Unit 1.2
Programming in Java

Introduction to class and object

- The fundamental idea behind object-oriented programming language is to combine both data and method that operate on data into a single unit called **object.**
- Object is an instance of a class i.e. variable of class type. Object are the basic runtime entities in an object oriented system. Objects contain data and code to manipulate the data. when a program is executed, the objects interact by sending message to one another. Generally, program objects are chosen such that they match closely with real world objects.
- A class is a framework that specifies what data and what functions will be included in objects of that class. It serves as a plan or blueprint from which individual objects are created. A Class is the collection of objects of similar type. Defining class doesn't create an object but class is the description of object's attributes and behaviors.
- For example: mango, apple and orange are members of class Fruit. so if we create a class Fruit then mango, apple, orange can be the object of Fruit

Creating a Class

- Class is created by using keyword 'class'.
- The data, or variables, defined within a class are called instance variables or fields.
- The code is contained within methods.
- Collectively, the methods and variables defined within a class are called members of the class.

Syntax:

```
class <Class Name>
{
        // fields, constructors, methods
}
```

Example:

```
class Student
{
        private int roll;
        public void setRoll(int roll)
        {
                this.roll=roll;
        }
        public int getRoll()
        {
                return(roll);
        }
}
```

Creating Object

Let us take the class Student defined previously.

Student s; // declare reference to object
s = new Student (); // allocate a Student object

The first line declares Student as a reference to an object of type Box. The next line allocates an actual object and assigns a reference to it to "s". These two line can be combined as

Student s = new Student ();

The new operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.

Frequently used classes in Java

1. Math Class

- The class **Math** contains several static methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- Some commonly used static methods are
  a. static double abs(double a) : Returns the absolute value in double of a double value
  b. static float abs(float a) : Returns the absolute value in float of a float value
  c. static int abs(int a) : Returns the absolute value in int of a int value
  d. static long abs(long a) : Returns the absolute value in long of a long value
  e. static double ceil(double a) : Returns the smallest double value that is greater than or equal to the argument and is equal to a mathematical integer.
  f. static double floor(double a) : Returns the largest double value that is less than or equal to the argument and is equal to a mathematical integer.
  g. static int round(float a): floating point value to be rounded to an integer
  h. static double max(double a, double b) : Returns the greater of two double values.
  i. static double min(double a, double b) : Returns the smaller of two double values.
  j. static double pow(double a, double b): Returns the value of the first argument raised to the power of the second argument.
  k. static double sin (double a) :Returns the trigonometric sine of an angle.
  l. static double cos (double a) :Returns the trigonometric cosine of an angle.
  m. static double tan (double a) :Returns the trigonometric tan of an angle.
  etc……..

2. Scanner Class

- The Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings
- It provides many methods to read and parse various primitive values.
- To use the Scanner class, we create an object of the class and use any of the available methods found in the Scanner class.
- To create an object of Scanner class, we usually pass the predefined object "System.in" as parameter, which represents the standard input stream.
- Some commonly used methods are
  a. boolean nextBoolean() : Reads a boolean value from the user
  b. double nextDouble(): read a double value from the user
  c. float nextFloat():read a float value from the user
  d. int nextInt(): read a int value from user
  e. String nextLine(): read a string value from the user
  f. long nextLong(): read a long value from the user.

3. String Class

- The string class is used to manipulate strings whose value will not change. The String class represents sequence of character within a double quote.
- Some commonly used methods are
  a. char charAt(int index) : Returns the char value at the specified index
  b. String concat(String str): Concatenates the specified string to the end of this string.
  c. boolean endsWith(String suffix) :Tests if this string ends with the specified suffix.
  d. boolean startsWith(String prefix): Tests if this string starts with the specified prefix.
  e. boolean startsWith(String prefix, int index): Tests if the substring of this string beginning at the specified index starts with the specified prefix.

f. boolean equals(Object anObject) :Compares this string to the specified object.
g. boolean equalsIgnoreCase(String anotherString): Compares this String to another String, ignoring case considerations.
h. int indexOf(int ch):Returns the index within this string of the first occurrence of the specified character.
i. int indexOf(int ch, int fromIndex):Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
j. int indexOf(String str): Returns the index within this string of the first occurrence of the specified substring.
k. int indexOf(String str, int fromIndex):Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
l. int lastIndexOf(int ch):Returns the index within this string of the last occurrence of the specified character
m. int lastIndexOf(int ch, int fromIndex): Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
n. int lastIndexOf(String str): Returns the index within this string of the last occurrence of the specified substring.
o. int lastIndexOf(String str, int fromIndex): Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
p. int length(): Returns the length of this string.
q. String toUpperCase() :Converts all of the characters in this String to upper case using the rules of the default locale.
r. String toLowerCase(): Converts all of the characters in this String to lower case using the rules of the default locale.
s. char [] toCharArray(): Converts this string to a new character array.
t. String substring(int beginIndex): Returns a new string that is a substring of this string.
u. String substring(int beginIndex, int endIndex): Returns a new string that is a substring of this string.

4. String Builder Class

- Java StringBuilder class is used to create mutable (modifiable) string i.e. the StringBuilder classes are used when there is a necessity to make a lot of modifications to Strings of characters.

Constructor

Public StringBuilder(String str) :Constructs a string builder initialized to the contents of the specified string.

Methods

1. **StringBuilder Appends(String str) :** Appends the specified string to this character sequence.
2. char charAt**(int index) :** Returns the `char` value in this sequence at the specified index.
3. StringBuilder delete(int start, int end) : Removes the characters in a substring of this sequence.
4. **String Builder** deleteCharAt(int index) : Removes the char at the specified position in this sequence.
5. **Int** indexOf(String str): Returns the index within this string of the first occurrence of the specified substring.
6. **int** indexOf(String str, int fromIndex) : Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
7. **StringBuilder** insert(int offset, String str) : Inserts the string into this character sequence.
8. **int** lastIndexOf(String str) : Returns the index within this string of the rightmost occurrence of the specified substring.
9. int length**() :** Returns the length (character count).
10. **StringBuilder** replace(int start, int end, String str) : Replaces the characters in a substring of this sequence with characters in the specified String.
11. **StringBuilder** reverse() : Causes this character sequence to be replaced by the reverse of the sequence.
12. **void** setCharAt(int index, char ch) : The character at the specified index is set to ch.
13. **String** substring(int start, int end) : Returns a new String that contains a subsequence of characters currently contained in this sequence.

