UNIT – II

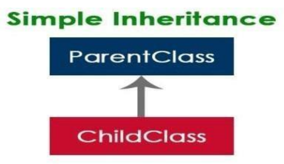
Inheritance

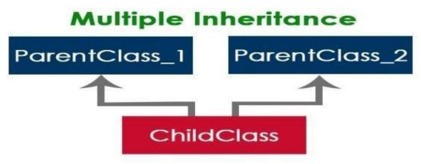
Inheritance Concept:

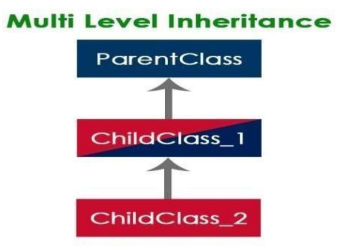
- The process of acquiring the properties of one class to another class is called as Inheritance.
- The inheritance is a very useful and powerful concept of object-oriented programming.
- In java, using the inheritance concept, we can use the existing features of one class in anotherclass.
- The inheritance provides a greate advantage called code re-usability.
- With the help of code re-usability, the commonly used code in an application need not be writtenagain and again.
- The **Parent class** is the class which provides features to another class. The parent class is also known as **Base class** or **Superclass**.
- The **Child class** is the class which receives features from another class. The child class is alsoknown as the **Derived Class** or **Subclass.**
- In the inheritance, the child class acquires the features from its parent class. But the parent classnever acquires the features from its child class.

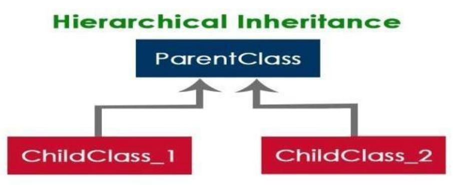
There are five types of inheritances, and they are as follows.

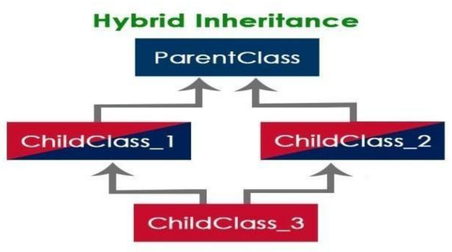
1. Simple Inheritance (or) Single Inheritance
2. Multiple Inheritance
3. Multi-Level Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance











Creating Child Class in java

- ➔ In java, we use the keyword **extends** to create a child class. The following syntax used to createa child class in java.
- ➔ In a java programming language, a class extends only one class. Extending multiple classes isnot allowed in java.

Syntax

```
Class  <ChildClassName>  extends  <ParentClassName>
{
   ...
   //Implementation of child class
   ...
}
```

1. Single Inheritance

In this type of inheritance, one child class derives from one parent class.

```
// Java program to demonstrate Single Inheritance

class  ParentClass{
        int a;
        void setData(int a) {
                this.a = a;
        }
}
class ChildClass extends ParentClass{
        void showData() {
                System.out.println("Value of a is " + a);
        }
}
public class SingleInheritance {

        public static void main(String[] args) {

                ChildClass obj = new ChildClass();
                obj.setData(100);
                obj.showData();
        }

}
```

OUTPUT

```
Value of a is 100
```

2. Multi-level Inheritance

In this type of inheritance, the child class derives from a class which already derived from another class.

```
// Java program to demonstrate Multilevel Inheritance
```

```
class ParentClass{
	int a;
	void setData(int a) {
		this.a = a;
	}
}
class ChildClass extends ParentClass{
	void showData() {
		System.out.println("Value of a is " + a);
	}
}
class ChildChildClass extends ChildClass{
	void display() {
		System.out.println("Inside ChildChildClass!");
	}
}
class MultipleInheritance {

	public static void main(String[] args) {

		ChildChildClass obj = new ChildChildClass();
		obj.setData(100);
		obj.showData();
		obj.display();
	}
}
```

