University of Engineering & Management, Kolkata

Department of CSE (IoT)

Subject Name: Advanced Programming

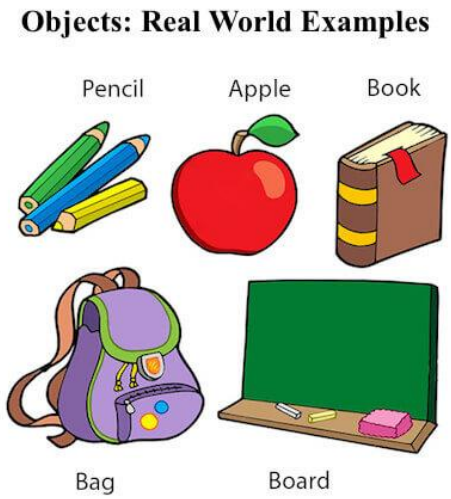
Subject Code: PCCCS405

Batch-2023-2027_2nd Year_4th Semester

Objects and Classes in Java

In object oriented programming, object and class plays vital role in programming. These are the two main pillars of OOP. Without object and class, we cannot create a program in Java. So, in this section, we are going to discuss about **objects and classes in Java.**

What is an object in Java?



An object is a real-word entity that has state and behaviour. In other words, an object is a tangible thing that can be touch and feel, like a car or chair, etc. are the example of objects. The banking system is an example of an intangible object. Every object has a distinct identity, which is usually implemented by a unique ID that the JVM uses internally for identification.

Characteristics of an Object:

- **State:** It represents the data (value) of an object.
- **Behavior:** It represents the behavior (functionality) of an object such as deposit, withdraw, etc.

- **Identity:** An object's identity is typically implemented via a unique ID. The ID's value is not visible to the external user; however, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

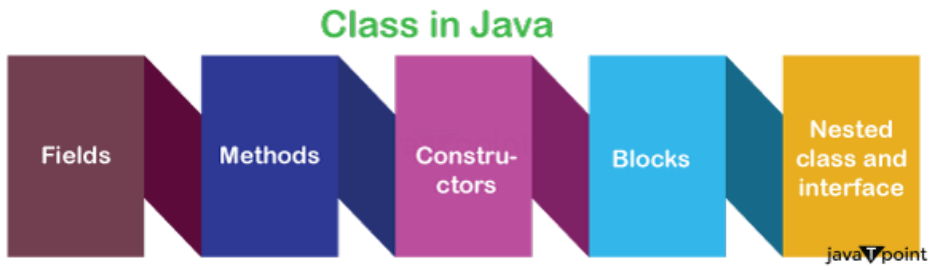
Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

What is a Class in Java?

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:



1. Fields

Variables stated inside a class that indicate the status of objects formed from that class are called fields, sometimes referred to as instance variables. They specify the data that will be stored in each class object. Different access modifiers, such as public, private, and protected, can be applied to fields to regulate their visibility and usability.

2. Methods

Methods are functions defined inside a class that includes the actions or behaviors that objects of that class are capable of performing. These techniques allow the outside world to function and change the object's state (fields). Additionally, methods can be void (that is, they return nothing) or have different access modifiers. They can also return values.

3. Constructors

Constructors are unique methods that are used to initialize class objects. When an object of the class is created using the new keyword, they are called with the same name as the class. Constructors can initialize the fields of an object or carry out any additional setup that's required when an object is created.

4. Blocks

Within a class, Java allows two different kinds of blocks: instance blocks, commonly referred to as initialization blocks and static blocks. Static blocks, which are usually used for static initialization, are only executed once when the class is loaded into memory. Instance blocks can be used to initialize instance variables and are executed each time a class object is generated.

5. Nested Class and Interface

Java permits the nesting of classes and interfaces inside other classes and interfaces. The members (fields, methods) of the enclosing class are accessible to nested classes, which can be static or non-static. Nested interfaces can be used to logically group related constants and methods together because they are implicitly static.

Syntax of a Class

```
1. class <class_name>{
2.     field;
3.     method;
4. }
```

Instance Variable in Java

A variable created inside the class but outside the method is known as an instance variable. An instance variable does not get memory at compile time; it gets memory at runtime when an object or instance is created.

Each class instance has its own copy of instance variables, which means that changes made to instance variables of one object do not affect the values of instance variables in other objects of the same class. Moreover, setter methods and constructors can be used to initialize them. In object-oriented programming, instance variables are crucial for encapsulating data within objects since they are frequently used to represent the state or properties of objects.

