Is it mandatory to define a constructor in a class?

No, it is optional. If we do not define a constructor, compiler will automatically consider it as a default constructor.

Define a constructor.

• Constructor is a special method provided in OOP language for creating and initializing an object.

• In java, the role of a constructor is only to initialize an object and new key role is creating an object.

Can we define a method with same name as that of the class?

Yes, it is allowed to define a method with same name as that of a class. No compile time error or runtime error will occur, but this practice is not recommended as per coding standards.

How can a compiler and a JVM differentiate between constructor and method definitions if both have same name as the class has?

A compiler and a JVM can differentiate between the definitions of the the constructor and method with the help of return type. If there is a return type then it is considered as a method else it is a constructor.

What are the rules for defining a constructor?

• Constructor name should be same as class name.

• It should not contain return type.

• It should not contain non Access Modifiers such as final ,static, abstract, synchronized etc.

• A logic return statement with a value is not allowed.

• It can have all four accessibility modifiers i.e. private , public, protected and default.

• It can have parameters.

• It can have a 'throws' clause which means we can throw an exception from a constructor.

• It can have a logic and as a part of logic it can have all java legal statements except the return statement with value.

• We cannot place 'return' in constructor.

If we place a return type in a constructor prototype will it throw an Error?

No, because compiler and JVM considers it as a method.

Why a return type is not allowed for constructor?

As there is a possibility of a method having the same name as class name, return type is not allowed in a constructor to differentiate constructor block from method block.

How does a compiler and JVM differentiate constructor and method invocations if both have same name as class name?

A compiler and a JVM differentiates constructor and method invocations by using 'new' keyword. If 'new' keyword is used in calling then a constructor is executed else method is executed.

Can we declare constructor as 'private'?

• Yes we can declare a constructor as private.

• All four access modifiers are allowed for a constructor.

• We can declare a constructor as private if we do not want to allow a user to create an object from outside that class.

- **Why a compiler given constructor is called as default constructor?**

• A constructor defined/given by a class is called as a default constructor because it obtains all its default properties from its class.

• They are as mentioned below:

1.Its accessibility modifier is same as its class accessibility modifier.
2.Its name is same as class name.
3.It does not have parameters and logic.

- **Why a constructor name is same as class name?**

• Every class object is created using the same 'new' keyword, so it must have information about the class to which it must create object.

• For this reason constructor name should be same as class name.

- **What is default accessibility modifier of default constructor?**

The default accessibility modifier of default constructor is assigned from its class.

- **In which situation it is mandatory for the developer to provide constructor explicitly?**

A developer needs to provide constructor explicitly when one needs to execute some logic at the time of object creation. This logic might be object initializeed logic or some other useful logic.

- **When does a compiler provide default constructor?**

Only if there is no explicit constructor defined by developer then in such situation a compiler will provide a default constructor.

- **Why a constructor name is same as class name?**

• Every class object is created using the same 'new' keyword, so it must have information about the class to which it must create object.

• For this reason constructor name should be same as class name.

- **What is default accessibility modifier of default constructor?**

The default accessibility modifier of default constructor is assigned from its class.

- **In which situation it is mandatory for the developer to provide constructor explicitly?**

A developer needs to provide constructor explicitly when one needs to execute some logic at the time of object creation. This logic might be object initializeed logic or some other useful logic.

- **When does a compiler provide default constructor?**

Only if there is no explicit constructor defined by developer then in such situation a compiler will provide a default constructor.

- **If class has an explicit constructor then will it have a default constructor?**

No. Compiler places default constructor only if there is no explicit constructor.

- **What is constructor chaining?**

Constructor Chaining is a technique of calling another constructor from one constructor. 'this()' is used to call same class constructor where as 'super()' is used to call super class constructor.
class SuperClass

{

```
public SuperClass(int i)

{

System.out.println("Super Class Constructor");

}

}



class SubClass extends SuperClass

{

public SubClass()

{

this(10); //Calling same class constructor

}



public SubClass(int i)
```

```
{

super(i); //Calling super class constructor

}

}
```

- **What will happen if one keeps a return type for a constructor?**

If one keeps a return type for a constructor it will be treated as a normal method, but compiler gives a warning saying that method has a constructor name.

```
class MyClass

{

int MyClass()

{

return 0; //No Compile time error but just a warning

}

}
```

- **Can we call sub class constructor from super class constructor?**

No. There is no way to call sub class constructor from a super class constructor in Java.

- **What is No-arg constructor?**

Constructor without any arguments is called no-arg constructor. Default constructor in java is always known as no-arg constructor.

```
class MyClass

{
```

```
public MyClass()

{

//No-arg constructor

}

}
```

- **What is the use of private constructor?**

Private constructors are used to restrict the instantiation of a class. When a class needs to prevent other classes from creating it's objects then private constructors are suitable for that. Objects to the class which has only private constructors can be created within the class. A very good use of private constructor is in singleton pattern. This ensures that only one instance of a class exists at any point of time. Here is an example of singleton pattern using private constructor.

```
class MyClass

{

private static MyClass object = null;

private MyClass()
{

//private constructor

}



public MyClass getObject()
```

```
{

if(object == null)

{

object = new MyClass(); //Creating object using private constructor

}

return object;

}

}
```

- **What is constructor chaining and how can it be achieved in Java ?h3>**
- **Can we use this() and super() in a method?**

No, we can't use this() and super() in a method.

```
class SuperClass

{

public SuperClass()

{

System.out.println("Super Class Constructor");

}
```

```
}

class SubClass extends SuperClass
{

public SubClass()

{

System.out.println("Sub Class Constructor");

}

void method()

{

this(); //Compile time error


super(); //Compile time error

}


}
```

- **What is a Default Constructor ?**

The 'no argument'constructor provided by Java Compiler when no constructor is specified is known as a default constructor.

- **Can we overload constructors?**

Yes, we can overload constructors.

- **Does a constructor create the object?**

'New' operator in Java creates the objects. Constructor comes in the later stage in object creation. Constructor's job is to initialize the members after the object has reserved memory for itself.

- **What are the common uses of "this" keyword in java ?**

"this" keyword is a reference to the current object and can be used for the following -

1. Passing itself to another method.

2. Referring to the instance variable when local variable has the same name.

3. Calling another constructor in constructor chaining