OUTPUT

```
Value of a is 100
Inside ChildChildClass!
```

3. Hierarchical Inheritance

In this type of inheritance, two or more child classes derive from one parent class.

```
  // Java program to demonstrate Hierarchical Inheritance

class ParentClass{
	int a;
	void setData(int a) {
		this.a = a;
	}
}
class ChildClass extends ParentClass{
	void showData() {
		System.out.println("Inside ChildClass!");
		System.out.println("Value of a is " + a);
	}
}
class ChildClassToo extends ParentClass{
	void display() {
		System.out.println("Inside ChildClassToo!");
		System.out.println("Value of a is " + a);
```

```
        }
}
class HierarchicalInheritance {

        public static void main(String[] args) {

                ChildClass child_obj = new ChildClass();
                child_obj.setData(100);
                child_obj.showData();

                ChildClassToo childToo_obj = new ChildClassToo();
                childToo_obj.setData(200);
                childToo_obj.display();

        }

}
```

OUTPUT

```
Inside ChildClass!
Value of a is 100
Inside ChildClassToo!
Value of a is 200
```

4. Hybrid Inheritance

The hybrid inheritance is the combination of more than one type of inheritance.

Forms of Inheritance

The inheritance concept used for the number of purposes in the java programming language. One of the main purposes is substitutability. The substitutability means that when a child class acquires properties from its parent class, the object of the parent class may be substituted with the child class object. For example, if B is a child class of A, anywhere we expect an instance of A we can use an instance of B.

The substitutability can achieve using inheritance, whether using extends or implements keywords. The following are the different forms of inheritance in java.

- Specialization
- Specification
- Construction
- Extension
- Limitation
- Combination

Specialization

It is the most ideal form of inheritance. The subclass is a special case of the parent class. It holds the principle of substitutability.

Specification

This is another commonly used form of inheritance. In this form of inheritance, the parent class just specifies which methods should be available to the child class but doesn't implement them. The java provides concepts like abstract and interfaces to support this form of inheritance. It holds the principle of substitutability.

Construction

This is another form of inheritance where the child class may change the behavior defined by the parent class (overriding). It does not hold the principle of substitutability.

Extension

This is another form of inheritance where the child class may add its new properties. It holds the principle of substitutability.

Limitation

This is another form of inheritance where the subclass restricts the inherited behavior. It does not hold the principle of substitutability.

Combination

This is another form of inheritance where the subclass inherits properties from multiple parent classes. Java does not support multiple inheritance type.

Benefits and Costs of Inheritance

The inheritance is the core and more useful concept Object Oriented Programming. With inheritance, we will be able to override the methods of the base class so that the meaningful implementation of the base class method can be designed in the derived class. An inheritance leads to less development and maintenance costs. Vides lot of benefits and few of them are listed below.

Benefits of Inheritance

- Inheritance helps in code reuse. The child class may use the code defined in the parent classwithout re-writing it.
- Inheritance can save time and effort as the main code need not be written again.
- Inheritance provides a clear model structure which is easy to understand.
- An inheritance leads to less development and maintenance costs.
- With inheritance, we will be able to override the methods of the base class so that the meaningful implementation of the base class method can be designed in the derived class. An inheritance leads to less development and maintenance costs.
- In inheritance base class can decide to keep some data private so that it cannot be altered by the derived class.

Costs of Inheritance

- Inheritance decreases the execution speed due to the increased time and effort it takes, theprogram to jump through all the levels of overloaded classes.
- Inheritance makes the two classes (base and inherited class) get tightly coupled. This meansone cannot be used independently of each other.
- The changes made in the parent class will affect the behavior of child class too.
- The overuse of inheritance makes the program more complex.

Limitations of Inheritance

- Main disadvantage of using inheritance is that the two classes (parent and child class) gets **tightly coupled**. This means that if we change code of parent class, it will affect to all the child classes which is inheriting/deriving the parent class, and hence, it cannot be independent of eachother.

- ➔ Inherited functions work slower than normal function as there is indirection.
- ➔ Improper use of inheritance may lead to wrong solutions.
- ➔ Often, data members in the base class are left unused which may lead to memory wastage.
- ➔ Inheritance increases the coupling between base class and derived class.

Access Modifiers(Member Access)

In Java, the access specifiers (also known as access modifiers) used to restrict the scope oraccessibility of a class, constructor, variable, method or data member of class and interface.

There are four access specifiers, and their list is below.

- default (or) no modifier
- **public**
- protected
- **private**

- ➔ The **default members** are accessible within the same package but not outside the package.
- ➔ The **public members** can be accessed everywhere.
- ➔ The **protected members** are accessible to every child class (same package or other packages).
- ➔ The **private members** can be accessed only inside the same class.

