

Object Oriented Programming in Java

Er.Sital Prasad Mandal

BCA- 2nd sem

Mechi Campus

Bhadrapur, Jhapa, Nepal

(Email : info.sitalmandal@gmail.com)

<https://ctaljava.blogspot.com/>



Text Book

1. Deitel & Dietel. -Java: How to-program-. 9th Edition. TearsorrEducation. 2011, ISBN: 9780273759168
2. Herbert Schildt. "Java: The CoriviaeReferi4.ic e 61 Seventh Edition. McGraw -Hill 2006, 1SBN; 0072263857

3. Object Oriented Programming Concepts



Object Oriented Programming Concepts

1. Fundamentals of Classes:

- A Simple Class
- Creating Class Instances
- Adding methods to a class

2. Calling Functions/Methods

3. Abstraction

4. Encapsulation

5. Using 'this' Keyword

6. Constructors, Default constructors

7. More on methods: Passing by Value,by Reference, Access Control, Methods that Return Values

8. Polymorphism

9. Method Overloading

10. Recursion; Nested and Inner Classes

3. Object Oriented Programming Concepts

Fundamentals of Classes: A Simple Class

Classes & Objects

```
class Box
{
    private:
        // data members
    public:
        // member functions
};
```

← **Class**

Box b1

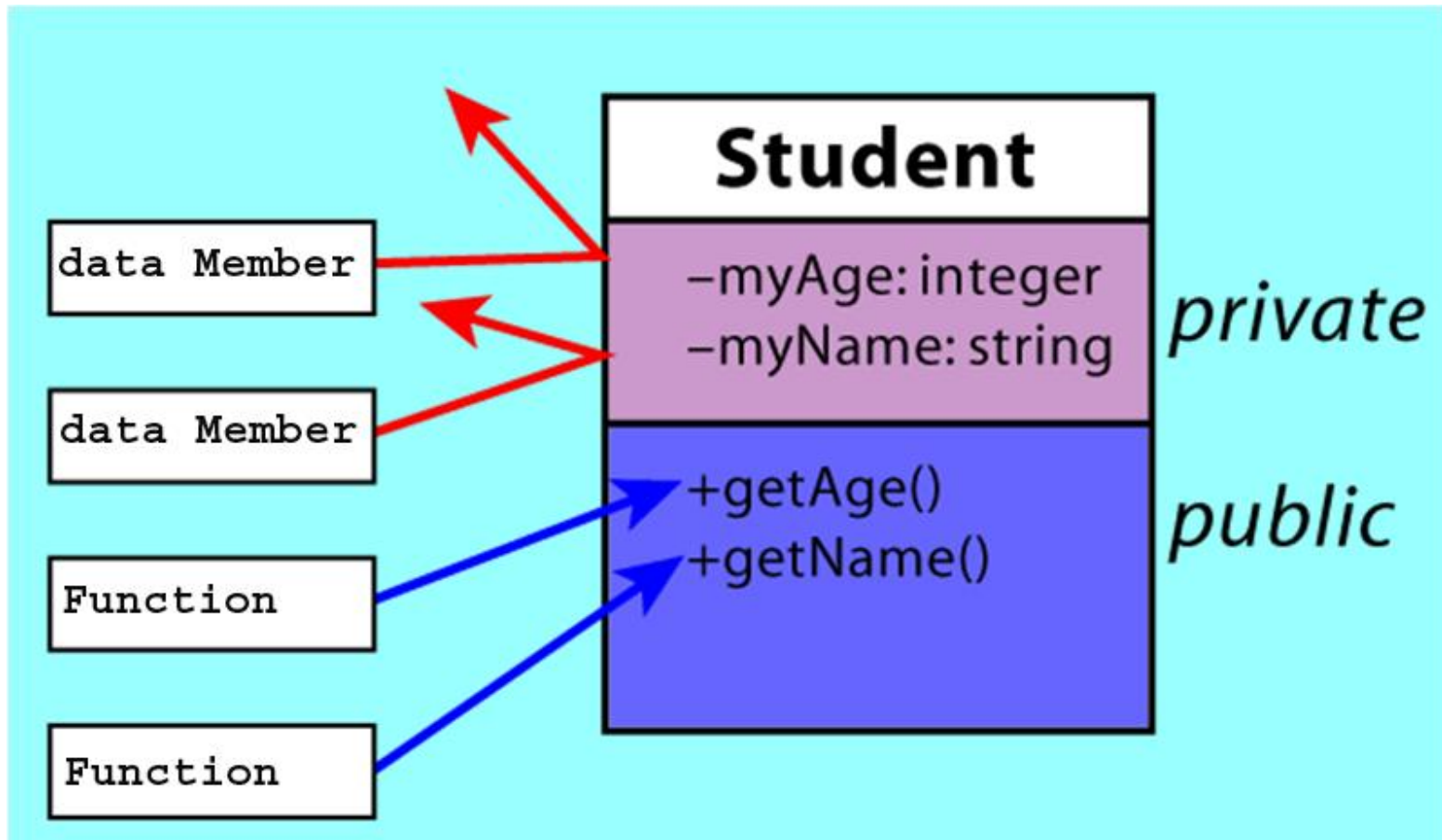
Box b2

Box b3

← **Objects**

3. Object Oriented Programming Concepts

Fundamentals of Classes: A Simple Class

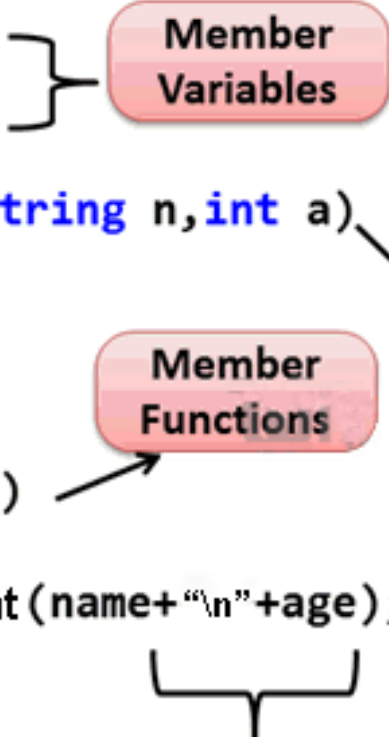


3. Object Oriented Programming Concepts

