Constructor

Constructor in Java is a block of code like a method that's called when an instance of an object is created. The following table shows the differences between constructor and method in Java.

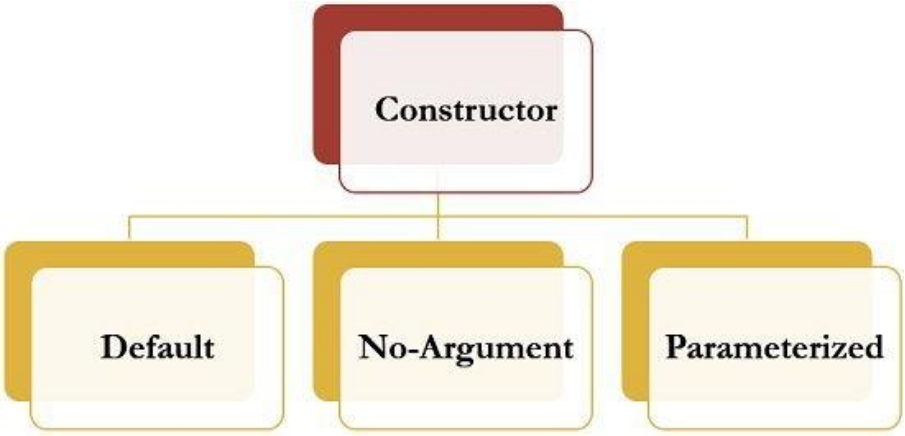
Java Constructor	Java Method
A constructor is used only to initialize the state of an object, this means that it allows to provide initial values for class fields when the object is created.	**A method is used to expose the behavior of an object and may be for initializing the state of an object**
A constructor must not have a return type.	**A method must have a return type.**
The constructor is invoked implicitly. Which is called automatically when a new instance of an object is created.	**The method is invoked explicitly (not automatically by passing a message)**
The Java compiler provides a default constructor if you don't have any constructor in a class.	**The method is not provided by the compiler in any case.**
The constructor name must be same as the class name.	**The method name may or may not be same as class name.**
The constructors are not considered members of a class.	**The methods are members of a class.**

The basic format for coding a constructor:

```
public ClassName (parameter-list)
{
   statements...
}
```

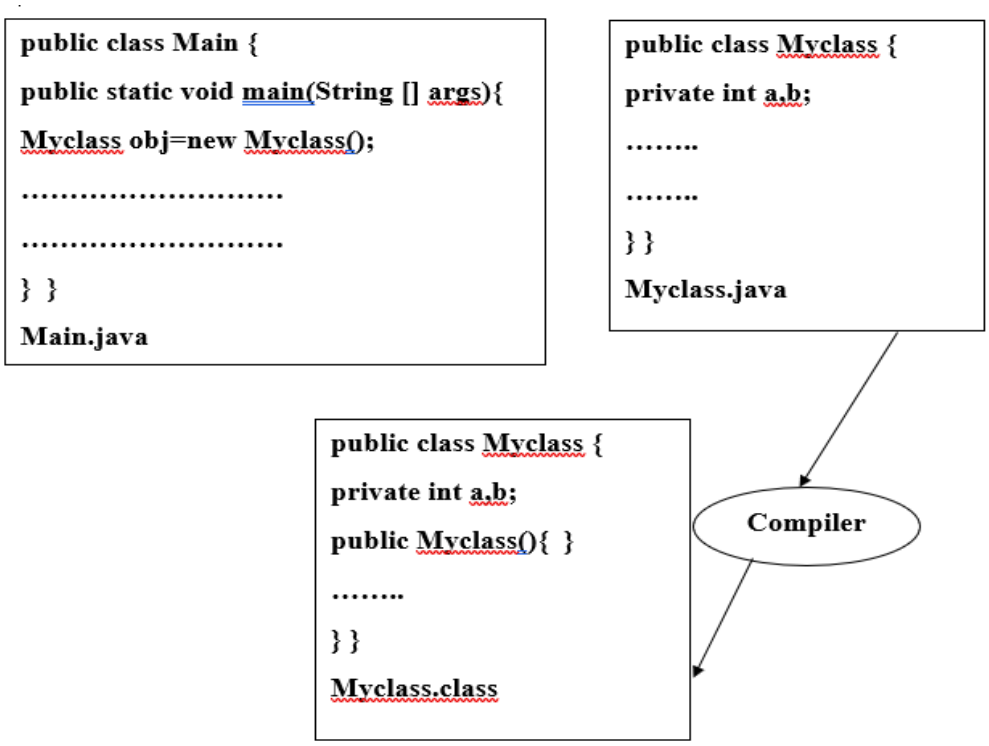
Constructor Types

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



1- Default constructor

In computer programming languages, the term **default constructor** can refer to a **constructor** that is automatically generated by the compiler in the absence of any programmer-defined **constructors** (e.g. in **Java**) and is usually a **nullary constructor**. All Java classes have at least one constructor even if we don't explicitly define one. The compiler automatically provides a public no-argument constructor for any class without constructors.



