListener	actionPerformed()
JCheckBox JRadioButton JComboBox JMenuItem	ItemListener	itemStateChanged()
keyboard	keyListener	keyTyped()
Mouse	MouseListener	mouseClicked()
Window	WindowListener	windowOpened() windowClosed()
Component	ComponentListener	componentHidden() componentShown() componentMoved()

		componentResized()

File & Directory Chart

Method	Description
canRead()	check whether the file is readable or not
canWrite()	check whether the file is writable or not
createNewFile()	create an empty file
delete()	delete a file
exists()	check whether the file exists
getName()	return the name of the file
getAbsolutePath()	return the absolute pathname of the file
length()	return the size of the file in bytes
list()	return an array of the files in the directory
mkdir()	create a directory

Read() Methods Chart

read()	reads a byte of data from the input stream
readBoolean()	reads data of boolean type
readChar()	reads data of character type
readInt()	reads data of integer type
readObject()	reads the object from the input stream

Write() Methods Chart

write()	writes a byte of data to the output stream
writeBoolean()	writes data of boolean type
writeChar()	writes data of character type
writeInt()	writes data of integer type
writeObject()	writes object to the output stream

Paperback Books by Ray Yao

C# Cheat Sheet

C++ Cheat Sheet

JAVA Cheat Sheet

JavaScript Cheat Sheet

PHP MySQL Cheat Sheet

Python Cheat Sheet

Html Css Cheat Sheet

Linux Command Line