Fundamentals of Classes: A Simple Class

```
public class Student
{
    string name;
    int age;

    void setdata(string n,int a)
    {
        name = n;
        age = a;
    }
    void showdata()
    {
        System.out.print (name+"\n"+age);
    }
}
```



Depend on execution of object

```
public static void Main(String[] args)
{
    Student obj1,obj2;
    obj1 = new Student();
    obj2 = new Student();

    obj1.setdata("Ram",20);
    obj2.setdata("Shyam",10);

    obj1.showdata();
    obj2.showdata();
}
```

obj1

name = Ram
age = 20

obj2

name = Shyam
age = 20

3. Object Oriented Programming Concepts

Fundamentals of Classes: A Simple Class

Classname	<u>paul:Student</u>	<u>peter:Student</u>
Data Members	name="Paul Lee" grade=3.5	name="Peter Tan" grade=3.9
Member Functions	getName() printGrade()	getName() printGrade()

Two instances of the **Student** class

3. Object Oriented Programming Concepts

Fundamentals of Classes: A Simple Class

Classname (Identifier)	Student	Circle
Data Member (Static attributes)	name grade	radius color
Member Functions (Dynamic Operations)	getName() printGrade()	getRadius() getArea()

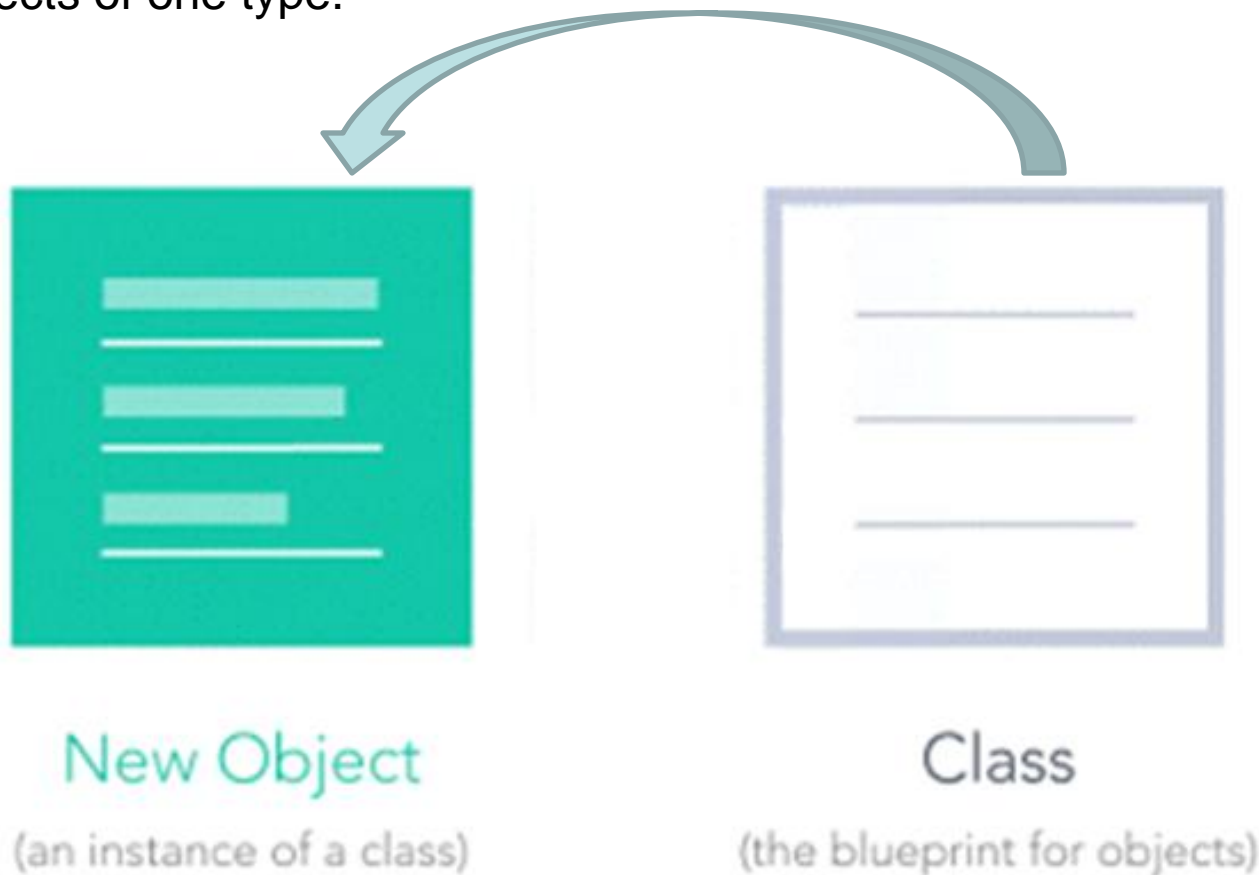
SoccerPlayer	Car
name number xLocation yLocation	plateNumber xLocation yLocation speed
run() jump() kickBall()	move() park() accelerate()