Method in Java

In Java method is a block of code inside a class that's intended to carry out a certain function. To provide a mechanism to interact with the state of an object and to encapsulate behaviour within objects, methods are required.

Advantages of Methods

- **Code Reusability:** Methods encourage code reusability by permitting the same block of code to be used repeatedly inside a program. Once defined, a method can be called from any area of the program where it is available.
- **Code Optimisation:** Methods allow for code optimization by enclosing intricate or repetitive functionality into reusable components. The modularization of logic facilitates readability and simplifies code maintenance.

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in the Heap memory area.

In Java, an instance of a class-also referred to as an object-is created using the new keyword. The new keyword dynamically allocates memory for an object of that class and returns a reference to it when it is followed by the class name and brackets with optional arguments.

MyClass.java

```
1. public class MyClass {
2.     int myField;
3.     public MyClass(int value) {
4.         myField = value;
5.     }
6.     public static void main(String[] args) {
7.         // Using the new keyword to create an instance of MyClass
8.         MyClass myObject = new MyClass(10);
9.         // Accessing the instance variable of the object
10.        System.out.println("Value of myField: " + myObject.myField);
11.    }
12.}
```

Output:

Value of myField: 10

Explanation

In this example, the constructor MyClass(int value) is called with the parameter 10, the reference to the newly constructed object is assigned to the variable myObject, and new MyClass(10) dynamically allocates memory for an object of type MyClass. Lastly, a printout of the instance variable myField's value is obtained.

Object and Class Example: main within the class

In Java, the main() method can be declared in a class, which is typically done in demonstration or basic programs. Having the main() method defined inside of a class allows a program to run immediately without creating a separate class containing it.

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

File: Student.java

```
1. //Java Program to illustrate how to define a class and fields
2. //Defining a Student class.
3. class Student{
4.  //defining fields
5.  int id;//field or data member or instance variable
6.  String name;
7.  //creating main method inside the Student class
8.  public static void main(String args[]){
9.   //Creating an object or instance
10.  Student s1=new Student();//creating an object of Student
11.  //Printing values of the object
12.  System.out.println(s1.id);//accessing member through reference variable
13.  System.out.println(s1.name);
14. }
15.}
```

Output:

0
Null

Explanation

Two fields are defined for the Student class in this Java programme: an int type for the id and a string type for the name. The Student class itself defines the primary method. The new keyword is used to create an object s1 of type Student inside the main procedure. The fields' default values-0 for int and null for String-are printed because they are not explicitly initialised. The values of the s1 object's name and id fields are finally printed by the programme.

Object and Class Example: main() Method Outside the Class

In real-world development, it is usual practice to organise Java classes into distinct files and to place the main method outside of the class it is intended to execute from. This strategy improves the readability, maintainability, and reusability of the code.

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

File: TestStudent1.java

```
1. //Java Program to demonstrate having the main method in
2. //another class
3. //Creating Student class.
4. class Student{
5.  int id;
6.  String name;
7. }
8. //Creating another class TestStudent1 which contains the main method
9. class TestStudent1{
10. public static void main(String args[]){
11.  Student s1=new Student();
12.  System.out.println(s1.id);
13.  System.out.println(s1.name);
14. }
15.}
```

Output:

0
Null

Explanation

The main method in this Java programme is shown to be in a different class than the Student class. There are no methods defined for the two fields, name and id, in the Student class. The main method then resides in a another class called TestStudent1, where the default constructor is used to generate an object s1 of type Student. The fields name and id are written with their default values, which are null for String and 0 for int, since they are not explicitly initialised.

Initializing Object in Java

There are the following three ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

1) Initialization through Reference Variable

Initializing an object means storing data in the object. Let's see a simple example where we are going to initialize the object through a reference variable.

File: TestStudent2.java

```
1. class Student{
2.  int id;
3.  String name;
4. }
5. class TestStudent2{
6.  public static void main(String args[]){
7.   Student s1=new Student();
8.   s1.id=101;
9.   s1.name="Sonoo";
10.  System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12.}
```

Output:

101 Sonoo

Explanation

There are two classes in this Java code: Student and TestStudent2. The two fields that the Student class defines, id and name, stand for the student's ID and name, respectively. The main method, which is the program's entry point, is specified in the TestStudent2 class. The new keyword is used to create an object s1 of type Student inside the main procedure. Next, values 101 and "Sonoo" are initialised in the id and name fields of s1.

We can also create multiple objects and store information in it through reference variable.

