



Classes and Objects



OBJECTIVES OF THIS CHAPTER

- 7.1 Class
- 7.2 Objects
- 7.3 Static Members
- 7.4 Constructors
- 7.5 Overloading in Java (Methods and Constructors)

INTRODUCTION:

Languages like BASIC, C are considered to be Procedural Languages where each program is nothing more than a crisscross of functions. There is a lack of data binding among function and data members. Such a programming type fails to design a robust application. To deal with this drawback, java offers Object Oriented Programming paradigm which is also known as OOP. Here, “**Object**” means a real-world entity such as a pen, chair, table, computer, watch, etc. An Object-Oriented Programming is a methodology to design a program using classes and objects. This programming technique simplifies software development and maintenance by providing some concepts like Inheritance, Polymorphism, Abstraction etc. Let's learn more about OOPs by understanding class and object.

7.1 CLASS:

A **class** in java is a user defined prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, a class can be defined as a template/blueprint that describes the behavior and/or state that the object of its type supports. Thus, we can simply say, “class in Java determines how an object will behave and what the object will contain”. Usually a class contains fields and methods related to an object to be used in a program. We can explain the concept of Class and Object in the form of a diagram as follows:

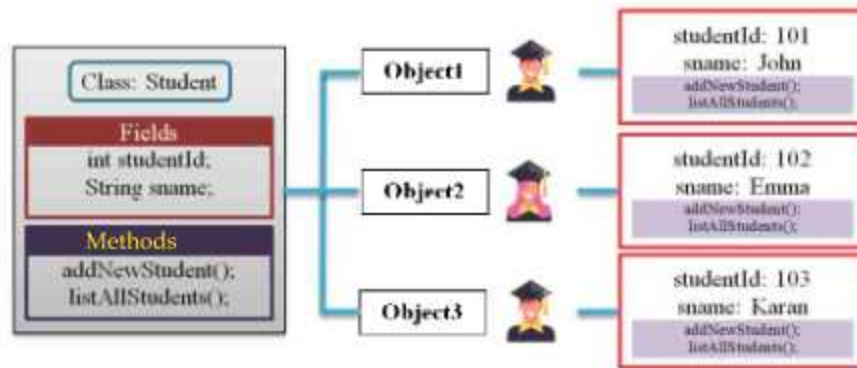


Fig 7.1 Concept of Class and Objects

In Java, a class declaration can include some of the components given below.

7.1.1 Basic Components of Java Class:

- **Modifiers** : Modifiers are the keywords which describes the access rights of members of a class. In java, a class can be public or has default access i.e. friendly.
- **Class Keyword** : class keyword is used to create a class.
- **Class Name** : This is an identifier of class. This name of class should begin with letter. As per java naming conventions, first letter of each class name should be capital.
- **Body** : The class body is surrounded by braces, i.e. { }. Body part of java class can contain two type of elements.
 - o Fields
 - o Methods

Each of these components are having their own importance. A general syntax of creating a class is as under:

```
[Modifier] class ClassName
{
  Body of Java Class
  // Fields
  // Methods
}
```

Note : Modifier shown in square brackets specifies that it is optional part of class.

Example Program 7.1:

```
public class CompApp
{
    int ch1,ch2;           //Instance Variables
    static String title;   // Class Variables

    void showTitle()
    {
        int a=10;         // Local Variables
        System.out.println("Title of Chap. is "+title);
        -----
    }
}
```

Fields OR Member Variables

Methods

7.1.2 Fields (Member Variables) in Class:

Variables declared inside the class are called Fields. We can declare fields with different access specifiers such as private, public, protected etc. We can categorize the member variable in following types:

- **Instance Variables** : These are the variables that are inherent to an object and that can be accessed from inside any method, constructor or block. They are destroyed when the object is destroyed.
- **Class Variables** : Class variables or static variables are declared with the static keyword in a class. They are similar to instance variables, but they are created when the program starts and destroyed when the program stops. The main difference with instance variables is in what scope they are available. A class variable can be accessed with class name, while an instance variable is accessible only by class object.