Examples of classes

3. Object Oriented Programming Concepts

Fundamentals of Classes: A Simple Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.



3. Object Oriented Programming Concepts

Fundamentals of Classes: A Object



It is a basic unit of Object-Oriented Programming and represents real life entities.

A typical Java program creates many objects, Which interact by invoking methods.

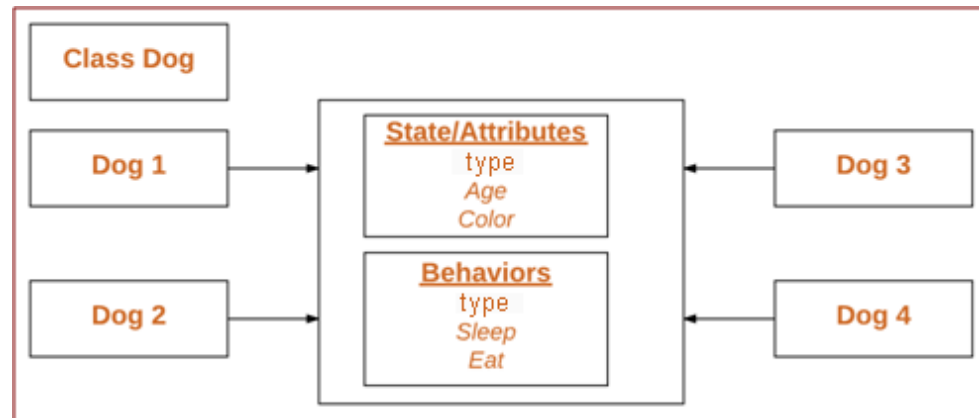
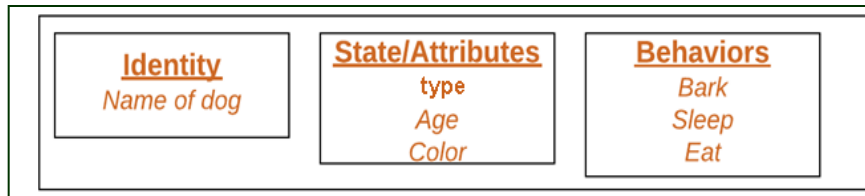
An object consists of :

State: It is represented by attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog



3. Object Oriented Programming Concepts

Initializing an object: new operator also invokes the class constructor

```
// Class Declaration
public class Dog
{
    // Instance Variables
    String name;
    String type;
    int age;
    String color;
    // Constructor Declaration of Class
    public Dog(String name, String type,
               int age, String color)
    {
        this.name = name;
        this.type = type;
        this.age = age;
        this.color = color;
    }
    // method 1
    public String getName()
    {
        return name;
    }
    // method 2
    public String getType()
    {
        return type;
    }
}
```

```
// method 3
    public int getAge()
    {
        return age;
    }
    // method 4
    public String getColor()
    {
        return color;
    }

    @Override
    public String toString()
    {
        return("Hi my name is " + this.getName()+
               ".\n type : " + this.getType() +
               ".\n age : " + this.getAge() +
               ".\n color: " +
               this.getColor());
    }

    public static void main(String[] args)
    {
        Dog d1= new Dog("Max","bulldog", 5,
                        "white");
        System.out.println(d1.toString());
    }
}
```

