Unit-2 : Introducing Classes, Objects and methods

2.1 Class fundamentals

A class is a blueprint for creating an objects. Class encapsulates data and methods to manipulate the data. Class is a combination of data members and member functions.

Syntax :

```
Class ClassName
{
  Data_members;
  member_functions()
  { function_body;
  }
}
```

Note:- Methods and Member functions are same term.

Example :

```
Class person
{
  String Name;
  int age;
  void display_details()
  {
    System.out.println("Name of the person is: "
                       + Name);
    System.out.println("Age of the person is:" +Age);
  }
}public class Rajesh
  public static void main(String args[])
```

```
{
    Person p = new person();
    P.Name = "Rajesh parajuli";
    P.Age = 27;
    P.display_details();
}
}
```

To compile: javac Rajesh.java

To Run: java Rajesh

output: Name of the person is: Rajesh parajuli
Age of the person is: 27

Example 2: To Represent Class in Java program

Java program to find given number is even or odd.

```
import java.util.Scanner;
public class Evenodd
{
    public static void main(String args[])
    {
        Scanner scan = new Scanner(System.in)
        System.out.println("Enter the number:");
        int num = scan.nextInt();
        findEvenodd(num); // Calling function
    }
    public static void findEvenodd(int num);
    {
        if (num%.2 == 0)
            System.out.println("even number");
        else
            System.out.println("odd number");
    }
}
```

2.2. Object Creation

An instance of a class is called an object.
Objects are created using the new keyword.

Syntax:

```
Class_Name object_Name = new Class_Name();
```

Example:

```
Class Animal
{
  String name;
  void display()
  {
    System.out.println("Animal name is: "+name);
  }
}
public Class Demo
{
  public static void main(String[] args)
  {
    //object creation
    Animal dog = new Animal();
    dog.name = "Dog";
    dog.display();
  }
}
```

To compile : javac Demo.java
To Run : java Demo

output: Animal name is : Dog

2.3 methods

Methods define the behavior of a class. methods are member functions that have blocks of code to perform a task.

Syntax:

```
return_type method_Name (parameters)
{
   // block of codes or method body;
}
```

Example:

```
class Calculator
{
  int add (int a, int b)
  {
    return a+b;
  }
}

public class Demo
{
  public static void main (String [] args)
  {
    Calculator calc = new Calculator ();
    int sum = calc.add (5,10);
    System.out.println ("sum is: " +sum);
  }
}
```

output: Sum is: 15

2.4 Command Line Arguments

The java Command line argument is an argument that is passed to the main method at the time of running (Execution) of the java program

Syntax:

```
public static void main(String args[])
{
  // args stores Command-line arguments
}
```

Example:

```
class CommandLine
{
  public static void main(String args[])
  {
    System.out.println("your first argument is:"
                       + args[0]);
  }
}
```

To compile: javac CommandLine.java
To Run : java CommandLine Rajesh

output: your first argument is: Rajesh

Example2: "program that prints all the values"

```
class A
{
  public static void main(String args[])
  {
    for(int i=0; i<args.length; i++)
      System.out.println(args[i]);
  }
}
```

To compile: javac A.java
To Run : java A Rajesh prakash suman kabi
output: Rajesh
prakash
suman
kabi

2.5 Constructors

* A Constructor is a special type of member function that initializes an object when it is created.
* Constructors are automatically called when an instance of class is created. The constructor has the same name as the class name and does not have return type even void.

Syntax:

```
class class_Name
{
    class_Name ( ) // Constructor
    {
        // Constructor body
    }
}
```

Types of Constructor in java

(i) Default Constructor
(ii) Parameterized Constructor

(i) Default Constructor :- The constructor having no arguments and provides the default values to the objects like 0, null etc. depending on the data type of the member variable of the class is called Default Constructor. If there is no any constructor in the class java automatically create the default constructor.