7.1.3 Methods in Class:

A method in Java is a group of instructions that performs a specific task. It provides the reusability of code. We can divide a complex problem into smaller parts known as modules which makes our program easy to understand and reusable. In Java, there are two basic types of methods:

- **Standard Library Methods**: These type of methods are also known as Pre-defined Methods. These built-in methods in Java are already defined in Java library. Examples of this type of method can be print() method that comes that under java.io. PrintStream which prints the string that is written within the quotation. sqrt() is another method of Math class which returns the square root of specific number given as an argument.
- **User-Defined Methods**: We can create our own method based on our requirements in java. All methods designed by user itself are called "User

Defined Methods". We can create any number of methods in a java class to define behavior of our Object.

User can design the method according to the requirement by using data fields or local variables and statements to preform requisite task. We can define local variable of a method as under:

- **Local variables:** These are the temporary variables defined inside methods. They are declared and initialized within that method, and will be made eligible for garbage collection once the execution of method is completed. These variables are not accessible outside the methods they are declared in.

7.1.4 Access Modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of these elements by applying the access modifier on them. There are four access modifiers in java.

- **Private:** Accessible within same class only.
- **Default:** Accessible within same class and same package only.
- **Protected:** Accessible within same class, same package and outside the package using subclass only.
- **Public:** Accessible anywhere outside the class and package.

We can easily understand the difference between all these access modifiers with the help of a table given following:

Access Modifier	Within Class	Within Package	Outside Package by Subclass only	Outside Package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

Table 7.1 Access Modifiers in Java

Note : Java package is a group of similar type of classes, interfaces and subpackages. Think of it as a folder in a file system.

Following program shows the basic program of Java class with its component

```
class FindArea
{
    float radius, area;           //Fields
    void getRadius(float r)       //Method
    {
```

```

        radius=r;
    }
    void showArea ()                //Method
    {
        final float PIE=(float)3.143;
        area=PIE*radius*radius;
        System.out.println("The Area of Circle is: "+area);
    }
}

```

7.2 OBJECTS

A Java object is a member (also called an instance) of a Java class. Each object has an identity, a behavior and a state. The identity is an internal ID assigned to each object, state of an object is stored in fields (variables), while methods (functions) display the object's behavior. Objects are created at runtime from templates, also known as classes. It is a basic unit of Object-Oriented Programming and represents real life entities. A typical Java program creates many objects, which as we know, interact by invoking methods.

If we consider the real-world, we can find many objects around us like cars, dogs, humans, etc. All these objects have a state and a behavior. In the case of a dog as an example of Object, its states can be - name, breed, color, and the behavior is - barking, wagging the tail, running. Similarly, in software development, methods operate on the internal state of an object and the object-to-object communication is also done via methods. An object has two characteristics:

- **State:** represents the data (value) stored in every field of an object.
- **Behavior:** represents the functionality of an object given in the form of method, such as read Value, display Value etc.

We shall have a look on all these terms in the form of a program later, and try to understand the importance of each one.

7.2.1 Use of new Keyword in Java

The Java new keyword is used to create an instance of the class. In other words, it instantiates a class by allocating memory for a new object and returning a reference to that memory. We can also use the new keyword to create the array object. Some of the main characteristics of new keyword are as under:

- It is used to create the object.
- It allocates the memory at runtime.
- All objects occupy memory in the heap area.
- It invokes the object constructor.

7.2.2 Creating Object in Java:

The object is a basic building block of JAVA Program. We cannot use the attributes or methods of any class without creating an object. There are various ways to create an object in Java. Usually, we use **new** keyword to create an object in java. It allocates memory (heap) for the newly created object and also returns the reference of that object to allocated memory. The syntax for creating an object is:

Syntax of creating object using new Keyword:

```
ClassName Object_Name=new ClassName ([argument list-if any]);  
Or  
ClassName Object_Name;  
Object_Name =new ClassName ([argument list-if any]);
```

For Example:

```
FindArea obj=new FindArea();  
Or  
FindArea obj;  
obj=new FindArea();
```

Now we have understood the concept of Class and Objects. Lets make a simple Java programs to demonstrate Class and Objects in JAVA.