3. Object Oriented Programming Concepts

Assignment

- 
1. Ways to create an object of a class
 2. Creating multiple objects by one type only (A good practice)

<https://www.geeksforgeeks.org/classes-objects-java/>

Difference between abstract class and interface

3. Object Oriented Programming Concepts


Abstraction

Example for Java Abstraction



	Owner
	<ul style="list-style-type: none">• Car Description• Service History• Petrol Mileage History

	Registration
	<ul style="list-style-type: none">• Vehicle Identification Number• License plate• Current Owner• Tax due, date

	Garage
	<ul style="list-style-type: none">• License plate• Work Description• Billing Info• Owner

Here, you can see that an **Owner** is interested in details like Car description, service history, etc; **Garage Personnel** are interested in details like License, work description, bill, owner, etc; and **Registration Office** interested in details like vehicle identification number, current owner, license plate, etc.

It means each application identifies the details that are important to it.

<https://ctaljava.blogspot.com/>

3. Object Oriented Programming Concepts


Abstraction

Example for Java Abstraction



	Owner
	<ul style="list-style-type: none">• Car Description• Service History• Petrol Mileage History

	Registration
	<ul style="list-style-type: none">• Vehicle Identification Number• License plate• Current Owner• Tax due, date

	Garage
	<ul style="list-style-type: none">• License plate• Work Description• Billing Info• Owner

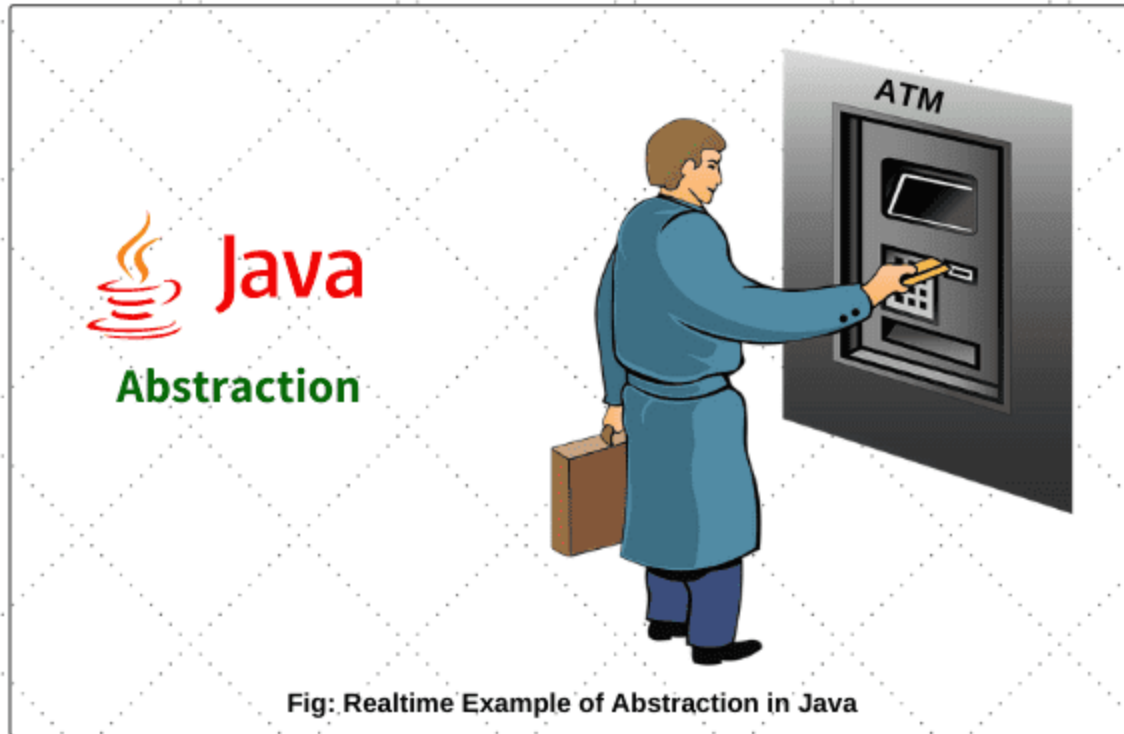
Here, you can see that an **Owner** is interested in details like Car description, service history, etc; **Garage Personnel** are interested in details like License, work description, bill, owner, etc; and **Registration Office** interested in details like vehicle identification number, current owner, license plate, etc.

It means each application identifies the details that are important to it.

<https://ctaljava.blogspot.com/>

3. Object Oriented Programming Concepts

Abstraction



Let's first take ATM machine as a real-time example. We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc. in our daily life. But we don't know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.

3. Object Oriented Programming Concepts

Abstraction

- We try to obtain an **abstract view**, model or structure of a real life problem, and reduce its unnecessary details.
- With definition of properties of problems, including the data which are affected and the operations which are identified, the model abstracted from problems can be a standard solution to this type of problems.
- It is an efficient way since there are ambiguous real-life problems that have similar properties.

