Object Oriented Programming:

Procedural (or Modular) programming allowed sharing of data by all modules. At times that causes problem. OOP separates data from code at object level.

## **Definitions:**

### **Class**

* Collection of attributes and methods
* A type e.g. Lamp, Vehicle

### **Object**

* An instance of a class e.g. kitchenLamp, bedroomLamp

### **Method (Operation) – functions**

* Control attribute state

### **Attribute**

* State, variable

## **Class Anatomy:**

public class Vehicle {

private int speed;

private int weight;

public int getSpeed() { //getter method

return speed;

}

public int getWeight() { //another getter method

return weight;

}

public void setSpeed (int speed) { //setter method

this.speed = speed;

}

public void setWeight(int weight) { //another setter method

this.weight = weight

}

}

## **Constructor:**

public class Vehicle {

public Vehicle() { //No argument constructor

this.weight = 200;

}

public Vehicle (Int speed) { //One argument constructor

this();

this.speed = speed;

}

## **Subclass of Vehicle:**

public class Truck extends Vehicle {

public Truck() {

}

private boolean isFourWheel() {

return fwd;

}

public void setFourWheel(Boolean fwd) {

this.fwd = fwd;

}

}

## **Creating Objects:**

public class Example {

public static void main(Strings[] args) {

Vehicle geo = new Vehicle();

Truck ram = new Truck();

Truck dakota = new Truck();

geo.setSpeed(75);

ram.setSpeed(100);

ram.setFourWheel(true);

dakota.setSpeed(110);

……………..

……………..

## **Another example:**

public class Student {

String name;

//Constructor

Student(){

this.name = "Jane Doe";

}

public static void main(String[] args) {

Student std = new Student(); //New keyword creates a student object

//and initializes it.

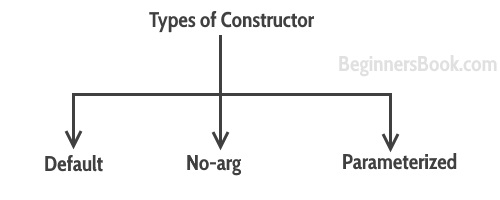
System.out.println(std.name);

}

}

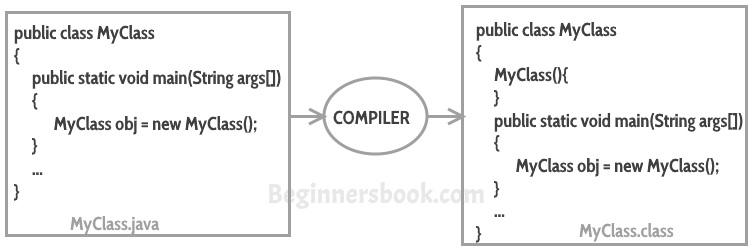
## Types of Constructors

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



### Default constructor

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:



#### Alternate version:

public class MyClass

{

public static void main (Strings args[])

{

MyClass obj = new MyClass();

}

…

}

MyClass.java

public class MyClass

{

MyClass(){

{

public static void main (String args [])

}

MyClass obj = new MyClass();

}

…

}

MyClass.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;

//parameterized constructor with two parameters

Employee(int id, String name){

this.empId = id;

this.empName = name;

}

void info(){

System.out.println("Id: "+empId+" Name: "+empName);

}

public static void main(String args[]){

Employee obj1 = new Employee(10245,"Jane");

Employee obj2 = new Employee(92232,"John");

obj1.info();

obj2.info();

}

}

**Output:**

Id: 10245 Name: Jane

Id: 92232 Name: John

#### **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;

//no argument constructor

public Example2()

{

this.var = 10;

}

//parameterized constructor

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());

System.out.println("var is: "+obj2.getValue());

}

}

**Output:**

var is: 10

var 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());

}

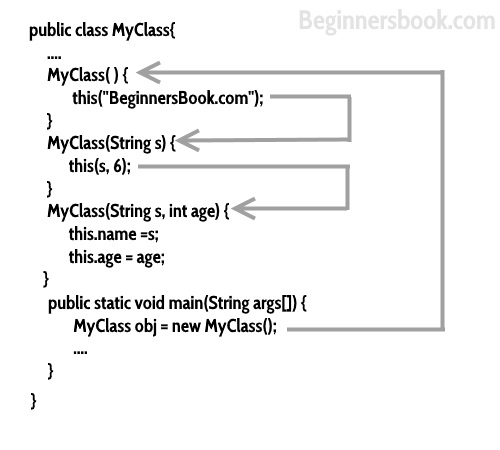
}

**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.



Alternate version:

public class MyClass{

….

My Class() {

this (“BeginnersBook.com”);

}

MyClass(String) {

this(s, 6);

}

MyClass(String s, int age) {

this.name =s;

this.age =age;

}

public static void main (String args []) {

MyClass obj = new MyClass ();

….

}

}

## 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();

}

}

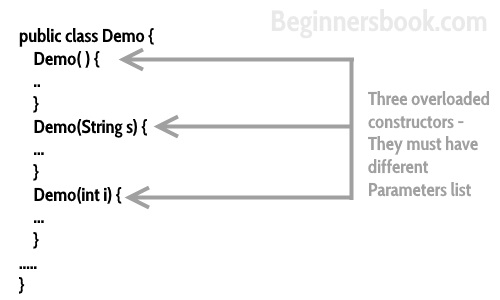
**Output:**

MyParentClass Constructor

MyChildClass Constructor

## 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.



Alternate version:

public class Demo {

Demo() {

..

}

Demo(String s){

…

}

Demo (int i) {

…

}

…..

}

Three overladed constructs – They must have different Parameters list

## 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 a Copy Constructor, it copies the values of one object

\* to 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("BeginnersBook");

/\* 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: BeginnersBook

Website: BeginnersBook

## Quick Recap

1. Every class has a constructor whether it’s a normal class or an abstract class.
2. Constructors are not methods and they don’t have any return type.
3. Constructor name should match with class name.
4. Constructor can use any access specifier, they can be declared as private also. Private constructors are possible in java but there scope is within the class only.
5. **Like constructors method can also have name same as class name, but still they have return type, through which we can identify them that they are methods not constructors.**
6. If you don’t implement any constructor within the class, compiler will do it for.
7. **this() and super() should be the first statement in the constructor code.** If you don’t mention them, compiler does it for you accordingly.
8. Constructor overloading is possible but overriding is not possible. Which means we can have overloaded constructor in our class but we can’t override a constructor.
9. Constructors cannot be inherited.
10. If Super class has a no-argument constructor then compiler would not insert a default constructor in child class as it does in normal scenario.
11. Interfaces do not have a constructor.
12. Abstract class can have constructor and it gets invoked when a class, which implements interface, is instantiated. (i.e. object creation of concrete class).
13. A constructor can also invoke another constructor of the same class – By using this(). If you want to invoke a parameterized constructor then do it like this: **this(parameter list)**.

**More on Constructor:**

## Difference between Constructor and Method

In a nutshell, following are summarized as the key difference between a constructor and a method.

1. The purpose of constructor is to initialize the object of a class while the purpose of a method is to perform a task by executing java code.
2. Constructors cannot be abstract, final, static and synchronized while methods can be.
3. Constructors do not have return types while methods do.