1. **Java Naming conventions**

|  |  |
| --- | --- |
| **Name** | **Convention** |
| Class name | Should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| Interface name | Should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| Method name | Should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| Variable name | Should start with lowercase letter e.g. firstName, orderNumber etc. |
| Package name | Should be in lowercase letter e.g. java, lang, sql, util etc. |
| Constants name | Should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |

1. **Arrays (Give examples of double array)**

Normally, array is a collection of similar type of elements that have contiguous memory location.

We can store only fixed set of elements in a java array.

Array in java is index based i.e. first element of the array is stored at 0 index.



**Advantages of Java Array**

Code Optimization: It makes the code optimized, we can retrieve the data easily.

Random access: We can get any data located at any index position.

**Disadvantages of Java Array**

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

**Types of Array in java**

1. Single Dimensional Array
2. Multidimensional Array

**Syntax to Declare an Array in java**

dataType varname[];

dataType[] varname;

dataType []varname;

**Syntax to Declare and Instantiate an Array in java**

dataType varname[] = new dataType[size of array];

dataType[] varname = new dataType[size of array];

dataType []varname = new dataType[size of array];

**Example of single dimensional java array**

Below is an example of java array where we are declaring, instantiating, initializing and traversing an array.

**class** Testarray {

**public** **static** **void** main(String args[]) {

**int** a[] = **new** **int**[5];// declaration and instantiation

a[0] = 10;// initialization

a[1] = 20;

a[2] = 70;

a[3] = 40;

a[4] = 50;

// printing array

**for** (**int** i = 0; i < a.length; i++)// length is the property of array

System.***out***.println(a[i]);

}

}

**Homework:**

Reference: <https://www.w3resource.com/java-exercises/array/index.php>

1. Write a Java program to sort a numeric array and a string array.
2. Write a Java program to sum values of an array
3. Write a Java program to print the following grid.

Expected Output:

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

- - - - - - - - - -

4) Write a Java program to calculate the average value of array elements.

5. Write a Java program to test if an array contains a specific value.

6.Write a Java program to find the index of an array element.

7. Write a Java program to remove a specific element from an array.

8. Write a Java program to copy an array by iterating the array.

9. Write a Java program to insert an element (specific position) into an array.

10. Write a Java program to find the maximum and minimum value of an array.

11. Write a Java program to reverse an array of integer values.

12. Write a Java program to find the duplicate values of an array of integer values.

13. Write a Java program to find the duplicate values of an array of string values.

14. Write a Java program to find the common elements between two arrays (string values).

15. Write a Java program to find the common elements between two arrays of integers.

16. Write a Java program to remove duplicate elements from an array.

17. Write a Java program to find the second largest element in an array.

18. Write a Java program to find the second smallest element in an array.

19. Write a Java program to add two matrices of the same size.

20. Write a Java program to convert an array to ArrayList.

21. Write a Java program to convert an ArrayList to an array.

22. Write a Java program to find all pairs of elements in an array whose sum is equal to a specified number.

23. Write a Java program to test the equality of two arrays.

24. Write a Java program to find a missing number in an array.

25. Write a Java program to find common elements from three sorted (in non-decreasing order) arrays.

26. Write a Java program to move all 0's to the end of an array. Maintain the relative order of the other (non-zero) array elements.

27. Write a Java program to find the number of even and odd integers in a given array of integers.

28. Write a Java program to get the difference between the largest and smallest values in an array of integers. The length of the array must be 1 and above.

29. Write a Java program to compute the average value of an array of integers except the largest and smallest values.

30. Write a Java program to check if an array of integers without 0 and -1.

31. Write a Java program to check if the sum of all the 10's in the array is exactly 30. Return false if the condition does not satisfy, otherwise true.

32. Write a Java program to check if an array of integers contains two specified elements 65 and 77.

33. Write a Java program to remove the duplicate elements of a given array and return the new length of the array.

Sample array: [20, 20, 30, 40, 50, 50, 50]

After removing the duplicate elements the program should return 4 as the new length of the array.