In simple terms, it is hiding the unnecessary details & showing only the essential parts/functionalities to the user.

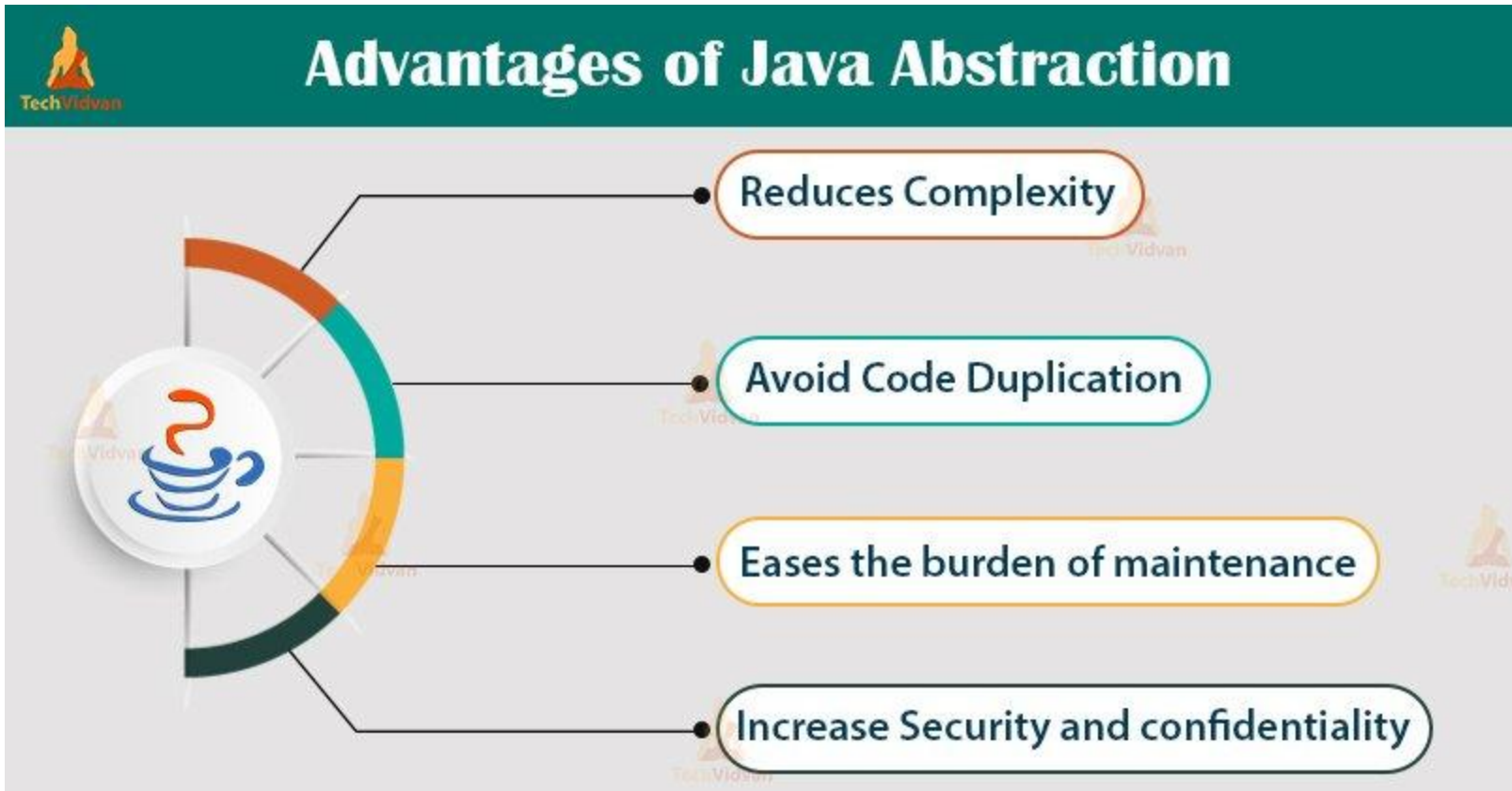
Data binding : Data binding is a process of binding the application UI and business logic. Any change made in the business logic will reflect directly to the application UI.

