Constructors in Java

Constructor is a special method which name should be same as class name and does not return any thing which is used to create the object.

Constructor has same name as the class and looks like this in a java code.

```
public class MyClass{

    //This is the constructor

    MyClass(){

    }     }
```

How does a constructor work

To understand the working of constructor, lets take an example. lets say we have a class MyClass.
When we create the object of MyClass like this:

```
MyClass obj = new MyClass()
```

The **new keyword** here creates the object of class MyClass and invokes the constructor to initialize this newly created object.

A simple constructor program in java

Here we have created an object obj of class Hello and then we displayed the instance variable name of the object. As you can see that the output is Bcastudents.com which is what we have passed to the name during initialization in constructor. This shows that when we created the object obj the constructor got invoked. In this example we have used **this keyword**, which refers to the current object, object obj in this example.

```
public class Hello {
    String name;
    //Constructor
    Hello(){
        this.name = "SCAstudents.com";
    }
    public static void main(String[] args) {
        Hello obj = new Hello();
        System.out.println(obj.name);
    }}
```

Output:

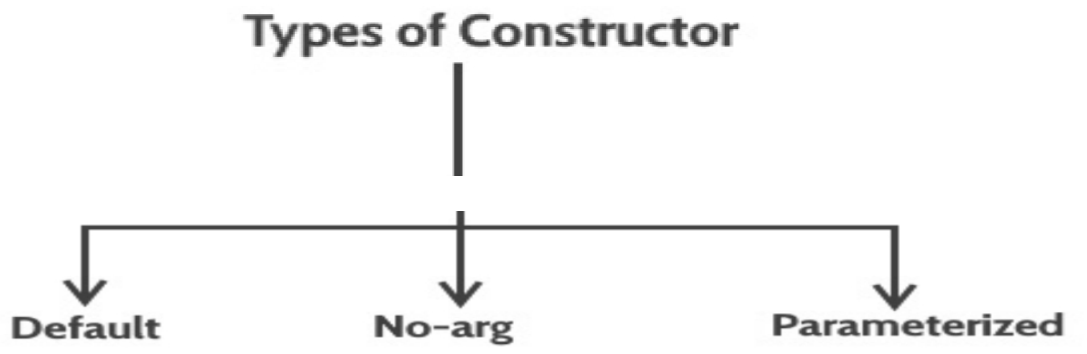
```
SCAstudents.com
```

```
public class Ex{
// constructor
Ex(){
System.out.println(“SCA Students”);
}


public static void main(String args[]){
Ex e = new Ex();
} //new keyword creates the object of Ex and invokes the
} constructor to initialize the created object
```

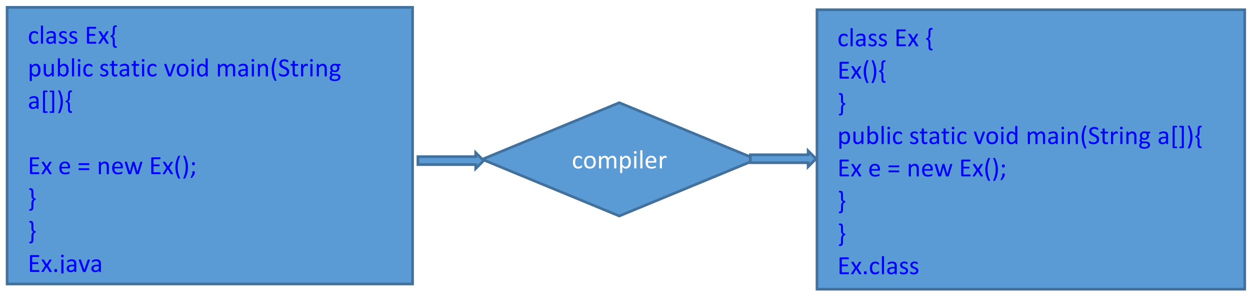
Types of Constructors

There are three types of constructors: Default, No-arg constructor and Parameterized.