5. Character class

- This class provides several methods for determining a character's category (lowercase letter, digit, etc.) and for converting characters from uppercase to lowercase and vice versa
- Some of the static methods provided by Character class for manipulating characters and string are
    a. static boolean isDigit(char ch): returns true if the specified character is digit.

```
Example:
   char c='a';
   char d='1';
   System.out.println(Character.isDigit(c));
   System.out.println(Character.isDigit(d));
   Output:    false
              True
```

    b. static boolean isLetter(char ch): returns true if the specified character is letter.

```
Example:
   char c='a';
   char d='1';
   char e='*';
   System.out.println(Character.isLetter(c));
   System.out.println(Character.isLetter(d));
   System.out.println(Character.isLetter(e));

   Output: true
           false
           false
```

    c. static boolean isLetterOrDigit(char ch): returns true if the specified character is letter or digit.

```
Example:
   char c='a';
   char d='1';
   char e='*';
   System.out.println(Character.isLetterOrDigit(c));
   System.out.println(Character.isLetterOrDigit(d));
   System.out.println(Character.isLetterOrDigit(e));
   Output: true
           true
           false
```

    d. static boolean isLowerCase(char ch): returns true if the specified character is a lower case character

```
Example:
   char c='a';
   System.out.println(Character.isLowerCase(c));
   Output: true
```

    e. static boolean isUpperCase(char ch): returns true if the specified character is a upper case character

```
Example:
   char c='a';
   System.out.println(Character.isUpperCase(c));
   Output: false
```

    f. static char toLowerCase(char ch): converts the specified character to lower case.

```
Example:
   char c='A';
   char u=Character.toLowerCase(c);
   System.out.println(u);
   Output: a
```

    g. static char toUpperCase(char ch): converts the specified character to upper case.

```
Example:
```

```
char c='a';
char u=Character.toUpperCase(c);
System.out.println(u);
Output: A
```

h. static boolean isSpaceChar(char ch): return true if the specified character is space character.

```
Example:
char c1=' ';
System.out.println(Character.isSpaceChar(c1));
Output: true
```

Data Conversion

1. String to int/byte/short/long/boolean
   Example:
   - **a.** int a=Integer.parseInt("100"); // string 100 converted to numeric 100 and assigned to integer variable a
   - **b.** float a=Float.parseFloat("100");
   - **c.** double a=Double.parseDouble("100");
   - **d.** byte a = Byte.parseByte("100");
   - **e.** short =Short.parseShort("100");
   - **f.** long a=Long.parseLong("100");
   - **g.** boolean a= Boolean.parseBoolean("true");

2. Primitive to String
   Example:
   - String a=Integer.toString(1);
   - String a=Double.toString(1.5);
   - String a=Boolean.toString(true);

String Format Method

- The java string format() method returns a formatted string using the given locale, specified format string and arguments

  Syntax:

  static String format(String form, Object… args)

  where

  form– format of the output string
  args– It specifies the number of arguments for the format string. It may be zero or more.

  Example:

```
class Demo3
{
   public static void main(String args[])
   {
      String str = "Bhesh Thapa";

      // Concatenation of two strings
      String str1 = String.format("My Name is %s", str);

      // Output is given up to 8 decimal places
      String str2 = String.format("My answer is %.8f", 47.65734);

      // between "My answer is" and "47.65734000" there are 15 spaces
      String str3 = String.format("My answer is %15.8f", 47.65734);
```

```
        //Output is given up to 2 decimal place
         String str4 = String.format("My answer is %.2f", 47.65734);

         int roll=5;
         String str5=String.format("My name is %s and roll number is %d",str,roll);

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
       System.out.println(str4);
       System.out.println(str5);
    }
}
```

Output:

```
My Name is Bhesh Thapa
My answer is 47.65734000
My answer is     47.65734000
My answer is 47.66
My name is Bhesh Thapa and roll number is 5
```

Programming Examples

1. WAP to enter and display a roll number of a student.

```
class Student
{
        private int roll;
        public void setRoll(int roll)
        {
                this.roll=roll;
        }
        public int getRoll()
        {
                return(roll);
        }
}
class Demo
{
        public static void main(String [] args)
        {
                Student s=new Student();
                s.setRoll(5);
                int r=s.getRoll();
                System.out.println("Roll="+r);
        }
}
```

Output:

```
Roll=5
```

2. WAP to input length and breadth of rectangle and find its area.

```
import java.util.Scanner;
class Rectangle
```

```
{
        private double len,bre;
        public void setData(double len, double bre)
        {
                this.len=len;
                this.bre=bre;
        }
        public double getArea()
        {
                return(len*bre);
        }
}

class Demo
{
        public static void main(String [] args)
        {
                Scanner sc=new Scanner(System.in);
                Rectangle r=new Rectangle();
                System.out.println("Enter length");
                double len=sc.nextDouble();
                System.out.println("Enter breadth");
                double bre=sc.nextDouble();
                r.setData(len,bre);
                double a=r.getArea();
                System.out.println("Area of rectangle="+a);
        }
}
```

Output:

```
Enter length
4
Enter breadth
5
Area of rectangle=20.0
```

3. Create a class called Sum having two data member fn and sn and two function setData(float, float) to initialize these fields and getSum() to return the sum of two number. Write a main program to test your class.

```
import java.util.Scanner;
class Sum
{
        private float fn, sn;
        public void setData(float fn, float sn)
        {
                this.fn=fn;
                this.sn=sn;
        }
        public float getSum()
        {
                float a=fn + sn;
                return(a);
        }
}


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

```
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the first number");
		float m=sc.nextFloat();
		System.out.println("Enter the second number");
		float n=sc.nextFloat();
		Sum a=new Sum();
		a.setData(m,n);
		float sm=a.getSum();
		System.out.println("The sum of two number is "+sm);
	}
}
```

Output:

```
Enter the first number
12
Enter the second number
13
The sum of two number is 25.0
```

4. Create a class called Quadratic having the fields a, b, c and three function members called setData() to initialize these fields and displyRoots() to display the real roots of quadratic equation. Write a main program to test your class.

```
import java.util.Scanner;
class Quadratic
{
	private double a, b, c;
	public void setData(double a, double b, double c)
	{
		this.a=a;
		this.b=b;
		this.c=c;
	}
	public void displayRoots()
	{
		double desc=Math.pow(b,2)-4*a*c;
		double r1= (-b +Math.sqrt(desc)) / ( 2 * a);
		double r2= (-b-Math.sqrt(desc)) / ( 2 * a);
		if(desc<0)
		{
			System.out.println("Roots are imaginary");
		}

		else if(desc==0)
		{
			System.out.println("Roots are equal");
			System.out.println("Root1= "+r1);
			System.out.println("Root2= "+r2);
		}

		else
		{
			System.out.println("Roots are distinct");
			System.out.println("Root1= "+r1);
			System.out.println("Root2= "+r2);
		}
	}
}

public class Demo
```

```
{
	public static void main(String [] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the coefficient a, b an c");
		double a=sc.nextDouble();
		double b=sc.nextDouble();
		double c=sc.nextDouble();
		Quadratic q=new Quadratic();
		q.setData(a,b,c);
		q.displayRoots();
	}
}
```

Output:

```
Enter the coefficient a, b an c
1
-4
4
Roots are equal
Root1= 2.0
Root2= 2.0
```

5. WAP to input a number and count even and odd digits

```
import java.util.Scanner;
public class Demo
{
	public static void main(String [] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter a number");
		int num=sc.nextInt();
		int even=0,odd=0;
		while(num!=0)
		{
			int ext=num%10;
			if(ext%2==0)
			{
				even++;
			}
			else
			{
				odd++;
			}
			num=num/10;
		}
		System.out.println("Total Even Digits="+even);
		System.out.println("Total Odd Digits="+odd);
	}
}
```

Output:

```
Enter a number
1234
Total Even Digits=2
Total Odd Digits=2
```

6. WAP to reverse the digits in a number entered by user. Also check if it is palindrome or not.

```
import java.util.Scanner;
public class Demo
{
	public static void main(String [] args)
	{
```

```
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter a number");
                int num=sc.nextInt();
                int rev=0,tmp;
                tmp=num;
                while(num!=0)
                {
                        int ext=num%10;
                        rev=rev*10+ext;
                        num=num/10;
                }
                System.out.println("Reversed Number="+rev);
                if(tmp==rev)
                {
                        System.out.println("The Entered Number is Palindrome");
                }
                else
                {
                        System.out.println("The Entered Number Not is Palindrome");
                }
        }
}
```

Output:

```
Enter a number
123
Reversed Number=321
The Entered Number Not is Palindrome

C:\Users\Bheeshma\Desktop>java Demo
Enter a number
121
Reversed Number=121
The Entered Number is Palindrome
```

7. WAP to create a class Test that check if the input string is palindrome or not.