// Java program to demonstrate Access Modifiers

```
class ParentClass {
        int a = 10;
        public int b = 20;
        protected int c = 30;
        private int d = 40;

        void showData() {
                System.out.println("Inside ParentClass");
      System.out.println("___________________________");
                System.out.println("default member a = " + a); System.out.println("public
                member b = " + b); System.out.println("protected member c = " + c);
                System.out.println("private member d = " + d);
        }
}

class ChildClass extends ParentClass {

        void accessData() {
        System.out.println("
                _____________________________________")
                ;System.out.println("Inside ChildClass");
      System.out.println("___________________________");
                System.out.println("default member a = " + a);
                System.out.println("public member b = " + b);
                System.out.println("protected member c = " + c);
                //System.out.println("d = " + d);        // private member can't be accessed
        }
}
class AccessModifiersExample {

        public static void main(String[] args) {

                ChildClass obj = new ChildClass();
                obj.showData();
                obj.accessData();
        }
}
```

OUTPUT

```
Inside ParentClass
_________________________
default member a = 10
public member b = 20
protected member c = 30
private member d = 40
_________________________
Inside ChildClass
_________________________
default member a = 10
public member b = 20
protected member c = 30
```

super keyword

- ➔ In java, super is a keyword used to refers to the parent class object.
- ➔ The super keyword came into existence to solve the naming conflicts in the inheritance.
- ➔ When both parent class and child class have members with the same name, then the superkeyword is used to refer to the parent class version.
- ➔ the super keyword is used for the following purposes.

• To refer parent class data members as:	**super.variable**
• To refer parent class methods as:	**super.method ()**
• To call parent class constructor as:	**super (values)**

- ➔ The **super** keyword is used inside the child class only.

// super to refer parent class data members

When both parent class and child class have data members with the same name, then the superkeyword is used to refer to the parent class data member from child class.

```
class ParentClass

        int num = 10;
}

class ChildClass extends ParentClass{

        int num = 20;
```

```
          void showData() {
                    System.out.println("Inside the ChildClass");
                    System.out.println("ChildClass num = " + num);
                    System.out.println("ParentClass num = " + super.num);
          }
}

public class SuperKeywordExample
{

          public static void main(String[] args)
 {
                    ChildClass obj = new ChildClass();

                    obj.showData();

                    System.out.println("\nInside the non-child class");
                    System.out.println("ChildClass num = " + obj.num);
                    //System.out.println("ParentClass num = " + super.num);          //super can't be used here

          }

}
```

OUTPUT

```
Inside  the  ChildClass
ChildClass  num  =  20
ParentClass num = 10

Inside the non-child class
ChildClass num = 20
```

// super to call parent class method

When both parent class and child class have method with the same name, then the super keyword isused to refer to the parent class method from child class.

```
class ParentClass
{
          int num1 = 10;

          void showData(){
                    System.out.println("\nInside the ParentClass showData method");
                    System.out.println("ChildClass num = " + num1);
          }
}

class ChildClass extends ParentClass
{
          int num2 = 20;
```

```
        void showData()
        {
                System.out.println("\nInside the ChildClass showData method");
                System.out.println("ChildClass num = " + num2);
                super.showData();
        }
}

class SuperKeywordExample
{
        public static void main(String[] args)
        {
                ChildClass obj = new ChildClass();
                obj.showData();
                //super.showData();      // super can't be used here
        }
}
```

OUTPUT

```
Inside the ChildClass showData method
ChildClass num = 20

Inside the ParentClass showData method
ChildClass num = 10
```

// super to call parent class constructor

When an object of child class is created, it automatically calls the parent class default-constructor beforeit's own. But, the parameterized constructor of parent class must be called explicitly using the **super** keyword inside the child class constructor.

```
class ParentClass
{
        int num1;
        ParentClass()
        {
                System.out.println("\nInside the ParentClass default constructor");
                num1 = 10;
        }
ParentClass(int value)
  {
                System.out.println("\nInside the ParentClass parameterized constructor");
                num1 = value;
        }
}
class ChildClass extends ParentClass
        int num2;
        ChildClass()
        {
                super(100);
                System.out.println("\nInside the ChildClass constructor");
                num2 = 200;
        }
}
```

```
class SuperKeywordExample
{
        public static void main(String[] args)
        {
                ChildClass obj = new ChildClass();
        }
}
```

To call the parameterized constructor of the parent class, the super keyword must be the first statement inside the child class constructor, and we must pass the parameter values.