Example 1:

> **Rule: If there is no constructor in a class, compiler automatically creates a default constructor.**


```
public class Bike{
 public void print(){
System.ot.println("Bike is created");
}}
public class Main{
public static void main(String args[]){
//calling a default constructor
Bike b=new Bike();
} }
```

The Output is:
Bike is created

Calling a Constructor: You call a constructor when you create a new instance of the class containing Bike b=new Bike();

2- 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.

Example 2:

```
public class Student{
private int id;
private boolean state;
public Student(){ id=10; state=true;}
 void display(){System.out.println(id+"   "+state);}  }
```

```
public class Main{
public static void main(String args[]){
Student s1=new Student();   Student s2=new Student();
s1.display();   s2.display();   }   }
```

The output is :

10 true

10 true

Calling a Constructor: You call a constructor when you create a new instance of the class containing

```
Student s1=new Student();
Student s2=new Student();
```

H.W. what will be the output when there is no constructor in class Student?

Hint : In the above class ,when you are not creating any constructor , the compiler provides you a default constructor.

3- Parameterized constructor

Constructor with arguments (or you can say parameters) is known as Parameterized constructor. The parameterized constructor is used to provide different values to the fields of objects.

Example 3:

```
public class Student{
  private  int id;
   private boolean state;
   public  Student(int i,boolean b){
  id = i;     state = b;      }
   public void display(){System.out.println(id+"  "+state); }  }
 public class Main{
   public static void main(String args[]){
   Student s1 = new Student(111,true);
   Student s2 = new Student(222,false);
   s1.display();      s2.display();     }  }
```

The Output is:

111 true

222 false

Calling a Constructor: You call a constructor when you create a new instance of the class containing

```
Student s1 = new Student(111,true);
Student s2 = new Student(222,false);
```

Constructor Overloading

As like of method **overloading**, **Constructors** also can be **overloaded**. The same **constructor** declared with different parameters in the same class is known as **constructor overloading**. Compiler differentiates which **constructor** is to be called depending upon the number of parameters and their sequence of data types

Example 4:

```
public class Student{
 private int id;
 private   boolean state;
 private int age;
   public Student(int i,boolean  b){
    id = i;      state = b;      }
   public  Student(int i,boolean b,int a){
    id = i;       state = b;       age=a;      }
    public void display(){System.out.println(id+"  "+ state +"  "+age);}
}

public class Main{
    public static void main(String args[]){
    Student s1 = new Student(111,true);
    Student s2 = new Student(222,false,25);
    s1.display();      s2.display();
   }
}
```

The Output is:

111 true 0

222 false 25

Example 5 (Trace):

What is the function of the following program and what will be the output?

```
class Counter {
private int number;
public void add() {  number = number+1; }
public void add(int n){ number = number+n; }
public void initialize() { number = 0;   }
public void initialize(int k) {number = k;  }
public int getNumber() { return number; }
public Counter() { number = 0;  } }
```

```
public class Main{
public static void main(String[] args){

Counter c1=new Counter();
c1.add(0; c1.add(7);
System.out.println(c1.getNumber());

c1.initialize();  c1.add(5); c1.add(10);
System.out.println(c1.getNumber());

Counter c2=new Counter();
c2.initialize(10); c2.add();c2.add(20);
System.out.println(c2.getNumber());
}}
```

Example 6 (write):

Define a class called PairOfDice, this class has two integer variable members d1 and d2 which represent two dices. It has two constructors the first one is used for setting the initial values for dices while the second one is used for calling the method roll. The roll method is throwing the dices, so it needs to use the random method.

Use this class for representing a two players game each player has two dices. Players will throw the dices and stop when the summation of dices values for two players are equal then the program will print the number of these throws.

```
public class PairOfDice {

public int d1;
public int d2;
public PairOfDice() {roll();   }
public PairOfDice(int v1, int v2) {
d1 = v1; d2 = v2;   }

public void roll() {
d1 = (int)(Math.random()*6) + 1;
d2 = (int)(Math.random()*6) + 1; }
 }
```

```
public class Game{
public static void main(String[] args) {
PairOfDice player1 = new PairOfDice();
PairOfDice player2 = new PairOfDice();

int countRolls; / / Counts how many times the two p a i r s of
int total1, total2; countRolls = 0;

do {

player1.roll();
total1 = player1.d1 + player1.d2;

player2.roll();
total2 = player2.d1 + player2.d2;

countRolls++;

} while (total1 != total2);

System.out.println(" I took " + countRolls + " rolls until the totals were the same. ");
} }
```

H.W.

Complete the following program to create three counters each of which uses a different constructor.

```
class Counter {

private int number, reused;

public void add() { number = number+1; }

public void initialize() { number = 0;  reused = reused+1; }

public int getNumber() { return number; }

public int getReused() { return reused; }

public Counter() { number = 0; reused = 0; }

Counter(int x) { number = x; reused = 0; }

Counter(int x, int y) { number = x; reused = y; }

Counter(float z) { number = (int) z; reused = 0; }
}
```