Syntax:

```
class class_Name
{   class_Name ( )
    {
    }
}
```

Program Example

```
Class A
{ int a;
  String name;
  A()
  {
    a=0;
    name=null;
  }
  void show()
  {
    System.out.println("the value of a and name
                        is" +a+" "+ name);
  }
}
Class Demo
{
  public static void main(String args[])
  {
    A obj = new A(); //Automatically execute
                     Constructor
    obj.show(); // showing function execute to
                show the output.
  }
```

Output:

the value of a and name is 0 null

(2) parameterized Constructor:- The Constructor having parameters or arguments is known as parameterized Constructor. this keyword is used to represent the current members of the class.

Syntax:

```
class class_name{
class_name(int A, int B)
{ ----
}
```

Example:

```
Class Student
{
  String name;
  int age;
  Student (String n, int a)
  {
    this.name = n;
    this.age = a;
  }
  void display ( )
  {
    System.out.println("Name is: "+name);
    System.out.println("Age is: "+age);
  }
}
public Class Demo //Driver class for running program
{
  public static void main (String args[])
  {
    Student s = new Student ("Rajesh", 27);
    s.display();
  }
}
```

output: Name is: Rajesh
Age is: 27

2.6 Garbage Collection

Garbage Collection is java's process of automatically freeing up memory by destroying unused objects.

Automatically runtime memory management where java reclaims memory from objects no longer in use.

In other words, Garbage Collection is a way to destroy the unused objects. To do so, we were using free() function in C language and delete () in C++. But in java it is performed automatically.

→ Java memor efficient because gc removes the unreferenced objects from heap memory.

→ It is automatically done by the garbage collector (a part of JVM) so we don't need to make extra efforts.

→ finalize method is to perform clean up processing

(How can an object be unreferenced (destroyed)?

There are different ways to destroy the objects in java:

(i) By nulling the reference
(ii) By assigning a reference to another
(iii) By anonymous object etc.

(i) By nulling the reference

```
Student St = new Student();
St = null;
```

→ The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

(ii) By assigning a reference to another:

```
Student St1= new Student( );
Student st2 = new Student();
st1 = st2; // now st1 is available for
              garbage collecton.
```

(iii) By anonymous Object

```
new Student();
```

Note: Neither finalization nor garbage collection is guaranteed. finalize method is invoked each time before the object is garbage collected. It is used to perform cleanup processing. This method is defined in object class as:

```
protected void finalize()        // Also public but not private
{
}system.out.print(" object is destroyed");
```

Simple program example of garbage collection in java.

```
class GarbageCollection
{ protected void finalize()
  {system.out.println("object is destroyed!");
  }
  public static void main(String args[])
  {
    GarbageCollection g1 = new GarbageCollection();
    GarbageCollection g2 = new GarbageCollection();
     g1=null
     g2=null
     System.gc();
  }}
```

output:

```
object is destroyed
object is destroyed
```

2.7. This Keyword

This keyword can be used within any method (constructor, user-defined method) to represent the current object of the class.

The main purpose is to used this keyword is to avoid naming conflicts of class Data members and constructor parameters.

```
Class person
{
  String name; //instanse variable
  person(String name) // Parameter of constructor
  {
    this.name = name; // this refers to the current
  }                   instance variable/object.
   void display()
   {
    System.out.println("Name is " +this.name);
  }}
  public class Demo
  {
    public static void main(String args[])
    {
      person p = new person("Rajesh");
      P.display();
  }}
```

output: Name is Rajesh

Syntax:

```
Class Exampleofthis
{
    int x;
    Exampleofthis(int x)
    {
      this.x = x; // 'this' refers to the instance variable
    }
}
```

this keyword can be used for different following purposes.

→ this keyword is used to refer to current object.
→ this is always a reference to the object on which method was called.
→ this can be used to call current class constructor.
→ this can be passed as an argument to another method.

Example of reference variable & this.

```
Class A
{
  void display()
  {
    System.out.println(this);
  }
  public static void main(String args[])
  {
    A obj = new A();
    System.out.println(obj);
    obj.display();
  }
}
```

output: print the same referenced ID two times.