Program 7.2 (CAProg1.java) Program for creation and use of a class in Java

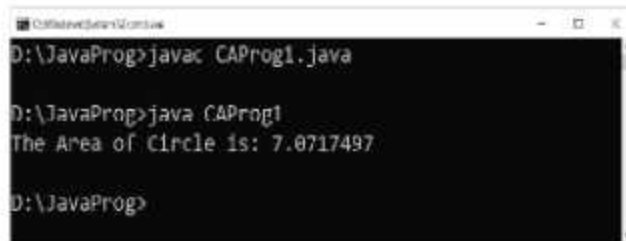
```
class FindArea  
{  
    float radius, area;                //Fields  
    void getRadius(float r)            //Method  
    {  
        radius=r;  
    }  
    void showArea ()                   //Method  
    {  
        final float PIE=(float)3.143; //using type casting  
        area=PIE*radius*radius;  
        System.out.println("The Area of Circle is: "+area);  
    }  
} //end of Find Area Class
```

```

class CProgl
{
    public static void main(String arg[])    {
        FindArea obj=new FindArea();        //creating object
        obj.get Radius(1.5f);
        obj.showArea();
    }
}

```

Compilation, Execution and Output of Program 7.2 (CProgl.java)



```

D:\JavaProg>javac CProgl.java

D:\JavaProg>java CProgl
The Area of Circle is: 7.0717497

D:\JavaProg>

```

7.3 STATIC MEMBERS (CLASS VARIABLES)

The static modifier means that the entity to which it is applied is available outside any particular instance of the class. It also means that the static methods or attributes are a part of the class and not an object. The memory is allocated to such an attribute or method at the time of class loading. The use of a static modifier makes the program more efficient by saving memory. A static field exists across all the class instances, and can be called without creating an object of the class. The use of a static modifier can be understood in the form of a program given below:

Program 7.3 (CProgl2.java) Program for static variable/class variable

```

class StaticDataMember
{
    int Variable=10;
    static int Static_Member=10;
    void showValue()
    {
        System.out.println("Variable: "+Variable);
        System.out.println("Static_Member: "+Static_Member);
    }
}

```

```

        void incValue()
        {
            Variable++;
            Static_Member++;
        }
    }
}

class CAProg2 {
    public static void main(String arg[]) {
        StaticDataMember obj1=new StaticDataMember();
        obj1.showValue();
        obj1.incValue();
        obj1.showValue();
        obj1.incValue();
        StaticDataMember obj2=new StaticDataMember();
        obj2.showValue();
        obj2.incValue();
        obj1.showValue();
    }
}

```

Compilation, Execution and Output of Program 7.3 (CAProg2.java)

```

D:\JavaProg>javac CAProg2.java
D:\JavaProg>java CAProg2
Non_Static: 10
Static_Member: 10
Non_Static: 11
Static_Member: 11
Non_Static: 10
Static_Member: 12
Non_Static: 12
Static_Member: 13
D:\JavaProg>

```

In the given example, we can clearly see the importance of static and instance data fields in a class. We have declared two different variables named Variable and Static_Member in the given class Static Data Member. We created two methods, showValue(); to display the values of both these fields whereas incValue(); to increase the value of both these variables with one. We created two different objects named obj1 and obj2. We called the given methods with both these objects in certain order. We can notice that the static variable of the class is free from the object with which it is called. The value of the static variable

is being shared among all the objects. On the other hand, Variable (instance variable) is associated with object. It gives different values when called with different objects. This is the main function of static variable which makes the field shared among all the objects of the same class.