```
import java.util.Scanner;
class Test
{
        private String str;
        public void setData(String str)
        {
                this.str=str;
        }
        public boolean isPalindrome()
        {
                String rev="";
                for(int i=str.length()-1;i>=0;i--)
                {
                        char ext=str.charAt(i);
                        rev=rev+ext;
                }

                if(str.equalsIgnoreCase(rev))
                {
                        return true;
                }
                else
                {
                        return false;
                }
        }
```

```
}

class Demo
{
	public static void main(String [] args)
	{
		Scanner sc=new Scanner(System.in);
		Test t=new Test();
		System.out.println("Enter string");
		String str=sc.nextLine();
		t.setData(str);
		boolean res=t.isPalindrome();
		if(res==true)
		{
			System.out.println("The input String is Palindrome");
		}
		else
		{
			System.out.println("The input String is Not Palindrome");
		}
	}
}
```

8. WAP to input line of text and count the number of digits, vowels, consonants, white space, words and other characters.

```
import java.util.Scanner;
public class Demo7
{
  public static void main(String [] args)
  {
	Scanner sc=new Scanner(System.in);
	System.out.println("Enter a sentence");
	String w=sc.nextLine();
	int countvowel=0;
	int countconsonant=0;
	int countspace=0;
	int countdigit=0;
	int countother=0;
	int countword=0;
	for(int i=0;i<w.length();i++)
	{
	    char c=w.charAt(i);
	    if(Character.isDigit(c))
	    {
			countdigit++;
	    }



	    else if(Character.isSpaceChar(c))
	    {
			countspace++;
	    }
	    else if(Character.isLetter(c))
	    {
	       if(c=='a'|| c=='e'|| c=='i'|| c=='o'||c=='u'|| c=='A'|| c=='E'|| c=='I'|| c=='O'||c=='U')
	        {
			countvowel++;
	        }
```

```
                    else
                    {
                            countconsonant++;
                    }
                }
                else
                {
                            countother++;
                }
            }
            countword=countspace+1;
            System.out.println("Total number of vowels in the sentence is "+countvowel);
            System.out.println("Total number of consonants in the sentence is "+countconsonant);
            System.out.println("Total number of spaces in the sentence is "+countspace);
            System.out.println("Total number of other characters in the sentence is "+countother);
            System.out.println("Total number of digits in the sentence is "+countdigit);
            System.out.println("Total number of words in the sentence is "+countword);
    }
}
```

Output

```
Enter a sentence
Ram 1 good * boy
Total number of vowels in the sentence is 4
Total number of consonants in the sentence is 6
Total number of spaces in the sentence is 4
Total number of other characters in the sentence is 1
Total number of digits in the sentence is 1
Total number of words in the sentence is 5
```

9. Define a class to represent a bank account. Include the following members.

Data Members:

i. Name of the account holder
ii. Account Number
iii. Account Type
iv. Balance Amount

Member Functions:

i. To assign initial values from user
ii. To deposit an amount
iii. To withdraw an amount
iv. To display name and balance

Write a main program to test your class

```
import java.util.Scanner;
class BankAccount
{
        private String name,acc_type;
        private int acc_no;
        private double balance;
        public void initial(String name, int acc_no, String acc_type, double balance)
        {
                this.name=name;
                this.acc_no=acc_no;
                this.acc_type=acc_type;
                this.balance=balance;
        }
        public void deposit(double c)
        {
```

```
                balance=balance+c;

        }
        public void withdraw(double c)
        {
                if(balance<c)
                {
                        System.out.println("Withdraw amount larger than balance");
                }
                else
                {
                        balance=balance-c;
                }
        }
        public void display()
        {
                System.out.println("Name= "+name);
                System.out.println("Balance amount= "+balance);
        }
}
public class DemoBankAccount
{
        public static void main(String [] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the name of accont holder");
                String n=sc.nextLine();
                System.out.println("Enter the account number");
                int an=sc.nextInt();
                System.out.println("Enter the account type");
                String at=sc.nextLine();
                sc.nextLine(); //For flushing
                System.out.println("Enter the initial balance");
                double d=sc.nextDouble();
                BankAccount b=new BankAccount();
                b.initial(n,an,at,d);
                b.deposit(10000);//depositing an amount of 10000
                b.withdraw(5000); // withdraw an amount of 5000
                b.display();
        }
}
```

Output:

```
Enter the name of accont holder
Bhesh Thapa
Enter the account number
1
Enter the account type
Enter the initial balance
100
Name= Bhesh Thapa
Balance amount= 5100.0
```

Access Modifier/Visibility Modifiers

- Access level modifiers determine whether other classes can access/use a particular field or invoke a particular method.
- There are two levels of access control:
  - At the top level i.e. Class Level: **public, or package-private** (no explicit modifier i.e. Default one).
  - At the member level: **public, private, protected, or package-private** (no explicit modifier).

- A class may be declared with the modifier public, in which case that class is visible to all classes everywhere. If a class has no modifier (the default, also known as **package-private**), it is visible only within its own package (packages are named groups of related classes — you will learn about them in a later lesson.)
- At the member level, you can also use the public modifier or no modifier (package-private) just as with top-level classes, and with the same meaning. For members, there are two additional access modifiers: private and protected. The private modifier specifies that the member can only be accessed in its own class. The protected modifier specifies that the member can only be accessed within its own package (as with package-private) and, in addition, by a subclass of its class in another package.

The following table shows the access to **members permitted by each modifier**.

Visibility	Public	Protected	*Default*	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, *through inheritance*	No	No
From any non-subclass class outside the package	Yes	No	No	No

Package

- **A package** is a collection of related classes and interfaces that provides access protection and namespace management.
- Package organizes classes and interfaces in logical manner.
- A unique name had to be used for each class to avoid name collisions.
- So, we need some way to be assured that the name you choose for a class will be reasonably unique and not collide with class names chosen by other programmers.
- For this, Java provides a mechanism for partitioning the class name space into more manageable chunks. This mechanism is the package.
- Java package can be categorized in two ways.
  1. Java API package.
  2. User defined package.

Defining a Package

- To create a package is quite easy: simply include a package command as the first statement in a Java source file. Any classes declared within that file will belong to the specified package. If you omit the package statement, the class names are put into the default package, which has no name. This is the general form of the package statement:

  package pkg;

  Here, pkg is the name of the package.
- Java uses **file system directories/folders** to store packages.
- We also can create a hierarchy of packages. The general form of a multileveled package statement is shown here:

  package [pkg1].[pkg2].[pkg3];

- For example, a package declared as package java.util; is stored in java\util in a Windows environment.

Java API package

- The java API package provides a wide range of useful classes ready for use in our application.
- To use the package, we use the import statement. Some frequently used java API's are
  - java.io
  - java.util
  - java.net

The general syntax to use the package is

```
import PackageName.ClassName // for specific class in package
                or
import PackageName.*// for entire classes in package
```

Example:

```
import java.util.Scanner;
import java.util.Date;
```

or, the above two statement can be written with single statement as below.

```
import java.util.*;
```

User defined package

- We can create our custom package by placing all the classes and interfaces in it.
- For this, we have to put a package statement at the top of the source file in which the class or interface is defined.
- The general syntax for defining package is given below.