Example of calling the class method with "this"

```
class A
{
  void print()
  {
    this.show();
    System.out.println("print method");
  }
  void show()
  {
    System.out.println("show method");
  }
}
class Demo {
  public static void main(String args[])
  {
    A obj = new A();
    obj.print();
  }
}
```

output: show method
print method

2.8 Static fields and methods

The static keyword is used for memory management. we can apply static keywords with java variables, methods, blocks and nested class.

Static fields : Static fields are static variables. Static variable refer the common property of all objects.

Ex: Company name of employees
College name of Students

Static variable will get the memory only once, and retain its value if any object changes the value of the static variables

Examples:

without Static variable

```
Class Counter
{
  int count=0;
  Counter()
  { count++;
    System.out.println(count);
  }
  public static void main(String
  args[])
  { Counter C1= new Counter();
    Counter C2 =new Counter();
    Counter C3 =new Counter();
  }
}
```

output:

1
1
1

with Static variable

```
Class Counter
{
static int count=0;
  Counter()
  { count++;
    System.out.println(count);
  }
public static void main(String
args[])
{ Counter C1 = new Counter();
  Counter C2 = new Counter();
  Counter C3 = new Counter();
}
}
```

output:

1
2
3

Example 2

program showing use of static variable

```
class student
{
  int student_rollno;
  String student_name;
  static String college_name = "amc";
  student(int r, String n)
  {
    student_rollno = r;
    student_name = n;
  }
  void display()
  {
    System.out.println(student_rollno + " " + student_name + " " + college_name);
  }
  public static void main(String args[])
  {
    student S1 = new student(10, "Alina");
    student S2 = new student(11, "sunil");
    student S3 = new student(12, "Laxmi");
    student S4 = new student(13, "surakxya");
    S1.display();
    S2.display();
    S3.display();
    S4.display();
  }
}
```

output:

```
10 Alina amc
11 sunil amc
12 Laxmi amc
13 surakxya amc
```

Static methods

A method is static when you used the static keyword with it. A static method belongs to the class rather than object of class.

→ A static method can be called without creating an object of a class so the main method in java must be static method.

→ Static method can access static data members and can change the value of it.

Example

```
class Calculator
{
  static int add(int a, int b)
  {
    return a+b;
  }
  void display()
  {
    System.out.println("This is a non
                        static method.");
  }
  public class Main{
    public static void main(String args[]){
      int result;
      result = calculator.add(10,20)
      System.out.println(result);
      Calculator calC = new Calculator();
      Calc.display();
    }
  }
```

output: 30
This is a non-static method.

Example 2: Access static data and change the value of it

```
class student
{
  int rollno;
  String name;
  Static String college = "Rastriya";
  Static void change()
  {
    college = "cmc";
  }
  Student(int r, String n)
  {
    rollno = r;
    name = n;
  }
  Void display()
  {
    System.out.println(rollno + " " + name + " " + college);
  }
  public static void main(String args[])
  {
    Student.change();
    Student S1 = new Student(10, "Alina");
    Student S2 = new Student(11, "Laxmi");
    Student S3 = new Student(12, "Sunil");
    S1.display();
    S2.display();
    S3.display();
  }
}
```

output:

```
10 Alina cmc
11 Laxmi cmc
12 Sunil cmc
```

2.10 variable Length Arguments (varargs)

The varargs allows the method to accept zero or multiple arguments. It is used when you are creating a function and you don't know in advance that how many arguments will be passed. Internally, the arguments are treated as an array.

Syntax:

```
return type methodName ( datatype ... varName)
{
    // method body
}
```

→ The ... (three dots) is used to indicate that the method accepts variable arguments.

→ Inside the method, the varargs behave like an array.

Program Example

```
Class varargsExample {
static void display_Numbers (int... numbers)
{
  System.out.println("Numbers of arguments: "
                     + numbers.length);
    for (int num : numbers)
      {System.out.println(num);
    }}
public class main
{public static void main(String[] args)
  {
```

```
varargsExample.displayNumbers(1,2,3,4,5);
varargsExample.displayNumbers(10,20);
    }
}
```

output:

```
Number of arguments: 5
1
2
3
4
5
Number of arguments: 2
10
20
```