7.4 CONSTRUCTORS IN JAVA

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. In Java, a constructor is a block of codes similar to the method but having some special properties. Constructor is called automatically when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated. It is a special type of method which is mainly used to initialize the object. We can explain the difference between method and constructor as under:

Method	Constructor
Method can be named as per the operation performed by it.	Constructors must have the same name as the class within which it is defined.
Method can return upto one value of any given type. However any number of arguments can be passed to a method.	Constructors do not return any value of any type. However any number of arguments can be passed to a constructor similar to the method.
A Method can be called explicitly any number of time when required.	Constructors are called only once at the time of Object creation.

7.4.1 Rules to be followed while defining constructors

There are some rules for creating constructors in java class. We can explain some of them as follows:

- Constructor of a class must have the same name as the class name in which it is declared.
- Access modifiers can be used in constructor declaration to control its access.
- A constructor in Java cannot be abstract, final, static, or Synchronized.
- A Constructor must have no explicit return type, even not void type.

7.4.2 Types of Constructors in Java

Now is the correct time to discuss the types of the constructor, so primarily there are two types of constructors in java:

1. **Default Constructor (No-Argument Constructor)** : A constructor that has no parameter is known as the default constructor. If we don't define a constructor in a class, then the compiler automatically creates a default constructor(with no arguments) for the class. And if we write a constructor

with arguments or no arguments then the compiler does not automatically create a default constructor. The default constructor initializes any uninitialized instance variables with default values as per given bellow:

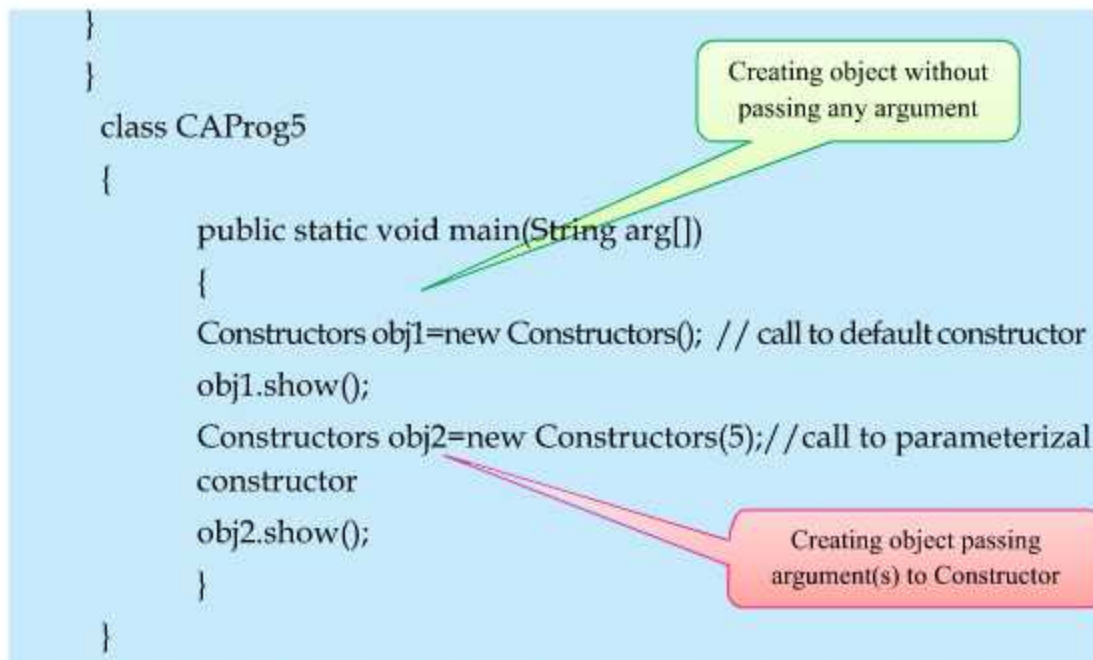
Type	Default Value
boolean	false
byte	0
short	0
int	0
long	0L
char	\u0000
float	0.0f
double	0.0d
object	Reference null