Abstraction is achieved in 2 ways :

1. Abstract class
2. Interfaces (Pure Abstraction)

3. Object Oriented Programming Concepts

Abstraction



3. Object Oriented Programming Concepts

Abstract class & method



```
// Abstract class
abstract class Animal {
    // Regular method
    public void sleep() {
        System.out.println("Sleeping 8 hours");
    }
    // Abstract method (does not have a body)
    public abstract void animalSound();
}

// Subclass (inherit from Animal)
class Dog extends Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("BOW BOW");
    }
}

// Subclass (inherit from Animal)
class Cat extends Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("MEOW MEOW");
    }
}

public class AbstractDemoMain {
    public static void main(String[] args) {
        Dog animal1 = new Dog(); // Create a Dog
        object
        animal1.animalSound();
        animal1.sleep();
        // Animal an = new Animal(); // wrong class
        cannot be initiated
        Animal dog = new Dog();
        Animal cat = new Cat();
    }
}
```

Output:

BOW BOW
Sleeping 8 hours

3. Object Oriented Programming Concepts

Abstraction Interface



```
// Interface
interface Animal1 {
    //interface method
    void animalSound(); // (does not have a
    body)
    void sleep(); // (does not have a body)
}

// Cat "implements" the Animal interface
class Cat1 implements Animal1 {
    public void animalSound() {
        // The body of animalSound() is provided
        here
        System.out.println("The Cat says: MEOW
        MEOW");
    }

    public void sleep() {
        // The body of sleep() is provided here
        System.out.println("Zzz");
    }
}
```

```
public class InterfaceAbstractDemoMain {
    public static void main(String[] args) {
        Cat1 myCat = new Cat1(); // Create
        a Cat object
        myCat.animalSound();
        myCat.sleep();
    }
}
```

The Cat says: MEOW MEOW
Zzz

3. Object Oriented Programming Concepts



Assignment

Notes on Interfaces:

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default **abstract** and **public**
- Interface attributes are by default **public**, **static** and **final**
- An interface cannot contain a constructor (as it cannot be used to create objects)

Why And When To Use Interfaces?

- 1) To achieve security - hide certain details and only show the important details of an object (interface).
- 2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces.

3. Object Oriented Programming Concepts



Encapsulation in Java

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

To achieve this, you must:

- declare class variables/attributes as private
- provide public get and set methods to access and update the value of a private variable

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.



Capsule

3. Object Oriented Programming Concepts

Simple Example of Encapsulation in Java

File: Student.java

//A Java class which is a fully encapsulated class.

//It has a private data member and getter and setter methods.

```
package com.javatpoint;
public class Student{
//private data member
private String name;
//getter method for name
public String getName(){
return name;
}
//setter method for name
public void setName(String name){
this.name=name
}
}
```

File: Test.java

//A Java class to test the encapsulated class.

```
package com.javatpoint;
class Test{
public static void main(String[] args){
//creating instance of the encapsulated class
Student s=new Student();
//setting value in the name member
s.setName("vijay");
//getting value of the name member
System.out.println(s.getName());
}
}
```

3. Object Oriented Programming Concepts

Simple Example of Encapsulation in Java

```
class Area {  
    // fields to calculate area  
    int length;  
    int breadth;  
    // constructor to initialize values  
    Area(int length, int breadth) {  
        this.length = length;  
        this.breadth = breadth;  
    }  
    // method to calculate area  
    public void getArea() {  
        int area = length * breadth;  
        System.out.println("Area: " + area);  
    }  
}
```

Area: 30

```
class EncapsulationAreaRectangle {  
    public static void main(String[] args) {  
        // create object of Area  
        // pass value of length and breadth  
        Area rectangle = new Area(5, 6);  
        rectangle.getArea();  
    }  
}
```

3. Object Oriented Programming Concepts



Using 'this' Keyword

The keyword “this” in Java is a reference variable. **The reference variable “this” points to the current object in the Java program.** Hence if you want to access any member or function of the current object, then you can do so by using ‘this’ reference.

What are the 6 ways to use this keyword in Java?

1. this can be used to get the current object.
2. this can be used to invoke current object's method.
3. this() can be used to invoke current class constructor
4. this can be passed as a parameter to a method call.
5. this can be passed as a parameter to a constructor.
6. this can be used to return the current object from the method.

3. Object Oriented Programming Concepts



Using 'this' Keyword

The keyword “this” in Java is a reference variable. **The reference variable “this” points to the current object in the Java program.** Hence if you want to access any member or function of the current object, then you can do so by using ‘this’ reference.



What are the 6 ways to use this keyword in Java?

1. this can be used to get the current object.
2. this can be used to invoke current object's method.
3. this() can be used to invoke current class constructor
4. this can be passed as a parameter to a method call.
5. this can be passed as a parameter to a constructor.
6. this can be used to return the current object from the method.

