**The Use Of Constructors In Abstract classes**

* By default, JVM is dealing with the constructor chaining because if you are creating an object by a new keyword then JVM invokes the constructor of that class and that constructor invokes the superclass constructor
* To use an abstract class in Java, we use the concrete class that extends the abstract class and provides the implementation to the abstract methods.
* When we use the constructor of the child class, the constructor of the parent class invoked by child class constructor either implicitly or explicitly.
* This is one of the reasons abstract class can have constructors in Java. Suppose if JVM doesn’t provide the default constructor to an abstract class then JVM not able to support constructor chaining.

**Difference Between Abstract And Interface**

|  |  |
| --- | --- |
| **Abstract** | **Interface** |
| An abstract class Is written when there are some common features shared by all  Objects    An abstract class contain some abstract methods and also concreate Method  Abstract class can also contain instance variables also  All the abstract methods of a abstract class must be implemented in its sub classes  It is created by using abstract keyword | An interface is written when all the features are implemented differently in different object  An interface contains only abstract method  Interface does not contain any instance only constants  All the abstract methods in interface should be implemented in implementation classes  It is created by using interface keyword |

**3) Up Casting && Down casting in java**

* Converting an object from one type to another is a very important aspect of Java which is popularly known as Typecasting
* Upcasting (Generalization or Widening) is casting to a parent type in simple words casting individual type to one common type is called upcasting Upcasting can be done implicitly.
* while downcasting (specialization or narrowing) is casting to a child type or casting common type to individual type**.**
* Downcasting is used more frequently than upcasting. Use downcasting when we want to access specific behaviours of a subtype**.**

**4)Anonymous block & how to use it**

* An anonymous block in java is a special member of a class. It does not have names and represents statements that are common to all the constructors of the class.

An anonymous block is used when you want to execute some common statements before all the constructors that are available in a class**.**

class Demo{

public Demo(){

System.out.println("default constructor");

}

public Demo(int i){

System.out.println("parameterized constructor");

}

{

System.out.print("Object is created by the ");

}

public static void main(String arr[]){

Demo b1 = new Demo();

Demo b2 = new Demo(1);

}

}

Output:

Object is created by the default constructor

Object is created by the parameterized constructor

* As you can see the statement of the anonymous block is executed before the constructor.
* This happened because, at the time of compilation, statements of the anonymous block are moved to the beginning of each constructor by the compiler.
* **Instanceof operator**
* This operator is used to test if an object belongs to a class or not. Note that the word instance means object.
* This operator can also be used to check if an object belongs to an interface or not.

**Syntax**

boolean variable = object instanceof class;

boolean variable = object instanceof interface;

**e.g.:** Boolean b=emp instance of Employee;

Here, we are testing if emp is an object of Employee class or not.

If emp is an object of Employee class, then true will be returned into b, otherwise b will contain false

**Constructor can be abstract**

One of the important property of java constructor is that it can not be abstract. If we are declaring a constructor as abstract as we have to implement it in a child class, but we know a constructor is called implicitly when the new keyword is used so it can’t lack a body and also it can not be called as a normal method**.**

**Constructor can be final**

One of the important property of java constructor is that it can not be final. As we know, constructors are not inherited in java. Therefore, constructors are not subject to hiding or overriding**.**

**Difference between compile time binding run time binding**

|  |  |
| --- | --- |
| **compile time binding** | **run time binding** |
| Compiler is responsible for the compile time address binding.  Compile time address binding is done before loading the program into memory.  It is static address binding  Program execution is faster | Execution time address binding is done by processor.  Execution time address binding is done at the time of program execution.  It is dynamic address binding.  Program execution is slower |