```
package <package name>                  package <package name>
public class <class name>      or       public interface <interface name>
{                                       {

}                                       }
```

Example:

Following example illustrate how to create user defined package.

Step1: Create java source file(s). Define user define package 'Operation' as first statement.

```
package Operation;
public class ArthOprn
{

        public double add(double a,double b)
        {
                return (a+b);
        }
        public double sub(double a, double b)
        {
                return (a-b);
        }
}
```

Step2: Compile it using

```
javac –d . ArthOprn.java // there must be space before and after .
```

The compilation will create a folder 'Operation' on the working directory which consist of ArthOprn.class file.

Step3: Create another java source file which make use of the above package. For using the above package, we must use 'import' statement as shown below.

```
import Operation.ArthOprn; // i.e. to access the classes of Operation_ArthOprn, we must
                                                                                    import it
public class MathOprn
{
        public static void main(String [] args)
        {
                ArthOprn a=new ArthOprn();
                System.out.println(a.add(4,5));
                System.out.println(a.sub(4,5));
        }
}
```

Step4: compile the above source file using

javac MathOprn.java

It will create MathOprn.class in the working directory.

Step5: Execute the above class using

java MathOprn

Step6: It will output

```
9.0
-1.0
```

Overloading Methods

- Method overloading is the process of creating methods that have same name, but different parameters list and different definition.
- So we obtain polymorphism using the concept of method overloading.
- Java supports overloading methods and uses method signatures to distinguish between methods.
- When we call a method in an object, java matches up the method name first then the number of parameters and type of parameters to decide which one of the definition to execute. This process is known as polymorphism.

WAP to find the area of circle and rectangle using method overloading.

```
import java.util.Scanner;
class ComputeArea
{

        public double area(double r)
        {
                return (3.1416 * r * r);
        }
        public double area(double l, double b)
        {
                return (l*b);
        }
}

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

```
                ComputeArea a=new ComputeArea();
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the radius");
                double radius=sc.nextDouble();
                System.out.println("The area of circle is "+a.area(radius));
                System.out.println("Enter the length");
                double length=sc.nextDouble();
                System.out.println("Enter the breadth");
                double breadth=sc.nextDouble();
                System.out.println("The area of rectangle is " + a.area(length,breadth));
        }
}
```

Output:

```
C:\Users\Bheeshma\Desktop>java Demo
Enter the radius
1
The area of circle is 3.1416
Enter the length
4
Enter the breadth
6
The area of rectangle is 24.0
```

Constructor

- A constructor is a special member function having the same name as that of the class which is used to automatically initialize the objects of the class type with legal initial values.
- The constructor is invoked automatically whenever an object of its associated class is created.
- It is called constructor because it constructs the values of data members of the class.

Properties:

- They are also used to allocate memory for a class object.
- They execute automatically when an object of a class is created.
- Constructor's name is same as that of class name.
- They should be declared in the "public" section. It is sometime declared in private section when we need to restrict creating object of that class. This is called private constructor.
- They do not have return types, not even void and therefore, and they cannot return any values.

A constructor is declared and defined as below

```
class Demo
{
        field declaration;
        --------------
        --------------
        public Demo() // Default Constructor
        {

        }
}
```

Types of constructor

i. Default Constructor

- A constructor that does not take any parameter is called default constructor.

- This constructor is always called by compiler if no user-defined constructor is provided.
- The following Class definition shows a default constructor.

```
class Demo
{
        public Demo() // Default Constructor
        {

        }
}
```

- This constructor is also called as implicit constructor because if we do not provide any constructor with a class, the compiler provided one would be the default constructor.
- And it does not do anything other than allocating memory for the class object.

ii. Parameterized Constructor

- The constructors that can take arguments or parameters are called parameterized constructor.
- In this, we pass the initial value as arguments to the constructor function when the object is declared.
- The argument to be passed must match the signature i.e. no or parameters and types of constructor defined.
- The following Class definition shows a parameterized constructor.

```
class Demo
{
        public Demo(int x, int y)  //Parameterized Constructor
        {

        }
}
```

WAP to find the area of rectangle using the concept of constructor

```
class Rectangle
{
        private double len,bre;

        public Rectangle()
        {
                len=0;
                bre=0;
        }
        public Rectangle(double len, double bre)
        {
                this.len=len;
                this.bre=bre;
        }
        public double getArea()
        {
                return(len*bre);
        }
}

public class Demo
{
        public static void main(String [] args)
        {
                Rectangle r=new Rectangle(4,5);
                double res=r.getArea();
```

```
                System.out.println("Area of Rectangle="+res);
        }
    }
```

Output:

```
C:\Users\Bheeshma\Desktop>java Demo
Area of Rectangle=20.0
```

Constructor Overloading

- A constructor is said to be overloaded if it has both default and parameterized constructor.
- When more than one constructor functions are defined in the same class then we say the constructor is overloaded.
- All the constructors have the same name as the corresponding class, and they differ in terms of number of arguments, data types of argument or both.
- This makes the creation of object flexible.

```
import java.util.Scanner;
class OverloadingConstructor
{
    public OverloadingConstructor(double r)
    {
        double ara=3.1416 * r * r;
        System.out.println("The area of circle is"+ ara);
    }
    public OverloadingConstructor(double l, double b)
    {
        double ara=l * b;
        System.out.println("The area of rectangel is "+ara);
    }
}
public class DemoOverloadingConstructor
{
    public static void main(String [] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the radius");
        double radius=sc.nextDouble();
        new OverloadingConstructor(radius);
        System.out.println("Enter the legth");
        double length=sc.nextDouble();
        System.out.println("Enter the breadth");
        double breadth=sc.nextDouble();
        new OverloadingConstructor(length,breadth);
    }
}
```

Output:

```
Enter the radius
1
The area of circle is3.1416
Enter the legth
6
Enter the breadth
5
The area of rectangel is 30.0
```

Private Constructor

- A private constructor is a special instance constructor.

- It is generally used in classes that contain static members only.
- If a class has one or more private constructors and no public constructors, **other classes cannot create instances of this class.**

The following program demonstrates the concept of private constructor

```
class Demo
{
	private Demo()
	{

	}
	public  static void display()
	{
		System.out.println("Hello");
	}

}
class Test
{
	public static void main(String [] args)
	{
		//Demo d=new Demo(); this statement is illegal now as the constructor is private
		Demo.display(); //Static member are directly accessed via class name
	}
}
```

Constructor Chaining

- Constructor chaining is the process of calling one constructor from another constructor with respect to current object.
- **Within same class**: It can be done using **this** keyword for constructors in same class
- This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable

Rules of constructor chaining :

- The this expression should always be the first line of the constructor.
- There should be at-least be one constructor without the this keyword
- Constructor chaining can be achieved in any order.

The following program demonstrates the concept of constructor chaining'

```
	public class Check
	{
		int a;
		public Check()
		{
			this(2);//will invoke parameterized constructorz
			a=0;
			System.out.println("Displayed from Default Constructor, Value of a="+a);
		}
		public Check(int x)
		{
			a=x;
			System.out.println("Displayed from Parameterized Constructor, Value of a="+a);
		}
	}
	public class Demo
	{
		public static void main(String [] args)
```

```
        {
                new Check(); //will invoke default constructor
        }
    }
```

Output:

```
Displayed from Parameterized Constructor, Value of a=2
Displayed from Default Constructor, Value of a=0
```

Nested and Inner Classes

A class within another class is known as nested classes. Nested classes are used because of the following reasons.

1. Logical grouping of classes.
2. Increased encapsulation
3. More readable and maintainable codes

There are two types of nested class.