File: TestStudent3.java

```
1. class Student{
2.  int id;
3.  String name;
4. }
5. class TestStudent3{
6.  public static void main(String args[]){
7.   //Creating objects
8.   Student s1=new Student();
9.   Student s2=new Student();
10.  //Initializing objects
11.  s1.id=101;
12.  s1.name="Sonoo";
13.  s2.id=102;
14.  s2.name="Amit";
15.  //Printing data
16.  System.out.println(s1.id+" "+s1.name);
```

```
17.  System.out.println(s2.id+" "+s2.name);
18. }
19.}
```

Output:

101 Sonoo
102 Amit

Explanation

This Java code shows how to create and initialise many Student class objects in the TestStudent3 class. Using the new keyword, two Student objects, s1 and s2, are first created. Subsequently, each object's name and id fields are initialized independently. Id is set to 101, and the name is set to "Sonoo" for S1, and 102 and name are set to "Amit" and S2, respectively.

2) Initialization through Method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method.

Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

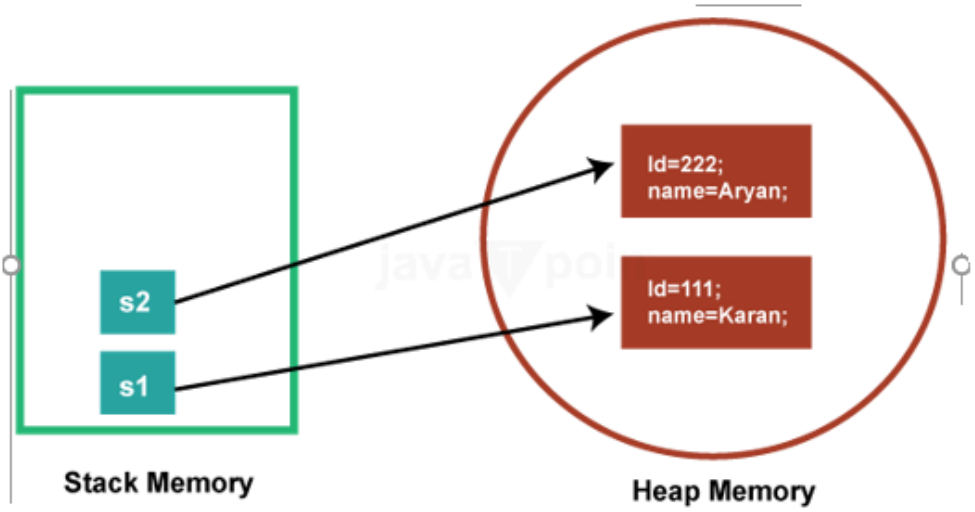
```
1.  class Student{
2.   int rollno;
3.   String name;
4.   void insertRecord(int r, String n){
5.    rollno=r;
6.    name=n;
7.   }
8.   void displayInformation(){System.out.println(rollno+" "+name);}
9.  }
10.class TestStudent4{
11. public static void main(String args[]){
12.  Student s1=new Student();
13.  Student s2=new Student();
14.  s1.insertRecord(111,"Karan");
15.  s2.insertRecord(222,"Aryan");
16.  s1.displayInformation();
17.  s2.displayInformation();
18. }
19.}
```

Output:

111 Karan
222 Aryan

Explanation

The provided Java code includes two classes: Student and TestStudent4. The Student class includes rollno and name fields, along with the methods insertRecord and displayInformation to initialise and print the respective fields' data. Two Student objects are created in the main method of the TestStudent4 class, and their corresponding insertRecord methods are called to set their rollno and name.



As we can see in the above figure, the object gets the memory in the heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 are reference variables that refer to the objects allocated in memory.

3) Initialization through a Constructor

The concept of object initialization through a constructor is essential to object-oriented programming in Java. Special methods inside a class called constructors are called when an object of that class is created with the new keyword. They initialise the state of objects by entering initial values in their fields or carrying out any required setup procedures. The constructor is automatically invoked upon object instantiation, guaranteeing correct initialization of the object prior to usage.