34. Write a Java program to find the length of the longest consecutive elements sequence from a given unsorted array of integers.

Sample array: [49, 1, 3, 200, 2, 4, 70, 5]

The longest consecutive elements sequence is [1, 2, 3, 4, 5], therefore the program will return its length 5.

35. Write a Java program to find the sum of the two elements of a given array which is equal to a given integer.

Sample array: [1,2,4,5,6]

Target value: 6.

36. Write a Java program to find all the unique triplets such that sum of all the three elements [x, y, z (x ≤ y ≤ z)] equal to a specified number.

Sample array: [1, -2, 0, 5, -1, -4]

Target value: 2.

37. Write a Java program to create an array of its anti-diagonals from a given square matrix.

Example:

Input :

1 2

3 4

Output:

[

[1],

[2, 3],

[4]

]

38. Write a Java program to get the majority element from an given array of integers containing duplicates.

Majority element: A majority element is an element that appears more than n/2 times where n is the size of the array.

39. Write a Java program to print all the LEADERS in the array.

Note: An element is leader if it is greater than all the elements to its right side.

40. Write a Java program to find the two elements from a given array of positive and negative numbers such that their sum is closest to zero.

41. Write a Java program to find smallest and second smallest elements of a given array.

42. Write a Java program to segregate all 0s on left side and all 1s on right side of a given array of 0s and 1s.

43. Write a Java program to find all combination of four elements of an given array whose sum is equal to a given value.

44. Write a Java program to count the number of possible triangles from an given unsorted array of positive integers.

45. Write a Java program to cyclically rotate a given array clockwise by one.

46. Write a Java program to check whether there is a pair with a specified sum of a given sorted and rotated array.

47. Write a Java program to find the rotation count in a given rotated sorted array of integers.

48. Write a Java program to arrange the elements of an given array of integers where all negative integers appear before all the positive integers.

49. Write a Java program to arrange the elements of an given array of integers where all positive integers appear before all the negative integers.

50. Write a Java program to sort an array of positive integers of an given array, in the sorted array the value of the first element should be maximum, second value should be minimum value, third should be second maximum, fourth second be second minimum and so on.

51. Write a Java program to separate 0s on left side and 1s on right side of an array of 0s and 1s in random order.

52. Write a Java program to separate even and odd numbers of an given array of integers. Put all even numbers first, and then odd numbers.

53. Write a Java program to replace every element with the next greatest element (from right side) in an given array of integers.

**Declaration, Instantiation and Initialization of Java Array**

int a[]={33,3,4,5};//declaration, instantiation and initialization

**class** Account {

**int** acc\_no;

String name;

**float** amount;

**void** insert(**int** a, String n, **float** amt) {

acc\_no = a;

name = n;

amount = amt;

}

**void** deposit(**float** amt) {

amount = amount + amt;

System.***out***.println(amt + " deposited");

}

**void** withdraw(**float** amt) {

**if** (amount < amt) {

System.***out***.println("Insufficient Balance");

} **else** {

amount = amount - amt;

System.***out***.println(amt + " withdrawn");

}

}

**void** checkBalance() {

System.***out***.println("Balance is: " + amount);

}

**void** display() {

System.***out***.println(acc\_no + " " + name + " " + amount);

}

}

**class** TestAccount {

**public** **static** **void** main(String[] args) {

Account a1 = **new** Account();

a1.insert(832345, "Ankit", 1000);

a1.display();

a1.checkBalance();

a1.deposit(40000);

a1.checkBalance();

a1.withdraw(15000);

a1.checkBalance();

}

}

1. **Java OOPs Concepts**

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

**Object**

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

**Class**

Collection of objects is called class. It is a logical entity.

**Inheritance**

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**

When one task is performed by different ways i.e. known as polymorphism e.g. to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

**Abstraction**

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

**Encapsulation**

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**Advantage of OOPs over Procedure-oriented programming language**

1) OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.

2) OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.

3) OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

**What is difference between object-oriented programming language and object-based programming language?**

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

1. **Object and Class in Java**

We will learn about java objects and classes. In object-oriented programming technique, we design a program using objects and classes.