Table: Default values of instance variables

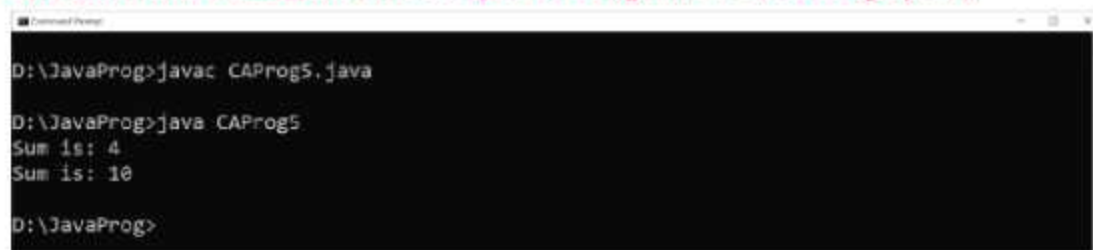
2. **Parameterized Constructor:** A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor. For example:

Program 7.4 (CAProg5.java) Java Program for Constructors

```
class Constructors
{
    int no1,no2,total;// fields
    Constructors() // default constructor
    {
        no1=no2=2;
    }
    Constructors(int inp1) // Parameterized constructor
    {
        no1=no2=inp1;
    }
    void show()//method
    {
        total=no1+no2;
        System.out.println("Sum is: "+total);
    }
}
```



Compilation, Execution and Output of Program 7.4 (CAProg5.java)



We can pass an object of a class as an argument to a constructor also. Such a constructor is known as **Copy Constructor**. Such kind of constructors are beneficial in the case when we want to create a copy of an existing object in JAVA program. For Example:

```
Constructors(Constructor obj)                //Copy Constructor  
{  
    no1=obj.no1;  
}
```

7.5 OVERLOADING IN JAVA

Overloading allows different methods or constructors to have the same name, but different signatures where the signature can differ by the number of parameters or data type of parameters or both. Overloading is related to compile-time (or static) polymorphism. Overloading can never be based on the return type of a method. We can classify the overload in following two types:

1. Method Overloading
2. Constructor Overloading

7.5.1 Method Overloading in Java

Sometime, we may need to define multiple operations associated with one common behavior of a class based on number of arguments, then we can use method overloading. In this case, multiple methods are having same name but different in number of parameters or data type of parameters. Such a way of creating methods is known as Method Overloading. It increases the readability of the program. Suppose you have to perform addition of the given numbers but there can be any number of arguments. In the given problem, if you write the method such as sum Two (int, int) for two parameters, and sum Three (int, int, int) for three parameters and so on, then it may be difficult for us as well as other programmers to efficiently use the behavior of the method because of differences in their names. We can overcome this problem using method overload.

Method Overloading can further be used in different ways :

1. **Method Overloading by changing number of arguments :** In this type of method overloading, we can have two or more methods with same name but different number of arguments. For example: we can declare first set Value() method to assign any default value to the field of a class and then another method with same name but with argument to assign the given value to the class field. Lets see the implementation in a program as follows.

Program 7.5 (CAProg6.java) Java Program for Method Overloading:

```
class CAProg6 {  
    int marks;  
    void setValues()    {  
        marks=0;  
    }  
    void setValues(int inp1)    //Method overloading  
        marks=inp1;  
    }  
    void show()        {  
        System.out.println("Marks: "+marks);  
    }  
    public static void main(String arg[])    {  
        CAProg6 obj1=new CAProg6();  
        CAProg6 obj2=new CAProg6();  
    }  
}
```

```

        obj1.setValues();
        obj2.setValues(450);
        System.out.println("Member of Object 1");
        obj1.show();
        System.out.println("Member of Object 2");
        obj2.show();
    }
}

```

Compilation, Execution and Output of Program 7.5 (CAProg6.java)



```

D:\JavaProg>javac CAProg6.java
D:\JavaProg>java CAProg6
Member of Object 1
Marks: 0
Member of Object 2
Marks: 450
D:\JavaProg>

```

As we can see in the output, when we passed no argument with setValues methods then method without argument selected for execution and marks variable assigned 0 value. When we passed 450 value as an argument to the setValues method then method with argument selected for execution and passed value assigned to the marks variable. This is a main conception of method overload by changing number of arguments.