Here's an example demonstrating object initialization through a constructor:

File: ObjectConstructor.java

```
1.  class Student {
2.      int id;
3.      String name;
4.      // Constructor with parameters
5.      public Student(int id, String name) {
6.          this.id = id;
7.          this.name = name;
8.      }
9.      // Method to display student information
10.     public void displayInformation() {
11.         System.out.println("Student ID: " + id);
12.         System.out.println("Student Name: " + name);
13.     }
```

```
}
public class ObjectConstructor {
    public static void main(String[] args) {
        // Creating objects of Student class with constructor
        Student student1 = new Student(1, "John Doe");
        Student student2 = new Student(2, "Jane Smith");
        // Displaying information of the objects
        student1.displayInformation();
        student2.displayInformation();
    }
}
```

Output:

Student ID: 1
Student Name: John Doe
Student ID: 2
Student Name: Jane Smith

Explanation

In this example, the id and name fields of a Student object are initialised using a constructor defined by the Student class, which accepts two parameters: id and name. Upon creating objects student1 and student2 using this constructor, the fields of each are initialised with the values supplied. This method makes ensuring that objects are created with the correct starting values, which makes it easier to instantiate and use objects later on in the programme.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
class Employee{
    int id;
    String name;
    float salary;
    void insert(int i, String n, float s) {
        id=i;
        name=n;
        salary=s;
    }
    void display(){System.out.println(id+" "+name+" "+salary);}
}
public class TestEmployee {
public static void main(String[] args) {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
```

```
19.    e3.insert(103,"nakul",55000);
20.    e1.display();
21.    e2.display();
22.    e3.display();
23.}
24.}
```

Output:

101 ajeet 45000.0
102 irfan 25000.0
103 nakul 55000.0

Explanation

The employee class in this Java code has three fields: id, name, and salary. It also has two methods: insert, which sets the values for these fields, and display, which prints the values. The main function of the TestEmployee class creates three Employee objects (e1, e2, and e3). To initialise the id, name, and salary fields of each object with precise values, the insert method is called on each of the objects. Then, each object's display method is called, displaying object initialization and information display via method invocation. Each object's id, name, and salary values are printed to the console.

Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

File: TestRectangle1.java

```
1.  class Rectangle{
2.   int length;
3.   int width;
4.   void insert(int l, int w){
5.    length=l;
6.    width=w;
7.   }
8.   void calculateArea(){System.out.println(length*width);}
9.  }
10.class TestRectangle1{
11. public static void main(String args[]){
12.  Rectangle r1=new Rectangle();
13.  Rectangle r2=new Rectangle();
14.  r1.insert(11,5);
15.  r2.insert(3,15);
16.  r1.calculateArea();
17.  r2.calculateArea();
18.}
19.}
```

Output:

55
45

Explanation

This Java code defines a Rectangle class with fields for length and width, along with methods to insert dimensions and calculate the area. In the TestRectangle1 class's main method, two Rectangle objects are instantiated, and their dimensions are set using the insert method.

What are the different ways to create an object in Java?

There are the following ways to create an object in Java.

1. By new keyword

The most common way to create an object in Java is by using the new keyword followed by a constructor.

For example: ClassName obj = new ClassName();. This allocates memory for the object and calls its constructor to initialize it.

2. By newInstance() method

This method is part of the java.lang.Class class and is used to create a new instance of a class dynamically at runtime. It invokes the no-argument constructor of the class.

For example: ClassName obj = (ClassName) Class.forName("ClassName").newInstance();.

3. By clone() method

The clone() method creates a copy of an existing object by performing a shallow copy. It returns a new object that is a duplicate of the original object. For example: ClassName obj2 = (ClassName) obj1.clone();.

4. By deserialization

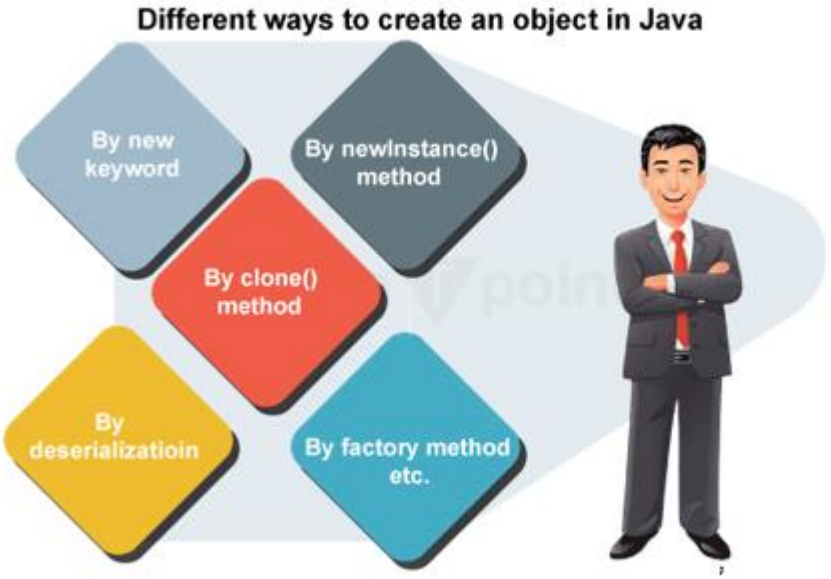
Objects can be created by deserializing them from a stream of bytes. This is achieved using the ObjectInputStream class in Java. The serialized object is read from a file or network, and then the readObject() method is called to recreate the object.

5. By factory method

Factory methods are static methods within a class that return instances of the class. They provide a way to create objects without directly invoking a constructor and can be used to encapsulate object creation logic.

For example: ClassName obj = ClassName.createInstance().

We will learn these ways to create object later.



Anonymous Object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If we have to use an object only once, an anonymous object is a good approach. For example:

```
new Calculation();//anonymous object
```

Calling method through a reference:

```
Calculation c=new Calculation();
c.fact(5);
```

Calling method through an anonymous object

```
new Calculation().fact(5);
```

Let's see the full example of an anonymous object in Java.

```
class Calculation{
 void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}
public static void main(String args[]){
 new Calculation().fact(5);//calling method with anonymous object
}
}
```

Output:

Factorial is 120

Explanation

The factorial of a given integer n can be computed using the Java code that defines the class Calculation and its factorial function. A for loop computes the factorial in the fact method by iterating from 1 to n. After that, the console prints the outcome. The creation of an anonymous Calculation class object and the instantaneous invocation of its fact function with parameter 5 in the main method illustrates how to use anonymous objects in Java for one-time method calls.

Creating Multiple Objects by One Type Only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of Primitive Variables

```
1. int a=10, b=20;
```

Initialization of Reference Variables

```
1. Rectangle r1=new Rectangle(), r2=new Rectangle();//creating two objects
```

Let's see the example:

```
1. //Java Program to illustrate the use of Rectangle class which
2. //has length and width data members
3. class Rectangle{
4.  int length;
5.  int width;
6.  void insert(int l,int w){
7.   length=l;
8.   width=w;
9.  }
10. void calculateArea(){System.out.println(length*width);}
11.}
12.class TestRectangle2{
13. public static void main(String args[]){
14.  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
15.  r1.insert(11,5);
16.  r2.insert(3,15);
17.  r1.calculateArea();
18.  r2.calculateArea();
19.}
20.}
```

Test it Now

Output:

55
45

Explanation

The usage of a Rectangle class with length and width as its data members is demonstrated in this Java programme. There are methods in the class to compute the area of a rectangle and insert dimensions. The main method of the TestRectangle2 class uses a comma-separated list to generate two Rectangle objects (r1 and r2) in one line, demonstrating the ability to instantiate multiple objects of the same type simultaneously. Each object's dimensions are then specified by calling the insert method, and each rectangle's area is then calculated and printed by calling the calculateArea method.

Real World Example: Account

File: TestAccount.java

```
1. //Java Program to demonstrate the working of a banking-system
2. //where we deposit and withdraw amount from our account.
3. //Creating an Account class which has deposit() and withdraw() methods
4. class Account{
5. int acc_no;
6. String name;
7. float amount;
8. //Method to initialize object
9. void insert(int a,String n,float amt){
10. acc_no=a;
11. name=n;
12. amount=amt;
13. }
14. //deposit method
15. void deposit(float amt){
16. amount=amount+amt;
17. System.out.println(amt+" deposited");
18. }
19. //withdraw method
20. void withdraw(float amt){
21. if(amount<amt){
22. System.out.println("Insufficient Balance");
23. }else{
24. amount=amount-amt;
25. System.out.println(amt+" withdrawn");
26. }
27. }
28. //method to check the balance of the account
29. void checkBalance(){System.out.println("Balance is: "+amount);}
30. //method to display the values of an object
31. void display(){System.out.println(acc_no+" "+name+" "+amount);}
32. }
33. //Creating a test class to deposit and withdraw amount
34. class TestAccount{
35. public static void main(String[] args){
36. Account a1=new Account();
```

```
a1.insert(832345,"Ankit",1000);
a1.display();
a1.checkBalance();
a1.deposit(40000);
a1.checkBalance();
a1.withdraw(15000);
a1.checkBalance();
}}
```

Output:

832345 Ankit 1000.0
Balance is: 1000.0
40000.0 deposited
Balance is: 41000.0
15000.0 withdrawn
Balance is: 26000.0

Explanation

With methods for depositing, withdrawing, checking balance, and showing account details, the Account class in this Java programme emulates a simple banking system. Other characteristics include account number, name, and amount. The creation, initialization, and presentation of an Account object a1 with account information take place in the main method of the TestAccount class. Next, the account is used for deposits and withdrawals, and after each transaction, the balance is checked.