Object is the physical as well as logical entity whereas class is the logical entity only.

**Object in Java**

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

**Class in Java**

It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

Syntax to declare a class:

**class** <class\_name>{

type variable1;

type variable2;

//…

type methodname1(type parameter1,type parameter2,…){

// body of method

}

type methodname2(type parameter1,type parameter2,…){

// body of method

}

//…

}

A class in Java can contain:

* fields
* methods
* constructors
* blocks
* nested class and interface

**Object in Java**

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance (result) of a class.

Object Definitions:

1. Object is a real world entity.
2. Object is a run time entity.
3. Object is an entity which has state and behavior.
4. Object is an instance of a class.

**Method in Java**

In java, a method is like function i.e. used to expose behavior of an object.

**Advantage of Method**

1. Code Reusability
2. Code Optimization

**‘new’ keyword in Java**

The new keyword is used to allocate memory at run time.

**Object and Class Example: main within class**

In this example, we have created a Student class that has two data members i.e. id and name. We are creating the object of the Student class by ‘new’ keyword and printing the objects value.

Here, we are creating main() method inside the class file: Student.java

**class** Student {

**int** id;// field or data member or instance variable

String name;

**public** **static** **void** main(String args[]) {

Student s1 = **new** Student(); // creating an object of Student

System.***out***.println(s1.id); // accessing member through reference variable

System.***out***.println(s1.name);

}

}

Object and Class Example: main outside class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

File: TestStudent1.java

**class** Student {

**int** id;

String name;

}

**class** TestStudent1 {

**public** **static** **void** main(String args[]) {

Student s1 = **new** Student();

System.***out***.println(s1.id);

System.***out***.println(s1.name);

}

}

**Function example:**

**class** Simple {

**public** **static** **void** main(String args[]) {

Simple s1 = **new** Simple();

System.***out***.println(s1.sumofnumbers(2, 3));

}

**int** sumofnumbers(**int** i, **int** j) {

**return** (i + j);

}

}

**There are 3 ways to initialize object in java.**

1. By reference variable
2. By method
3. By constructor

**1) Object and Class Example: Initialization through reference**

Initializing object simply means storing data into object. Let's see a simple example where we are going to initialize object through reference variable.

File: TestStudent2.java

**class** Student {

**int** id;

String name;

}

**class** TestStudent2 {

**public** **static** **void** main(String args[]) {

Student s1 = **new** Student();

s1.id = 101;

s1.name = "Sonoo";

System.***out***.println(s1.id + " " + s1.name);// printing members with a white space

}

}

Output:

101 Sonoo

**2) Object and Class Example: Initialization through method**

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

**class** Student {

**int** rollno;

String name;

**void** insertRecord(**int** r, String n) {

rollno = r;

name = n;

}

**void** displayInformation() {

System.***out***.println(rollno + " " + name);

}

}

**class** TestStudent4 {

**public** **static** **void** main(String args[]) {

Student s1 = **new** Student();

Student s2 = **new** Student();

s1.insertRecord(111, "Karan");

s2.insertRecord(222, "Aryan");

s1.displayInformation();

s2.displayInformation();

}

}

**3) Object and Class Example: Initialization through constructor**

We will learn about constructors in java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

**class** Employee {

**int** id;

String name;

**float** salary;

**void** insert(**int** i, String n, **float** s) {

id = i;

name = n;

salary = s;

}

**void** display() {

System.***out***.println(id + " " + name + " " + salary);

}

}

**public** **class** TestEmployee {

**public** **static** **void** main(String[] args) {

Employee e1 = **new** Employee();

Employee e2 = **new** Employee();

Employee e3 = **new** Employee();

e1.insert(101, "ajeet", 45000);

e2.insert(102, "Arjun", 25000);

e3.insert(103, "nakul", 55000);

e1.display();

e2.display();

e3.display();