Default constructor

If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf. This constructor is known as default constructor. You would not find it in your source code(the java file) as it would be inserted into the code during compilation and exists in .class file. This process is shown in the diagram below:



```
class Ex{
public static void main(String
a[]){

Ex e = new Ex();
}
}
Ex.java
```

```
class Ex {
Ex(){
}
public static void main(String a[]){
Ex e = new Ex();
}
}
Ex.class
```

* If you implement any constructor then you no longer receive a default constructor from Java compiler.

no-arg constructor:

Constructor with no arguments is known as **no-arg constructor**. The signature is same as default constructor, however body can have any code unlike default constructor where the body of the constructor is empty.

Although you may see some people claim that that default and no-arg constructor is same but in fact they are not, even if you write `public Demo() { }` in your class `Demo` it cannot be called default constructor since you have written the code of it.

Example: no-arg constructor

```
class Demo{
        public Demo()
        {
              System.out.println("This is a no argument constructor");
        }
        public static void main(String args[]) {
              new Demo();
        }}
```

Output:
This is a no argument constructor

Parameterized constructor- Constructor with arguments(or you can say parameters) is known as Parameterized constructor.Example: parameterized constructor

In this example we have a parameterized constructor with two parameters id and name. While creating the objects obj1 and obj2 I have passed two arguments so that this constructor gets invoked after creation of obj1 and obj2.

```
public class Employee {
     int empId;
     String empName;
     Employee(int id, String name){
            this.empId = id;
            this.empName = name;
       System.out.println("Id: "+id+" Name: "+name);
     }
     void info(){
             System.out.println("Id: "+empId+" Name: "+empName);
     }
     public static void main(String args[]){
             Employee obj1 = new Employee(10245,"Abhijeet");
             Employee obj2 = new Employee(92232,"Aditi");
             obj1.info();
             obj2.info();
     }    }
```

Output:

```
Id: 10245 Name: Abhijeet Id: 92232 Name: Aditi
```

Example2: parameterized constructor

In this example, we have two constructors, a default constructor and a parameterized constructor. When we do not pass any parameter while creating the object using new keyword then default constructor is invoked, however when you pass a parameter then parameterized constructor that matches with the passed parameters list gets invoked.

```
class Example2{
        private int var;
        public Example2()
        {
          this.var = 10;
        }
        public Example2(int num)
        {
            this.var = num;
        }
        public int getValue()
        {
                    return var;
        }
        public static void main(String args[])
        {
                     Example2 obj1 = new Example2();
                    Example2 obj2 = new Example2(100);
                    System.out.println("var is: "+obj1.getValue());// 10
                    System.out.println("var is: "+obj2.getValue()); // 100
        }}
```

Output:

```
var is: 10var is: 100
```

What if you implement only parameterized constructor in class

```
class Example3{
          private int var;
          public Example3(int num)
          {
                    var=num;
          }
          public int getValue()
          {
                    return var;
          }
          public static void main(String args[])
          {
                    Example3 myobj = new Example3();
                    System.out.println("value of var is: "+myobj.getValue());
          }}
```

Output: It will throw a compilation error. The reason is, the statement Example3 myobj = new Example3() is invoking a default constructor which we don't have in our program. when you don't implement any constructor in your class, compiler inserts the default constructor into your code, however when you implement any constructor (in above example I have implemented parameterized constructor with int parameter), then you don't receive the default constructor by compiler into your code.

If we remove the parameterized constructor from the above code then the program would run fine, because then compiler would insert the default constructor into your code.

Constructor Chaining

When A constructor calls another constructor of same class then this is called constructor chaining.

```
public class Ex{
Ex(){
this("SCAStudents");
}
Ex(String s){
this(s, 20);
}
Ex(String s, int age){
this.name = s;
this.age = age;
}
public static void main(String args[]){
Ex e = new Ex();
}
}
```

*this keyword will be introduced in details later.