3. Object Oriented Programming Concepts



Using 'this' Keyword

Access Instance Variable Using 'this'

```
class Test
{
    int val1;
    int val2;
    // Parameterized constructor
    Test(int val1, int val2)
    {
        this.val1 = val1 + val1;
        this.val2 = val2 + val2;
    }
    void display()
    {
        System.out.println("Value val1 = " + val1 + " Value val2 = " + val2);
    }
}
```

```
class this_Test{
    public static void main(String[] args)
    {
        Test object = new Test(5,10);
        object.display();
    }
}
```

Output:

```
Value val1 = 10 Value val2 = 20
```

3. Object Oriented Programming Concepts



Using 'this' Keyword

'this' Passed As The Method Parameter

```
class Test_method
{
    int val1;
    int val2;
    Test_method()
    {
        val1 = 10;
        val2 = 20;
    }
    void printVal(Test_method obj)
    {
        System.out.println("val1 = " + obj.val1 + "   val2 = " + obj.val2);
    }
    void get()
    {
        printVal(this);
    }
}

public class this_method{
    public static void main(String[] args)
    {
        Test_method object = new Test_method();
        object.get();
    }
}
```

Output:

```
val1 = 10   val2 = 20
```

3. Object Oriented Programming Concepts

Frequently Asked Questions “this”

Q #1) What is the difference between this and this () in Java?

Answer: In Java, this refers to the current object while this () refers to the constructor with matching parameters. The keyword ‘this’ works only with objects. The call “this ()” is used to call more than one constructor from the same class.

Q #2) Is this keyword necessary in Java?

Answer: It is necessary especially when you need to pass the current object from one method to another, or between the constructors or simply use the current object for other operations.

Q #3) What is the difference between this () and super () in Java?

Answer: Both this () and super () are keywords in Java. While this () represents the constructor of the current object with matching parameters, super () represents the constructor of the parent class.

Q #4) Can you use both this () and super () in a constructor?

Answer: Yes, you can use it. The constructor this () will point to the current constructor while super () will point to the parent class constructor. Remember that both this () and super () should be the first statement.

3. Object Oriented Programming Concepts

Frequently Asked Questions “this”

<https://www.studytonight.com/java/this-keyword-in-java.php>

3. Object Oriented Programming Concepts



Constructors

A constructor in Java is similar to a method that is invoked when an object of the class is created.

In [Java methods](#), a constructor has the same name as that of the class and does not have any return type. For example,

```
class Test {  
    Test() {  
        // constructor body  
    }  
}
```

Here, Test() is a constructor. It has the same name as that of the class and doesn't have a return type.

<https://www.softwaretestinghelp.com/java-constructor/>

<https://ctaljava.blogspot.com/>

3. Object Oriented Programming Concepts

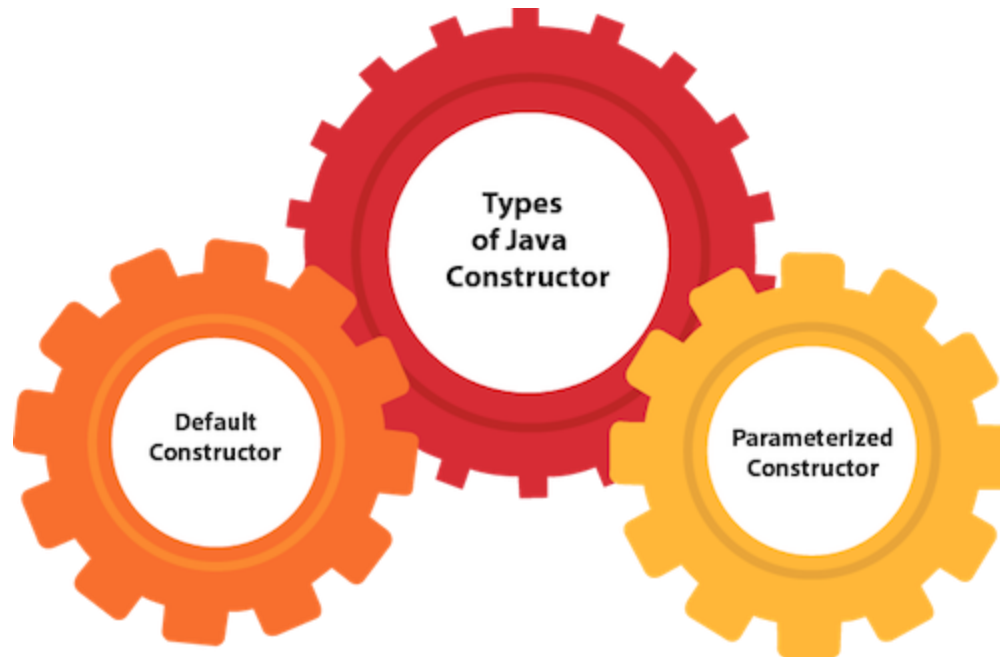


Constructors

Types of Constructor

In Java, constructors can be divided into 3 types:

1. Default Constructor
2. No-Arg Constructor
3. Parameterized Constructor



3. Object Oriented Programming Concepts



Constructors

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>(){  
}
```

```
class Default_Constructor {  
    int a;  
    boolean b;  
    public static void main(String[] args) {  
  
        // A default constructor is called  
        Default_Constructor obj = new Default_Constructor();  
  