}

}

**OutPut:**

101 ajeet 45000.0

102 Arjun 25000.0

103 nakul 55000.0

**Object and Class Example: Rectangle**

File: TestRectangle1.java

**class** Rectangle {

**int** length;

**int** width;

**void** insert(**int** l, **int** w) {

length = l;

width = w;

}

**void** calculateArea() {

System.***out***.println(length \* width);

}

}

**class** TestRectangle1 {

**public** **static** **void** main(String args[]) {

Rectangle r1 = **new** Rectangle();

Rectangle r2 = **new** Rectangle();

r1.insert(11, 5);

r2.insert(3, 15);

r1.calculateArea();

r2.calculateArea();

}

}

**Output:**

55

45

1. **What are the different ways to create an object in Java?**

* By new keyword
* By newInstance() method
* By clone() method
* By deserialization
* By factory method etc.

**Anonymous object**

Anonymous simply means nameless. An object which has no reference is known as anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, anonymous object is a good approach. For example:

new Calculation();//anonymous object

Calling method through reference:

Calculation c=new Calculation();

c.fact(5);

Calling method through anonymous object

new Calculation().fact(5);

Let's see the full example of anonymous object in java.

**class** Calculation {

**void** fact(**int** n) {

**int** fact = 1;

**for** (**int** i = 1; i <= n; i++) {

fact = fact \* i;

}

System.***out***.println("factorial is " + fact);

}

**public** **static** **void** main(String args[]) {

**new** Calculation().fact(5);// calling method with anonymous object

}

}

**Output:**

Factorial is 120

**Creating multiple objects by one type only**

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

int a=10, b=20;

Initialization of refernce variables:

Rectangle r1=new Rectangle(), r2=new Rectangle();//creating two objects

Let's see the example:

**class** Rectangle {

**int** length;

**int** width;

**void** insert(**int** l, **int** w) {

length = l;

width = w;

}

**void** calculateArea() {

System.***out***.println(length \* width);

}

}

**class** TestRectangle2 {

**public** **static** **void** main(String args[]) {

Rectangle r1 = **new** Rectangle(), r2 = **new** Rectangle();// creating two objects

r1.insert(11, 5);

r2.insert(3, 15);

r1.calculateArea();

r2.calculateArea();

}

}

**Output:**

55

45

Real World Example: Account

File: TestAccount.java

**class** Account {

**int** acc\_no;

String name;

**float** amount;

**void** insert(**int** a, String n, **float** amt) {

acc\_no = a;

name = n;

amount = amt;

}

**void** deposit(**float** amt) {

amount = amount + amt;

System.***out***.println(amt + " deposited");

}

**void** withdraw(**float** amt) {

**if** (amount < amt) {

System.***out***.println("Insufficient Balance");

} **else** {

amount = amount - amt;

System.***out***.println(amt + " withdrawn");

}

}

**void** checkBalance() {

System.***out***.println("Balance is: " + amount);

}

**void** display() {

System.***out***.println(acc\_no + " " + name + " " + amount);

}

}

**class** TestAccount {

**public** **static** **void** main(String[] args) {

Account a1 = **new** Account();

a1.insert(832345, "Ankit", 1000);

a1.display();

a1.checkBalance();

a1.deposit(40000);

a1.checkBalance();

a1.withdraw(15000);

a1.checkBalance();

}

}

Output:

832345 Ankit 1000.0

Balance is: 1000.0

40000.0 deposited

Balance is: 41000.0

15000.0 withdrawn

Balance is: 26000.0

1. **Constructor in Java**

In Java, constructor is a block of codes similar to method. It is called when an instance of object is created and memory is allocated for the object. It is a special type of method which is used to initialize the object.

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

There are two types of constructors in java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Syntax of default constructor:

<class\_name>(){

}

**Example of default constructor**

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

class Bike1{

Bike1(){System.out.println("Bike is created");}

public static void main(String args[]){

Bike1 b=new Bike1();

}

}

Default constructor is used to provide the default values to the object like 0, null etc. depending on the type.

**Java parameterized constructor**

A constructor which has a specific number of parameters is called parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor

In this example, we have created the constructor of Student class that has two parameters. We can have any number of parameters in the constructor.