Super()

Whenever a child class constructor gets invoked it implicitly invokes the constructor of parent class. You can also say that the compiler inserts a super(); statement at the beginning of child class constructor.

```
class MyParentClass {
   MyParentClass(){
        System.out.println("MyParentClass Constructor");
   }}class MyChildClass extends MyParentClass{
   MyChildClass() {
        System.out.println("MyChildClass Constructor");
   }
   public static void main(String args[]) {
        new MyChildClass();
   }}
```

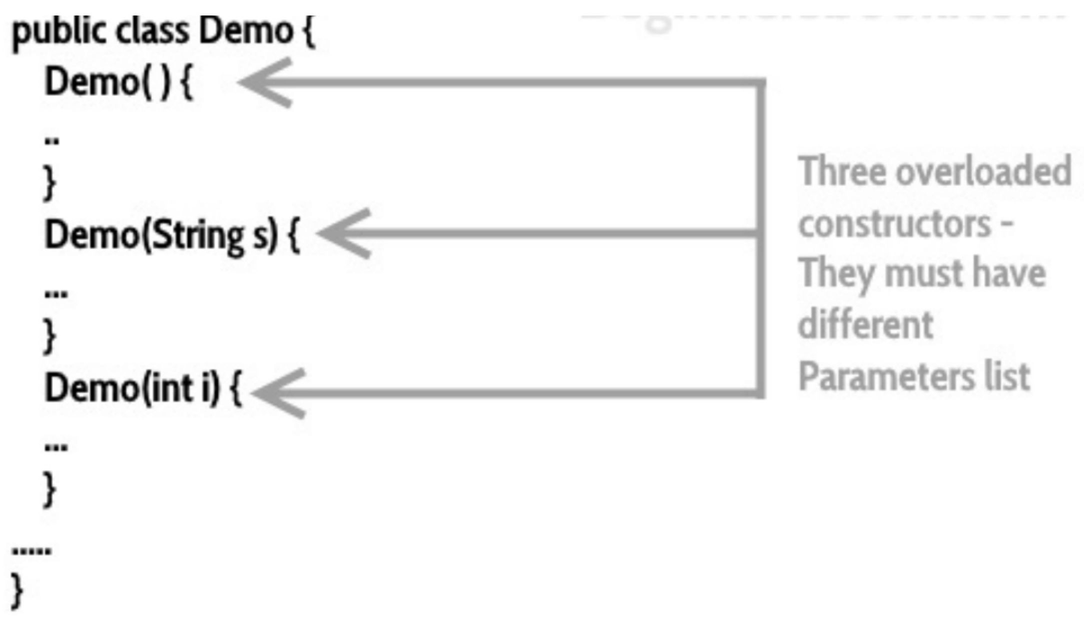
Output:

```
MyParentClass ConstructorMyChildClass Constructor
```

super keyword will be introduced later.

Constructor Overloading

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.



```
public class Demo {
   Demo( ) {
   ..
   }
   Demo(String s) {
   ...
   }
   Demo(int i) {
   ...
   }
.....
}
```

Java Copy Constructor

A copy constructor is used for copying the values of one object to another object.

```
class JavaExample{
   String web;
   JavaExample(String w){
         web = w;
   }
   /* This is the Copy Constructor, it copies the values of one object
    * to the another object (the object that invokes this constructor)*/
   JavaExample(JavaExample je){
         web = je.web;
   }
   void disp(){
         System.out.println("Website: "+web);
   }

   public static void main(String args[]){
         JavaExample obj1 = new JavaExample("SCAStudents");

/* Passing the object as an argument to the constructor This will invoke the
copy constructor */
         JavaExample obj2 = new JavaExample(obj1);
         obj1.disp();
         obj2.disp();
   }  }
```

Output:

```
Website: SCAStudents Website: SCAStudents
```