1. Static nested class
2. Non-static nested class

1. Static nested class

- If the nested class is static, then they are not associated with the instance of the outer class.
- Hence we can create the instance of the static nested class in the classes other than outer and inner without creating the instance of outer class.
- The static inner class can contain both static and non-static members.

Example: Static inner class with non-static member

```
class Outer
{
        static int x=100;
        static class Inner
        {
                public void display()
                {
                        System.out.println(x);
                }
        }
}
class NestedClass
{
        public static void main(String args [])
        {
                Outer.Inner in=new Outer.Inner(); // instance of static nested class
                in.display();
        }
}
```

Example: Static inner class with static member

```
class Outer
{
        static int x=100;
        static class Inner
        {
                public static void display()
                {
                        System.out.println(x);
                }
        }
```

```
}
class NestedClass
{
        public static void main(String args [])
        {
        -       Outer.Inner.display();
        }
}
```

2. Non –static Nested class

- An inner class that is not declared with static specifier is called non-static nested class.
- Non static nested classes can be defined both inside and outside the methods.

Example: Defining outside the method

```
class Outer
{
        int  x=20;
        void test()
        {
                Inner in = new Inner();
                in.show();
        }
        class Inner
        {
                void show()
                {
                        System.out.println(x);
                }
        }
}


class InnerClass
{
        public static void main(String [] args)
        {
                Outer ou=new Outer();
                ou.test();
        }
}
```

Output:

```
20
```

Example: Defining inside the method

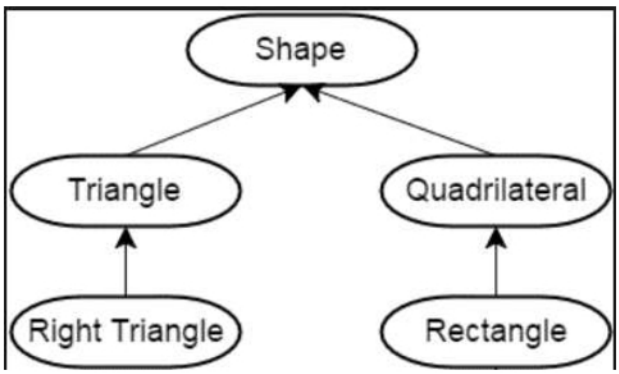
```
class Outer
{
        int  x=20;
        void test()
        {
                class Inner
                {
                        void show()
                        {
                                System.out.println(x);
                        }
```

```
                }
                Inner in = new Inner();
                in.show();
        }
}
class Demo
{
        public static void main(String [] args)
        {
                Outer ou=new Outer();
                ou.test();
        }
}
```

Output : 20

Inheritance

- Inheritance is the mechanism of deriving a new class from existing class.
- The existing class is called super class or base class or parent class and the child class is called subclass or derived class or extended class.
- The subclass inherits some of the properties from the base class and can add its own property as well.
- All the public and protected properties such as fields, methods etc. can be inherited by the derived class however private members can't be inherited.
- In java we can achieve inheritance using keyword '**extends**'.



- The main advantage of inheritance is the concept of **code reusability.** A code is said to be reusable if we can reuse the properties of superclass in other class by using the concept of inheritance. A programmer can use a class created by another person or company without any modification at all or with small modification such as deriving other classes from it to fit his situation. So reusing existing codes saves time and money and increase code reliability.

Inheritance Types

1. Single inheritance
2. Multiple inheritance
3. Hierarchical inheritance
4. Multilevel inheritance
5. Hybrid inheritance

1. Single inheritance

In single inheritance, a class is derived from only one existing class. i.e. only one super class.

The general form is given below

```
class A
{
```

```
        //member of A
}
class B extends A
{
        //own member of B
}
```

2. Multiple inheritance
   In multiple inheritance, a class is derived from more than one existing classes. **Java doesn't support multiple inheritance due to ambiguity in parent class**. When compilers of programming languages that support this type of multiple inheritance encounter super classes that contain methods with the same name, they sometimes cannot determine which member or method to access or invoke. We use alternative approach '**interface**' to implement the concept of multiple inheritance in java without any ambiguity.

3. Hierarchical inheritance
   In hierarchical inheritance, two or more classes inherits the properties of one existing class. The general form is

```
class A
{
        //member of A
}
class B  extends A
{
        //own member of B
}
class C extends A
{
        //own member of  C
}
```

4. Multilevel inheritance
   In multilevel inheritance, a class is derived from another subclass. The general form is

```
class A
{
        //member of A
}
class B extends A
{
        //own member of B
}
class C extend B
{
        //own member of C
}
```

5. Hybrid inheritance
   In hybrid inheritance, it can be the combination of single and multiple inheritance or any other combination. One form can be as below

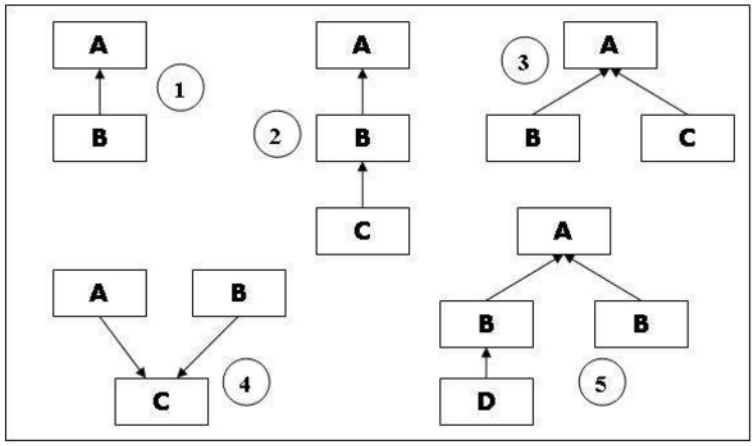
```
Class X
{
        //member of X
}
Class A extends X
```

```
{
        // own member of A
}
Class B extends A
{
        //own member of C
}
Class C extends A
{
        //own member of C
}
Class D extends A
{
        //own member of D
}
```

Above example consists both single and hierarchical inheritance hence called hybrid inheritance.

The below figure illustrates inheritance.

- (1) Simple / Single
- (2) Multilevel
- (3) Hierarchical
- (4) Multiple
- (5) Hybrid



Programming example:

1. A class **Room** consists of two fields **length** and **breadth** and method int **area ()** to find the area of room**.** A new class **BedRoom** is derived from class **Room** and consist of additional field **height** and two method **setData(int,int,int)** to set the value for three fields and **int volume()** to find the volume. Now write the java program to input the length, breadth and height and find the area and volume. What form of inheritance will the classes hold in this case?

[Single inheritance]

```
import java.util.Scanner;
class Room
{
        protected int length;
        protected int breadth;
        public int getArea()
```

```
        {
                return(length*breadth);
        }
}

class BedRoom extends Room
{
        private int height;
        public void setData(int x,int y, int z)
        {
                length=x;
                breadth=y;
                height=z;
        }
        public int getVolume()
        {
                return (length *breadth *height);
        }
}

public class InheritanceDemo1
{
        public static void main(String [] args)
        {
                BedRoom br=new BedRoom();
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the length");
                int l=sc.nextInt();
                System.out.println("Enter the breadth");
                int b=sc.nextInt();
                System.out.println("Enter the height");
                int h=sc.nextInt();
                br.setData(l,b,h);
                System.out.println("The area of Bedroom is"+br.getArea());
                System.out.println("The volume of Bedroom is"+br.getVolume());
        }
}
```

Output:

```
Enter the length
2
Enter the breadth
3
Enter the height
4
The area of Bedroom is6
The volume of Bedroom is24
```

2. A company needs to keep record of its following **Employees**:

   i) Manager ii) Supervisor

   The record requires name and salary of both employees. In addition, it also requires section_name (i.e. name of section, example Accounts, Marketing, etc.) for the Manager and group_id (Group identification number, e.g. 205, 112, etc.) for the Supervisor. Design classes for the above requirement. Each of the classes should have a function called set() to assign data to the fields and a function called get() to return the value of the fields. Write a main program to test your classes. What form of inheritance will the classes hold in this case?