**class** Student4 {

**int** id;

String name;

Student4(**int** i, String n) {

id = i;

name = n;

}

**void** display() {

System.***out***.println(id + " " + name);

}

**public** **static** **void** main(String args[]) {

Student4 s1 = **new** Student4(111, "Karan");

Student4 s2 = **new** Student4(222, "Aryan");

s1.display();

s2.display();

}

}

Output:

111 Karan

222 Aryan

**Constructor Overloading in Java**

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

**Example of Constructor Overloading**

**class** Student5 {

**int** id;

String name;

**int** age;

Student5(**int** i, String n) {

id = i;

name = n;

}

Student5(**int** i, String n, **int** a) {

id = i;

name = n;

age = a;

}

**void** display() {

System.***out***.println(id + " " + name + " " + age);

}

**public** **static** **void** main(String args[]) {

Student5 s1 = **new** Student5(111, "Karan");

Student5 s2 = **new** Student5(222, "Aryan", 25);

s1.display();

s2.display();

}

}

Difference between constructor and method in java

There are many differences between constructors and methods. They are given below.

|  |  |
| --- | --- |
| **Java Constructor** | **Java Method** |
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of an object. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. |
| Constructor name must be same as the class name. | Method name may or may not be same as class name. |

1. **Java static keyword**

The static keyword in java is used for memory management mainly. We can apply java static keyword on

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

The static keyword belongs to the class than instance of the class.

**Program of counter without static variable**

**class** Counter {

**int** count = 0;// will get memory when instance is created

Counter() {

count++;

System.***out***.println(count);

}

**public** **static** **void** main(String args[]) {

Counter c1 = **new** Counter();

Counter c2 = **new** Counter();

Counter c3 = **new** Counter();

}

}

Output:

1

1

1

**Program of counter by static variable**

**class** Counter2 {

**static** **int** *count* = 0;// will get memory only once and retain its value

Counter2() {

*count*++;

System.***out***.println(*count*);

}

**public** **static** **void** main(String args[]) {

Counter2 c1 = **new** Counter2();

Counter2 c2 = **new** Counter2();

Counter2 c3 = **new** Counter2();

}

}

Output:

1

2

3

**Java static method**

If you apply static keyword with any method, it is known as static method.

A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.

static method can access static data member and can change the value of it.

//Program of changing the common property of all objects(static field).

**class** Student9 {

**int** rollno;

String name;

**static** String *college* = "ITS";

**static** **void** change() {

*college* = "BBDIT";

}

Student9(**int** r, String n) {

rollno = r;

name = n;

}

**void** display() {

System.***out***.println(rollno + " " + name + " " + *college*);

}

**public** **static** **void** main(String args[]) {

Student9.*change*();

Student9 s1 = **new** Student9(111, "Karan");

Student9 s2 = **new** Student9(222, "Aryan");

Student9 s3 = **new** Student9(333, "Sonoo");

s1.display();

s2.display();

s3.display();

}

}

Output:

111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

**Restrictions for static method**

There are two main restrictions for the static method. They are:

1. The static method cannot use non static data member or call non-static method directly.
2. this and super cannot be used in static context.
3. **this keyword in java**

* this is a reference variable that refers to the current object.

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

**class** Student {

**int** rollno;

String name;

**float** fee;

Student(**int** rollno, String name, **float** fee) {

rollno = rollno; // parameters (formal arguments) and instance variables are same

name = name;

fee = fee;

}

**void** display() {

System.***out***.println(rollno + " " + name + " " + fee);

}

}

**class** TestThis1 {

**public** **static** **void** main(String args[]) {

Student s1 = **new** Student(111, "ankit", 5000f);

Student s2 = **new** Student(112, "sumit", 6000f);

s1.display();

s2.display();

}

}

**Output will be**

0 null 0.0

0 null 0.0

1. this can be used to invoke current class method (implicitly)
2. this() can be used to invoke current class constructor.
3. this can be passed as an argument in the method call.
4. this can be passed as argument in the constructor call.
5. this can be used to return the current class instance from the method.