- 2. Method Overloading by changing data type of arguments :** In this type of method overload, we can have multiple methods with same name but different data type of argument. For example: we can use setValues method to assign different values to different fields of a class. Such as:

Program 7.6 (CAProg7.java) Java Program for Method Overloading :

```

class CAProg7
{
    int marks;
    String name;
    void setValues(int inp1)    {
        marks=inp1;
        name="";
    }
    void setValues(String inp2) { //Method overloading
        marks=0;
        name=inp2;
    }
}

```

```

    }
    void show() {
        System.out.println("Marks: "+marks);
        System.out.println("Name: "+name);
    }
    public static void main(String arg[])
    {
        CAProg7 obj1=new CAProg7();
        CAProg7 obj2=new CAProg7();
        obj1.setValues("Shivpreet");
        obj2.setValues(450);
        obj1.show();
        obj2.show();
    }
} //end of class

```

Compilation, Execution and Output of Program 7.6 (CAProg7.java)



```

D:\JavaProg>javac CAProg7.java
D:\JavaProg>java CAProg7
Number of Object 1
Marks: 0
Name: Shivpreet
Number of Object 2
Marks: 450
Name:
D:\JavaProg>

```

- 3. Method Overloading by changing both, number of arguments and data type of arguments :** As its name implies, we can overload the methods by declaring multiple methods with same name but different number of arguments as well as different data types. We can implement this type of Method Overloading according to the application requirement. We can create methods as in previous examples by making changes like:

```

    void setValues(String inp1)
    {
        .....
    }
    void setValues(int inp1,String inp2)
    {
        .....
    }

```

- 4. Based on the sequence of data types in parameters:** The method overloading also depends on the ordering of data types of parameters within the method. We can use this type of Method Overloading by creating methods in previous examples with different arguments like:

```

void setValues(String inp1, int inp2)
{
    .....
}
void setValues(int inp1, String inp2)
{
    .....
}

```

7.5.2 Constructor Overloading in Java

We can also overload constructors as we do in method overloading. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task. Constructors are used to initialize the class fields. Sometimes, we need to pass different variables in different combination of arguments. In that case, constructor overloading becomes more convenient to do the needful. As like methods, we can also use constructor overloading in following types:

1. **Constructor Overloading by changing number of arguments :** In this type of Constructor overloading, we can have two or more Constructors with different number of arguments. For example: we can assign same value to all the fields of a class in one constructor and different values to different fields in other constructor.

Program 7.7 (CAProg8.java) Program for Constructor Overloading

```

class CAProg8
{
    int studentId;
    CAProg8() { //default consturtor
        studentId=101;
    }
    CAProg8(int sid) { // overloading constructor
        studentId=sid;
    }
    void show() //method
    {
        System.out.println("Your student ID is: "+studentId);
    }
    public static void main(String arg[])
    {
        CAProg8 obj1=new CAProg8();
        CAProg8 obj2=new CAProg8(2462);
    }
}

```

```

        System.out.println("Member of Object 1");
        obj1.show();
        System.out.println("-----\n Member of Object 2");
        obj2.show();
    }
}

```

Compilation, Execution and Output of Program 7.7 (CAProg8.java)



```

D:\JavaProg>javac CAProg8.java

D:\JavaProg>java CAProg8
Member of Object 1
Your student ID is: 101
-----
Member of Object 2
Your student ID is: 2462
D:\JavaProg>

```

2. **Constructor Overloading by changing data type of arguments :** In this type of Constructor overloading, we can have multiple Constructors with different data type of arguments. For example: we can declare two Constructors to assign integer value in one constructor and string value in other one. As we have studied in Method Overloading, we can use this type of constructor overloading using following constructors:

```

CAProg8(int inp1)
{
    .....
}

CAProg8 (String inp2) //overloading constructor with different type
of arguments
{
    .....
}