   [Hierarchical inheritance]

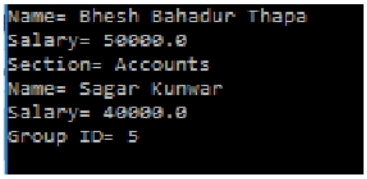
```
class Employee
```

```
{
	private String name;
	private double salary;
	public void setName(String name)
	{
		this.name=name;
	}
	public void setSalary(double salary)
	{
		this.salary=salary;
	}
	public String getName()
	{
		return name;
	}


	public double getSalary()
	{
		return salary;
	}
}
class Manager extends Employee
{
	private String section_name;
	public void setSection_name(String section_name)
	{
		this.section_name=section_name;
	}
	public String getSection_name()
	{
		return section_name;
	}
}

class Supervisor extends Employee
{
	private int group_id;
	public void setGroup_id(int group_id)
	{
		this.group_id=group_id;
	}
	public int getGroup_id()
	{
		return group_id;
	}
}
public class Demo
```

```
{
    public static void main(String[] args)
    {
        Manager m=new Manager();
        m.setName("Bhesh Bahadur Thapa");
        m.setSalary(50000);
        m.setSection_name("Accounts");
        System.out.println("Name= "+m.getName());
        System.out.println("Salary= "+m.getSalary());
        System.out.println("Section= "+m.getSection_name());

        Supervisor s=new Supervisor();
        s.setName("Sagar Kunwar");
        s.setSalary(40000);
        s.setGroup_id(5);
        System.out.println("Name= "+s.getName());
        System.out.println("Salary= "+s.getSalary());
        System.out.println("Group ID= "+s.getGroup_id());
    }
}
```

In this case, the classes hold **hierarchical inheritance.**

Output:



Interface

- An interface is a named collection of methods declaration without implementations i.e. abstract methods.
- Interface is similar to class, but they lack instant variable and their method are declared without body.
- Interface define what a class must do but not how it does.
- A class that implements the interfaces must implement all the methods define in the interface
- One significant difference between classes and interfaces is that classes can have fields whereas interfaces cannot. In addition, you can instantiate a class to create an object, which you cannot do with interfaces. Class can define constructor but interface can't define.

Defining Interface

```
<access> interface <interface name>
{
    //constant declaration
    //return_type method_name(parameter list);
}
```

In above, access is **public or package private, no other access is allowed**. Variable are implicitly final and static so that they can't be changed by implementing class. Similarly, **constant and methods access is also either public or package private(default), no other access is allowed**. **Also variables must be initialized.**

Example:

```
interface  Area
{
    final static double pi=3.1416//  interface variable  or simply we can declare as, double pi=3.1416
```

```
        double findArea(double radius);  // interface method or abstract method
}
```

Class vs Interface

CLASS	INTERFACE
Supports only multilevel and hierarchical inheritances but not multiple inheritance	Supports all types of inheritance: multilevel, hierarchical and multiple
"extends" keyword should be used to inherit	"implements" keyword should be used to inherit
Should contain only concrete methods (methods with body)	Should contain only abstract methods (methods without body)
The methods can be of any access specifier (all the four types)	The access specifier must be public only
Methods can be final and static	Methods should not be final and static
Variables can be private	Variables should be public only
Can have constructors	Cannot have constructors
Can have main() method	Cannot have main() method as main() is a concrete method

Implementing Interface

After defining an interface, then one or more class can implement that interface. To implement an interface, include the **implement** clause in class definition and then define all the methods declared in interface. We can implement multiple interface by same class at same time.

Example:

```
import java.util.Scanner;
interface  Area
{
        final double pi=3.1416;//  interface variable  or simply we can declare as,
                                                                        //double pi=3.1416
        double findArea(double radius);  // interface method also called as abstract method
}

class InterfaceDemo implements Area
{
        public double findArea(double r)// must define public
        {
                return (pi*r*r);
        }
}

public class Demo
{
        public static void main(String [] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter radius");
                double r=sc.nextDouble();
                InterfaceDemo id=new InterfaceDemo();
                double rs=id.findArea(r);
                System.out.println("The are of circle is" +rs);

                //another way using interface object
                Area a=new InterfaceDemo();   // a is a reference variable of type Area
                System.out.println("Enter radius");
```

```
                double rad=sc.nextDouble();
                double rs1=a.findArea(rad);
                System.out.println("The are of circle is" +rs1);
        }

}
```

Output:

```
Enter radius
1
The are of circle is3.1416
Enter radius
2
The are of circle is12.5664
```

Interface Extension/use of interface to achieve multiple inheritance

An interface can extend other interface just like a class can extend another class however a class can extend only one other class but an interface can extend any number of interfaces. The interface declaration includes a comma-separated list of all the interfaces that it extends.

Java doesn't provide multiple inheritance so interface provides an alternative for it. In java, a class can inherit i.e. extends only one class **but interface can inherit i.e. extends more than one interface and also class can implement more than one interface.** So if the situation come where we need to gather the properties of two or more than two kind of objects then we can use interface two achieve the **multiple inheritance.**

Example to show multiple inheritance using interface extension

```
interface A
{
        int a=5;
        void displayValueOfA(); //public or not no matter
}
interface B
{
        int b=7;
        void displayValueofB();
}
interface C extends A, B
{
        int c=9;
        void displayValueOfC();
}

class MyClass implements C  // since Myclass implement C and C extends both A and B so
                                        MyClass must define all the methods declared in A B an C
{
        public void displayValueOfA()
        {
                System.out.println("The value of a is"+a);
        }
        public void displayValueofB()
        {
                System.out.println("The value of b is"+b);
        }
        public void displayValueOfC()
        {
                System.out.println("The value of c is"+c);
```

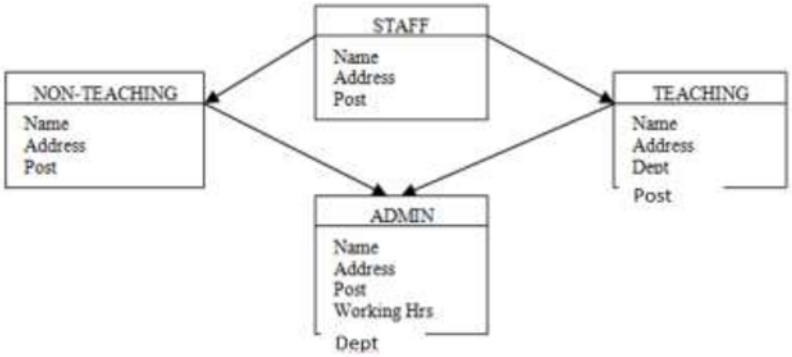
```
        }
    }
    public class Demo
    {
        public static void main(String [] args)
        {
            MyClass m=new MyClass();
            m.displayValueOfA();
            m.displayValueofB();
            m.displayValueOfC();
            /*
            //suppose we want to access member of B
            B obj=new MyClass();
            obj.displayValueofB();  //other member are not accessible

            //suppose we want to access member of A
            A obj1=new MyClass();
            obj1.displayValueofA();  //other member are not accessible

            //suppose we want to access member of C
            A obj2=new MyClass();
            Obj2.displayValueofA();  //other member are also accessible since A extend B and C
            Obj2.displayValueofB();
            Obj2.displayValueofC();
            */

        }

    }
```

