**Day 16**

**Fundamentals**

**Topic 1 Constructor**

**Constructor** : Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally.

Constructors have the same name as class or structure.

Constructors don’t have a return type. (Not even void)

Constructors are only called once, at object creation.

There can be three types of constructors in Java.

**1. Non-Parameterized constructor** : A constructor which has no argument is known as non-parameterized constructor(or no-argument constructor). It is invoked at the time of creating an object. If we don’t create one then it is created by default by Java.

class Student {

String name;

int age;

Student() {

System.out.println("Constructor called");

}

}

**2. Parameterized constructor** : Constructor which has parameters is called a parameterized constructor. It is used to provide

different values to distinct objects.

class Student {

String name;

int age;

Student(String name, int age) {

this.name = name;

this.age = age;

}

}

3**. Copy Constructor :** A Copy constructor is an overloaded

constructor used to declare and initialize an object from another object. There is only a user defined copy constructor in Java(C++ has a default one too).

class Student {

String name;

int age;

Student(Student s2) {

this.name = s2.name;

this.age = s2.age;

}

}

**Note** : Unlike languages like C++, Java has no Destructor. Instead, Java has an efficient garbage collector that deallocates memory automatically.

**2nd Topic Compile time polymorphism** : Compile time polymorphism is also known as static polymorphism. It is achieved by method overloading or operator overloading.

Method overloading : If a class has multiple methods having same name but different parameters, it is known as Method overloading.

Polymorphism

Polymorphism is the ability to present the same interface for differing underlying forms (data types). With polymorphism, each of these classes will have different underlying data. Precisely, Poly means ‘many’ and morphism means ‘forms’.

Types of Polymorphism IMP

1. Compile Time Polymorphism (Static)

2. Runtime Polymorphism (Dynamic)

Let’s understand them one by one :

Compile Time Polymorphism : The polymorphism which is implemented at the compile time is known as compile-time polymorphism. Example - Method Overloading

**Method Overloading** : Method overloading is a technique which allows you to have more than one function with the same function name but with different functionality. Method overloading can be possible on the following basis:

1. The type of the parameters passed to the function.

2. The number of parameters passed to the function.

class Student {

String name;

int age;

public void displayInfo(String name) {

System.out.println(name);

}

public void displayInfo(int age) {

System.out.println(age);

}

public void displayInfo(String name, int age) {

System.out.println(name);

System.out.println(age);

}

}

**Questions**

**1) What is method overloading?**

When a class has more than one method with same name but different parameters, then we call those methods are overloaded. Overloaded methods will have same name but different number of arguments or different types of arguments.

**2) What is method signature? What are the things it consist of?**

Method signature is used by the compiler to differentiate the methods. Method signature consist of three things.

a) Method name

b) Number of arguments

c) Types of arguments

**3) Can we declare one overloaded method as static and another one as non-static?**

Yes. Overloaded methods can be either static or non static.

**4) How do compiler differentiate overloaded methods from duplicate methods?**

Compiler uses method signature to check whether the method is overloaded or duplicated. Duplicate methods will have same method signatures i.e same name, same number of arguments and same types of arguments. Overloaded methods will also have same name but differ in number of arguments or else types of arguments.

**5) Is it possible to have two methods in a class with same method signature but different return types?**

No, compiler will give duplicate method error. Compiler checks only method signature for duplication not the return types. If two methods have same method signature, straight away it gives compile time error.

**6) In “MyClass” , there is a method called “myMethod” with four different overloaded forms. All four different forms have different visibility ( private, protected, public and default). Is “myMethod” properly overloaded?**

Yes. Compiler checks only method signature for overloading of methods not the visibility of methods.

(Click [here](https://javaconceptoftheday.com/method-overloading-in-java/) to read more about method overloading)

**7) Can overloaded methods be synchronized?**

Yes. Overloaded methods can be synchronized.

**8) Can we overload main() method?**

Yes, we can overload main() method. A class can have any number of main() methods but execution starts from **public static void main(String[] args)** only.

**9) Can we declare overloaded methods as final?**

Yes, we can declare overloaded methods as final.

**10) In the below class, is constructor overloaded or is method overloaded?**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class A  {      public A()      {          //-----> (1)      }        void A()      {          //-----> (2)      }  } |

None of them. It is neither constructor overloaded nor method overloaded. First one is a constructor and second one is a method.

**11) Overloading is the best example of dynamic binding. True or false?**

False. Overloading is the best example for static binding. (Click [here](https://javaconceptoftheday.com/difference-between-static-binding-and-dynamic-binding/) to see what is static binding and what is dynamic binding)

**12) Can overloaded method be overrided?**

Yes, we can override a method which is overloaded in super class.

**Constructors**

1) Define Constructor?

Java constructor is a unique method that initializes the objects, which is called when an instance of the class is created. The memory for the object is allocated when we call the constructor.

Basically, a constructor is a block of code. When we create an object of the class using the new() keyword, at least one constructor is called, and it initializes the objects and allocates memory to them.

If we do not specify any constructor, it will call the default constructor of the class. However, it is not necessary to specify an explicit constructor because the Java compiler provides a default constructor for every Java class.

2) How many types of Constructors are in Java?