```

3. **Constructor Overloading by changing both, number of arguments and data type of arguments :** As we can understand by its name, constructors can overload by giving different number of arguments as well as different data type: For example: we can declare multiple constructor with only one integer argument, both integer and string argument and/or only string argument. As we have studied in Method Overloading, we can use this type of constructor overloading using following constructors:

```

CAProg8(int inp1)
{
.....
}
CAProg8 (int inp1,String inp2)
{
.....
}

```

4. **Based on the sequence of data types in parameters :** The constructor overloading also depends on the ordering of data types of parameters within the constructor. As we have studied in Method Overloading, we can use this type of constructor overloading using following constructors:

```

CAProg8(String inp1, int inp2)
{
.....
}
CAProg8 (int inp1,String inp2)
{
.....
}

```

7.5.3 Advantages of Overloading in Java:

There are several advantages of Overloading in JAVA

1. Overloading in java improves code re-usability and readability.
2. Overloading in java offers flexibility to call similar methods with different types of data.
3. By using overloading in java, it is easier to remember one method name instead of multiple names. We can also create objects in different ways to define the default values of fields in different cases.
4. Consistency in naming method in java can be achieved using Overloading in methods of a class.
5. We can pass different amount of data to an object during instantiation process using constructor overloading.



Points to Remember

1. A class can be defined as a template/blueprint that describes the behavior and/or state that the object of its type supports.
2. Modifiers are the keywords which describes the access rights of members of a class.
3. Variables declared inside the class are called Fields.
4. Instance Variable are variables that are inherent to an object and that can be accessed from inside any method, constructor or block of a class.
5. Class variables or static variables are declared with the static keyword in a class. A class variable is accessible from an object instance, while an instance variable is not accessible from a static method.
6. A method in Java is a group of instructions that performs a specific task. It provides the reusability of code.
7. A Java object is a member (also called an instance) of a Java class. Each object has an identity, a behavior and a state.
8. The Java new keyword is used to create an instance of the class.
9. A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created.
10. Constructor of a class must have the same name as the class name in which it is declared.
11. A constructor that has no parameter is known as the default constructor.
12. Method Overloading means multiple methods are having same name but different in number of parameters or data type of parameters.
13. We can also overload constructors as we do in method overloading.

Exercise



Que:1 Multiple Choice Questions:

- i. An instance of a Java class is known as:
A. Method
B. Field
C. Object
D. Constructor
- ii. Declaring multiple methods with same name but different number of arguments is called.
A. Method Overloading
B. Declaration
C. Inheritance
D. None of These

- iii. _____ is invoked automatically when we create an object of a class.
- | | |
|------------------|----------------|
| A. Field | B. Method |
| C. Static Member | D. Constructor |
- iv. A constructor that has no parameter is known as :
- | | |
|------------------------------|----------------------------|
| A. Default Constructor | B. Constructor Overloading |
| C. Parameterized Constructor | D. Weak Constructor |
- v. A Constructor can not be _____.
- | | |
|-------------|---------------------|
| A. Abstract | B. Final |
| C. Static | D. All of the above |

Que:2 Fill in the blanks:

- i. A class can be defined as a _____ of all its objects.
- ii. _____ are the variables declared inside the class.
- iii. _____ keyword is used to create an object in Java.
- iv. Method Overloading can be achieved based on _____ or _____.
- v. Declaring multiple constructor in a class is called _____.

Que:3 Short Answer type Questions:

- i. Define class in JAVA.
- ii. What are the basic components of java class?
- iii. Explain Fields in a class.
- iv. What do you mean by Object?
- v. Define Instance variable?
- vi. What do you mean by class variable?
- vii. What are methods in JAVA?
- viii. Explain Private and Protected modifier.
- ix. How an object of a class can be created?
- x. What is constructor?
- xi. Explain the difference between Methods and Constructors.

Que:4 Long Answer type Questions:

- i. What is class? Explain Fields and methods in class.
- ii. Define Constructors. Explain different types of constructors with suitable example.
- iii. What do you mean by Method overloading? Explain any two ways of method loading.
- iv. How constructor Overloading take place? Write a program to explain different types of constructor overloading.