Output:

```
The value of a is5
The value of b is7
The value of c is9
```

Question:

The following figure shows minimum information required for each class/interface. Specify all the classes/interfaces and define functions to assign and display individual information. Write a main program to test ADMIN, TEACHING and NON_TEACHING. What form of inheritance will the classes/interfaces hold in this case?




```
interface STAFF
{
        public void setNAP(String name, String add, String post);
        public void showNAP();
}
```

```
interface NON_TEACHING extends STAFF
{

}

interface TEACHING extends STAFF
{
	public void setDept(String dept);
	public void showDept();
}
class ADMIN implements NON_TEACHING, TEACHING
{
	float wh;
	String name, add, post,dept;
	public void setWh(float wh)
	{
		this.wh=wh;
	}
	public void showWh()
	{
		System.out.println("Working Hour="+wh);
	}
	public void setNAP(String name, String add, String post)
	{
		this.name=name;
		this.add=add;
		this.post=post;
	}
	public void showNAP()
	{
		System.out.println("Name="+name);
		System.out.println("Address="+add);
		System.out.println("Post="+post);
	}
	public void setDept(String dept)
	{
		this.dept=dept;
	}
	public void showDept()
	{
		System.out.println("Department=dept");
	}
}




public class Demo
{
	public static void main(String [] args)
	{
		//Set and Get Admin
		ADMIN a=new ADMIN();
		a.setNAP("Bhesh", "Hemja", "Co-ordinator");
```

```
        a.setWh(7);
        a.showNAP();
        a.showWh();
        a.setDept("IT");
        a.showDept();

        //set and get Non-teaching staff
        NON_TEACHING nt=new ADMIN(); //nt is reference variable of
        NON_TEACHING interface type
        nt.setNAP("Seeta", "Pokhara", "Accountant");
        nt.showNAP();

        //set and get Teaching
        TEACHING t=new ADMIN();
        t.setNAP("Sunil", "Lamachaur","Lecturer");
        t.setDept("IT");
        t.showNAP();
        t.showDept();
    }
}
```

Output:

```
Name=Bhesh
Address=Hemja
Post=Co-ordinator
Working Hour=7.0
Department=IT
Name=Seeta
Address=Pokhara
Post=Accountant
Name=Sunil
Address=Lamachaur
Post=Lecturer
Department=IT
```

It is the combination of hierarchical, multilevel and multiple inheritance, so classes hold the property of hybrid inheritance.

Overriding Methods

- The process of redefining the inherited methods of the super class in the derived class is called method overriding.
- The method in the derived class should have same signature and same return type as that of base class in order to override it.
- Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime, it is also called as Run Time Polymorphism.

Example:

```
class BaseClass
{

    void display()
    {
        System.out.println("This is BaseClass...");
    }
}
class DerivedClass extends BaseClass
 {
    void display()
```

```
        {
                System.out.println("This is Derived Class...");
        }
}
public class OverriddenMethods
{
        public static void main(String[] args)
        {
                DerivedClass dc = new DerivedClass();
                dc.display();
        }
}
```

Output: This is Derived Class….

Overloading vs Overriding

Method overloading	Method overriding
1. The process of defining functions having the same name with different parameter list is called as **overloading method**.	1. When methods of the subclass having same name as that of superclass, overrides the methods of the superclass then it is called as **overriding methods**.
2. It is a relationship between methods in the same class.	2. It is a relationship between subclass method and a superclass method.
3. It is static binding –at the compile time.	3. It is dynamic binding—at the runtime.
4. It is the concept of **compile time polymorphism or early binding.**	4. It is the concept of **run-time polymorphism or late binding.**
5. It may or may not be observed during inheritance.	5. It is observed during inheritance.
6. Return types do not affect overloading.	6. Return types, method names and signatures of both overridden methods must be identical.

The Static Member/ Static Modifiers

- Static members are those members which belongs to the whole class rather than the object created from the class i.e. these members are common to all the objects and accessed without using the particular object.
- Static variable and static methods are referred to as static member.
- Static variable is used when we want to have a common variable to all instances of class.
- Also we can declare static block which gets executed exactly once, when the class is first loaded.
- The most common example of a static member is main(). main() is declared as static because it must be called before any object exists.

Properties of static methods

1. Are invoked using the class name rather than object
2. Can only access static data however non static member can access static data also.
3. Can only call other static methods.
4. Can't refer to 'this' or super

WAP to demonstrate the use of static members.

```
class MathOperation
{
```

```
        public static double mul(double x, double y)
        {
                return(x*y);
        }
        public static double divide(double x, double y)
        {
                return(x/y);
        }
}

public class MathApplication
{
        public static void main(String args[])
        {
                double a= MathOperation.mul(2.5,3.5);
                double b = MathOperation.divide(4.0,2.0);
                System.out.println("a= " +a);
                System.out.println("b= " +b);
                System.out.println("The value of pi is"+MathOperation.pi);
        }
}
```

this Keyword

This can be used inside any method to refer to the current object.

Example:

```
class Box
{
        private double width,height,depth;
        public Box(double w, double h, double d)
        {
                width = w;
                height = h;
                depth = d;
        }
}
```

Using the **this** keyword, we can distinctly identify formal parameters variable from class instance variable having same name. so when a local variable has the same name as instance variable, the local variable hides the instance variable. So we can use this to overcome the **instance variable hiding.**

Example:

```
class Box
        {
                private double width, height, depth;
                public Box(double width, double height, double depth)
                {
                        this.width = width;
                        this.height = height;
                        this.depth = depth;
                }
        }
```

So Inside Box( ), "**this**" will always refer to the invoking object.

Final Keyword/Modifiers

- Final is a non-access modifier applicable **only to a variable, a method or a class**.

- **Non Access Modifiers** are the keywords to notify JVM about a class's behaviour, methods or variables, etc. That helps introduce additional functionalities, such as the final keyword used to indicate that the variable cannot be initialized twice.
- Following are different contexts where final is used.
  a. Final Variable: To create constant variable
  b. Final Method: To prevent method overriding
  c. Final Class: To prevent inheritance

Final Variable: Using final to create the equivalent of the named constants.

- 'final ' keyword in java is similar as const keyword in C/C++.
- A variable declared as a final is prevented to modify its content.

Example:

```
class Fin
{
        public static void main(String [] args)
        {
                final double pi=3.1416;
                pi++;// this statement is wrong because we can't modify pi as it is declared final.
        }
}
```

Final Method: Using final keyword to prevent method Overriding

- In object-oriented terms, overriding means to override the functionality of an existing method. i.e. redefining the inherited method of super class in the derived class.
- The benefit of overriding is the ability to define a behaviour that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement.
- The method declared as 'final' can't be overridden.

Example 1:

```
class Animal
{
        public void move()
        {
                System.out.println("Animals can move");
        }
}

class Dog extends Animal
{
        public void move()         // method overridding
        {
                System.out.println("Dogs can walk and run");
        }
}



public class TestDog
{
        public static void main(String args[])
        {
                Animal a = new Animal(); // Animal reference and object
                Animal b = new Dog(); // Animal reference but Dog object
                a.move();// runs the method in Animal class
```

```
            b.move();//Runs the method in Dog class
        }
    }
```

Output:

```
Animal can move
Dogs can walk and run.
```

Example2:

```
class A
{
    final void abc()
    {
        System.out.println("This is final methods");
    }
}
class B extends A
{
    void  abc()  //  illegal because it can't be overridded as it is declared final in
    {

    }
}
```

Final Class: Using final to prevent inheritance.