There are two types of constructors in Java:

1. **Default Constructor (Non-parameterized Constructor)**
2. **Parameterized Constructor**

The syntax for the default constructor is as follows:

1. <**class** name>() {}

**Example:**

1. Employee()
2. {
3. //some code
4. }

The syntax for the parameterized constructor is as follows:

1. <**class** name>(arg1, arg2) {}

**Example:**

1. Employee(**int** i, String n)
2. {
3. id = i;
4. name = n;
5. }

3) Do we have a copy constructor in Java?

Unlike C++, there is no explicit copy constructor in Java. However, we can achieve the functionality of a copy constructor in Java by copying the values from one object to another, just like the copy constructor.

The following are some methods to copy the values from one object to another:

* By constructor
* By assigning the values of one object to another
* By clone() method of Object class

4) Write a Java Program to Copy the values from one object to another Object.

Below Java program copies the values from one object to another object:

**ConstructorDemo.java:**

1. **class** ConstructorDemo{
2. **int** id;
3. String name;
4. //constructor to initialize integer and string
5. ConstructorDemo(**int** i,String n){
6. id = i;
7. name = n;
8. }
9. //constructor to initialize another object
10. ConstructorDemo(ConstructorDemo c){
11. id = c.id;
12. name =c.name;
13. }
14. **void** display(){System.out.println(id+" "+name);}
15. **public** **static** **void** main(String args[]){
16. ConstructorDemo c1 = **new** ConstructorDemo(100,"Joy");
17. ConstructorDemo c2 = **new** ConstructorDemo(c1);
18. c1.display();
19. c2.display();
20. }
21. }

**Output:**

100 Joy

100 Joy

In the above example, we have created two instances of the ConstructorDemo and passed the first object value into the second constructor. This way, we can use a copy constructor in Java.

5) Is there any method to call a sub-class constructor from a superclass constructor?

The subclass constructor has its own private data members, so Java does not provide any way to access the sub-class constructor from a super class constructor. However, we can call a superclass constructor from a sub-class constructor by using the super keyword.

6) Can we have a constructor in the Interface?

No, we cannot have constructors in the Java interface.

7) Explain Constructor Chaining?

Constructor Chaining is a way to call one constructor from another constructor with respect to the current object. It can be achieved in the following two ways:

**From base class:** We can use the super keyword to call a constructor from the base class.

**Within the same class:** We can call a constructor within the same class by using **this()** keyword.

Below is an example of constructor chaining:

1. **class** TestSuper
2. {
3. **public** TestSuper(**int** i)
4. {
5. System.out.println("Super Class Constructor");
6. }
7. }
9. **class** TestSub **extends** TestSuper
10. {
11. **public** TestSub()
12. {
13. **this**(10);      //Calling same class constructor
14. }
16. **public** TestSub(**int** i)
17. {
18. **super**(i);      //Calling super class constructor
19. }
20. }

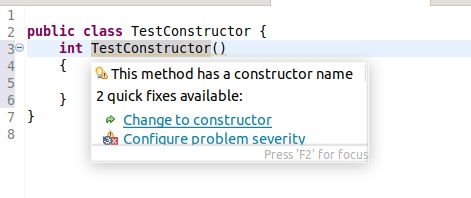
8) What happens if we provide a return type to a constructor?

If we provide a return type to a constructor, it will function as a general method. But, the compiler will display a warning message, "**This method has a Constructor name**".

**Consider the below example:**

1. **public** **class** TestConstructor {
2. **int** TestConstructor()
3. {
4. **return** 0;
5. }

The above program will be compiled gracefully, but it displays below warning message:



9) What is a private constructor?

Like methods, we can have the private constructors in Java. To make or create a constructor as private, use the **private** keyword while declaring it. It can only be accessed within that class.

The following are some usage scenarios when we need a private constructor:

* Internal Constructor chaining
* Singleton class design pattern

Below is an example of the private constructor:

**PrivateConstructor.java:**

1. **class** SingletonDemo {
2. **private** SingletonDemo() {
3. System. out.println("In a private constructor");
4. }
5. **public** **static** SingletonDemo getObject() {
6. // we can call this constructor
7. **if** (ref == **null**)
8. ref = **new** SingletonDemo();
9. **return** ref;
10. }
11. **private** **static** SingletonDemo ref;
12. }
13. **public** **class** PrivateConstructor {
14. **public** **static** **void** main(String args[]) {
15. SingletonDemo sObj = SingletonDemo.getObject();
16. }
17. }

**Output:**

In a private constructor

10) Why constructors in Java cannot be static?

The constructors cannot be static in Java. When we declare a method as static, it means the method belongs to the class and not to a specific object. But the constructor is always invoked to the reference of objects. So, there is no sense in making a constructor static.

11) Can we make a constructor final?

No, we cannot make a constructor final. If we made a constructor final, it would throw a compile-time error "**modifier final not allowed**".

12) Can we make a constructor abstract?

a body, which really makes no sense. It is automatically called at the time of object creation. So, it cannot be a block without a body.

13) what will happen when a constructor is declared as protected?

Generally, when we declare a method as protected, other classes can access that method in a different package by using inheritance only. But, when we declare a constructor protected, it behaves slightly differently than a method. The protected constructor can only be accessed by using a super keyword according to Java language standards.

14) Why constructor name is similar to the class name?

When we create an object of a class using a new keyword, it should have information about that particular class. That is why the constructor's name must be similar to the class name.

15) Why return type is not allowed for the constructor?

The return type is not allowed in the constructor because if we provide a return type in the constructor, it will act as the normal method. So, to differentiate between constructor and method block, the return type is not allowed in constructors.