        System.out.println("Default Value:");  
        System.out.println("a = " + obj.a);  
        System.out.println("b = " + obj.b);  
    }  
}
```

Output:
Default Value:
a = 0
b = false

3. Object Oriented Programming Concepts

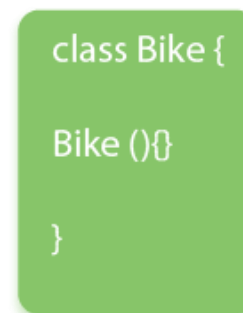
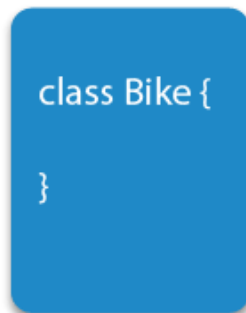


Constructors

Java Default Constructor

//Java Program to create and call a default constructor

```
class Bike1{  
    //creating a default constructor  
    Bike1(){System.out.println("Bike is created");}  
    //main method  
    public static void main(String args[]){  
        //calling a default constructor  
        Bike1 b=new Bike1();  
    }  
}
```



Output:
Bike is created

3. Object Oriented Programming Concepts



Constructors

Java Parameterized Constructor

```
public class Main {  
    int x;  
  
    public Main(int y) {  
        x = y;  
    }  
  
    public static void main(String[] args) {  
        Main myObj = new Main(5);  
        System.out.println(myObj.x);  
    }  
}
```

// Outputs 5

Output:
Bike is created

3. Object Oriented Programming Concepts



Important Notes on Java Constructors

- Constructors are invoked implicitly when you instantiate objects.
- The two rules for creating a constructor are:
 - The name of the constructor should be the same as the class.
 - A Java constructor must not have a return type.
- If a class doesn't have a constructor, the Java compiler automatically creates a **default constructor** during run-time. The default constructor initializes instance variables with default values. For example, the `int` variable will be initialized to 0.
- Constructor types:
 - No-Arg Constructor** - a constructor that does not accept any arguments
 - Parameterized constructor** - a constructor that accepts arguments
 - Default Constructor** - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
- A constructor cannot be `abstract` OR `static` OR `final`.
- A constructor can be overloaded but can not be overridden.

3. Object Oriented Programming Concepts



Assignment

Difference between constructor and method in Java

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
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 the class name.

3. Object Oriented Programming Concepts



Passing by Value,by Reference

Pass by Value: The method parameter values are copied to another variable and then the copied object is passed, that's why it's called pass by value.

Pass by Reference: An alias or reference to the actual parameter is passed to the method, that's why it's called pass by reference.

3. Object Oriented Programming Concepts



Passing by Value,by Reference

Pass by Value: The method parameter values are copied to another variable and then the copied object is passed, that's why it's called pass by value.

Pass by Reference: An alias or reference to the actual parameter is passed to the method, that's why it's called pass by reference.

Basically,

- **pass-by-value** means that the actual value of the variable is passed.
- **pass-by-reference** means the memory location is passed where the value of the variable is stored.

3. Object Oriented Programming Concepts



Passing by Value,by Reference

Java Pass By Value Example

```
public class Call_by_value {
```

```
/*
```

```
 * The original value of a will remain unchanged in  
 * case of call-by-value
```

```
*/
```

```
int a = 10;
```

```
void call(int a) {
```

```
    // this local variable a is subject to change in its value
```

```
    a = a+10;
```

```
}
```

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

```
    Call_by_value eg = new Call_by_value();
```

```
    System.out.println("Before call-by-value: " + eg.a);
```

```
/*
```

```
 * Passing an integer 50 to the call() method. The value of  
 * 'a' will still be unchanged since the passing parameter is a  
 * primitive type.
```

```
*/
```

```
    eg.call(50);
```

```
    System.out.println("After call-by-value: " + eg.a);
```

```
}
```

```
}
```

<https://ctaljava.blogspot.com/>

<terminated> Example [Java Application] C:\Program

Before call-by-value: 10

After call-by-value: 10

3. Object Oriented Programming Concepts

Passing by Value,by Reference

```
public class Pass_by_Reference {    Pass by Reference Example
    /*
     *  The original value of 'a' will be changed as we are trying
     *  to pass the objects. Objects are passed by reference.
     */
    int a = 10;
    void call(Pass_by_Reference eg) {
        eg.a = eg.a+10;
    }
    public static void main(String[] args) {
        Pass_by_Reference eg = new Pass_by_Reference();
        System.out.println("Before call-by-reference: " + eg.a);
        // passing the object as a value using pass-by-reference
        eg.call(eg);
        System.out.println("After call-by-reference: " + eg.a);
    }
}
```

```
<terminated> Example [Java Application] C:\Program Files\Java\jre
Before call-by-reference: 10
After call-by-reference: 20
```


3. Object Oriented Programming Concepts

Motivate