- A class that can't be inherited is called final class.
- This is achieved in java using the keyword 'final'.

Example:

```
final class A
{

}
class B extends A  // illegal because final class can't be inherited
{

}
```

Array

- An array is a set of related items of same data type that share common name.
- Arrays of any type like int, float, String etc. can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index.
- Types
    1. One-dimensional Array
    2. Two dimensional Array

One dimensional Array

- One dimensional array can store a list of items of same time.
- The general form of 1D array declaration is

```
type [] variablename;
```

Or

type variablename[];

Example:

int student[]; //declares an array variable with name "student" of type integer.

- A **new** operator is used to allocate memory for an array as below

```
int []student;
student=new int[10];
```

This example allocates a 10-element array of integers and links them to student.

The above two statement also can be combined to one statement as

```
int [] student=new int[10];
```

Multi-dimensional Array

- An array with two or more dimensions is called a multi-dimensional array.
- The general form for 2D array declaration is

```
type variablename[][];
        Or
type [][] variablename;
```

Example:

int student[][]; //declares 2D array variable with name "student" of type integer.

- A **new** operator is used to allocate memory for an array as below

```
int [][] student;
student=new int[5][4];
```

This example allocates a 2D array of integers with 5 rows and 4 columns and links them to student.

The above two statement also can be combined to one statement as

```
int [][]student=new int[5][4];
```

Programming Example

1. WAP to input 10 numbers and display them in reverse order.

```
import java.util.*;
public class Demo
{
    public static void main(String [] args)
    {
        Scanner sc=new Scanner(System.in);
        int []data=new int[10];
        System.out.println("Enter 10 numbers");
        for(int i=0;i<10;i++)
        {
            data[i]=sc.nextInt();
        }
        System.out.println("The elements in the array are");
        for(int i=9;i>=0;i--)
        {
            System.out.print(data[i]+"\t");
        }
    }
}
```

Output:

```
Enter 10 numbers
2
4
6
8
10
12
14
16
18
20
The elements in the array are
20      18      16      14      12      10      8       6       4       2
```

2. WAP to input two matrices of 3* 3 size and find the sum.

```
import java.util.*;
public class Demo
{
	public static void main(String [] args)
	{
		Scanner sc=new Scanner(System.in);
		int [][] mat1=new int[3][3];
		int [][] mat2=new int[3][3];
		int [][] mat3=new int[3][3];

		System.out.println("Enter the elements of first matrix");
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				mat1[i][j]=sc.nextInt();
			}
		}

		System.out.println("Enter the elements of Second matrix");

		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				mat2[i][j]=sc.nextInt();
			}
		}
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				mat3[i][j]=mat1[i][j] + mat2[i][j];
			}
		}

		System.out.println("Result=");
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				System.out.print(mat3[i][j]+"\t");
			}
			System.out.println();
		}
	}
}
```

Output:

```
Enter the elements of first matrix
1
2
3
4
5
6
7
8
9
Enter the elements of Second matrix
1
2
3
4
5
6
7
8
9
Result=
2       4       6
8       10      12
14      16      18
```

3. An array is called balanced if it's even numbered element a[0], a[2]… etc. are all even and its odd numbered element a[1], a[3]… etc are all odd. Write a function named isBalanced that accepts an array of interger and returns 1 if the array is balanced otherwise it returns 0.

```
import java.util.*;
class Test
{
        private int[] data;
        public void setData( int []data)
        {
                this.data=data;
        }
        public int isBalanced()
        {
                int flag=1;
                for(int i=0;i<data.length;i++)
                {
                        if(i%2==0)
                        {
                                if(data[i]%2==1)
                                {
                                        flag=0;
                                }
                        }
                        else
                        {
                                if(data[i]%2==0)
                                {
                                        flag=0;
                                }
                        }
                }
                return(flag);
        }
}
```

```
public class Demo
{
	public static void main(String [] args)
	{
		int [] data=new int[10];
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the elements of array");
		for(int i=0;i<10;i++)
		{
			data[i]=sc.nextInt();
		}
		Test t=new Test();
		t.setData(data);
		int res=t.isBalanced();
		if(res==1)
		{
			System.out.println("The input Array Is Balanced");
		}
		else
		{
			System.out.println("The input Array Is Not Balanced");
		}
	}
}
```

For each loop

- The Java for-each loop or enhanced for loop is an alternative approach to traverse the array or collection(ArrayList) in Java.
- For-each is another array traversing technique like for loop, while loop and do-while loop.
- The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable.
- It is known as the for-each loop because it traverses each element one by one.

Note:

- ArrayList is a part of **collection framework** and is present in java.util package.
- It provides us with dynamic arrays in Java.
- The advantage of ArrayList is that it has no size limit. It is more flexible than the traditional array.
- Some frequently used method are
    1. Add(Object o): This method is used to add an element at the end of the ArrayList.
    2. remove(Objecct o): This method is used to simply remove an object from the ArrayList.

Syntax:

```
for(data_type variable : array | collection)
{
	//body of for-each loop
}
```

- The Java for-each loop traverses the array or collection until the last element. For each element, it stores the element in the variable and executes the body of the for-each loop.

Example:
WAP to input name of 10 students and display using for-each loop.

```
import java.util.*;
public class Demo
{
    public static void main(String [] args)
    {
        Scanner sc=new Scanner(System.in);
        String []name=new String[10];
        System.out.println("Enter name 10 Students");
        for(int i=0;i<10;i++)
        {
            name[i]=sc.nextLine();
        }
        System.out.println("Name of the students are");

        for(String str:name)
        {
            System.out.print(str+"\t");
        }
    }
}
```

WAP to demonstrate displaying all the elements of collection(ArrayList) using for each loop.

```
import java.util.*;
public class Demo
{
    public static void main(String [] args)
    {
        //Creating a list of elements
        ArrayList<String> list=new ArrayList<String>();
        list.add("Ram");
        list.add("Hari");
        list.add("Shiva");

        //traversing the list of elements using for-each loop
        for(String s:list)
        {
          System.out.println(s);
        }
    }
}
```

Assignment-1

1. Create a class called MathTechnique that supports the following constant and methods.
    i. Pi=3.1416
    ii. areaOfRectangle(float length, float breadth) to return the area of rectangle
    iii. perimeterOfRectangel(float length, float breadth) to return the perimeter of rectangle
    iv. areaOfCircle(float radius) that returns the area of circle
    v. perimeterOfCircle(float radius) that returns the perimeter of circle

    Finally, write main program to test the above class.
2. Create a class StringOperation with following member function inside package StringPackage.
    a. boolean isPalindrome(String str): Return true if the input string is palindrome otherwise false

b. String reverse(String str): Return the reverse form of input string
c. int length(String str): Return length of input string
d. String uppercase(String str): Return the uppercase form of input string

Finally, write main program to test the above class.

3. An Education institute wishes to maintain a database of its employees. The database is divided into a number of classes whose hierarchical relationships are shown in figure. The figure also shows the minimum information required for each class. Specify all the classes and define functions to create the database and retrieve individual information as a when required. What form of inheritance will the classes hold in this case?

[Hybrid: Multilevel + Hierarchical]