OUTPUT

```
Inside the ParentClass parameterized constructor
Inside the ChildClass constructor
```

final” keyword

- final keyword before a class prevents inheritance.
  e.g.: final class A
  class B extends A //invalid
- final keyword before a method prevents overriding.
- final keyword before a variable makes that variable as a constant.
  e.g.: final double PI = 3.14159; //PI is a constant.

final with variables

- When a variable defined with the **final** keyword, it becomes a constant, and it does not allow usto modify the value.
- The variable defined with the final keyword allows only a one-time assignment, once a valueassigned to it, never allows us to change it again.

```
public class FinalVariableExample
{
        public static void main(String[] args)
        {
                final int a = 10;
                System.out.println("a = " + a);
                a = 100;        // Can't be modified
        }
}
```

OUTPUT

```
Main.java:9:    error: cannot assign a value to final variable a
                a = 100;        // Can't be modified
```

final with methods

- ➔ When a method defined with the **final** keyword, it does not allow it to override.
- ➔ The final method extends to the child class, but the child class can not override or re-define it.
- ➔ It must be used as it has implemented in the parent class.

```
class ParentClass
{
        int num = 10;
        final void showData()
         {
                System.out.println("Inside ParentClass showData() method");
                System.out.println("num = " + num);
        }
}

class ChildClass extends ParentClass
{
        void showData()
         {
                System.out.println("Inside ChildClass showData() method");
                System.out.println("num = " + num);
        }
}

class FinalKeywordExample
{

        public static void main(String[] args)
        {
                ChildClass obj = new ChildClass();
                obj.showData();
        }
}
```

OUTPUT

```
Main.java:13: error: showData() in ChildClass cannot override showData() in ParentClass
        void showData()
  overridden method is final
```

final with class

When a class defined with final keyword, it can not be extended by any other class.

```
final class ParentClass
{
        int num = 10;
        void showData()
    {
                System.out.println("Inside ParentClass showData() method");
                System.out.println("num = " + num);
        }
}

class ChildClass extends ParentClass
{
}

class FinalKeywordExample
{
            public static void main(String[] args)
   {
                        ChildClass obj = new ChildClass();
            }
}
```

OUTPUT

```
Main.java:11: error: cannot inherit from final ParentClass
class ChildClass extends ParentClass
```

Polymorphism

The polymorphism is the process of defining same method with different implementation. That meanscreating multiple methods with different behaviors.

In java, polymorphism implemented using method overloading and method overriding.

Ad hoc polymorphism

- ➔ The ad hoc polymorphism is a technique used to define the same method with differentimplementations and different arguments.
- ➔ In a java programming language, ad hoc polymorphism carried out with a method overloadingconcept.
- ➔ In ad hoc polymorphism the method binding happens at the time of compilation.
- ➔ Ad hoc polymorphism is also known as compile-time polymorphism.
- ➔ Every function call binded with the respective overloaded method based on the arguments.
- ➔ The ad hoc polymorphism implemented within the class only.

```
import java.util.Arrays;
class AdHocPolymorphismExample
{
            void sorting(int[] list)
            {
                        Arrays.parallelSort(list);
                        System.out.println("Integers after sort: " + Arrays.toString(list) );
            }
            void sorting(String[] names)
            {
                        Arrays.parallelSort(names);
                        System.out.println("Names after sort: " + Arrays.toString(names) );
            }

            public static void main(String[] args)
             {
                        AdHocPolymorphismExample obj = new AdHocPolymorphismExample();
                        int list[] = {2, 3, 1, 5, 4};
                        obj.sorting(list); // Calling with  integer  array
                        String[] names = {"rama", "raja", "shyam", "seeta"};
                        obj.sorting(names);        // Calling with String array
```

```
        }
}
```

OUTPUT

```
Integers after sort: [1, 2, 3, 4, 5]
Names after sort: [raja, rama, seeta, shyam]
```

Pure polymorphism

- ➔ The pure polymorphism is a technique used to define the same method with the same argumentsbut different implementations.
- ➔ In a java programming language, pure polymorphism carried out with a method overridingconcept.
- ➔ In pure polymorphism, the method binding happens at run time. Pure polymorphism is alsoknown as **run-time polymorphism**.
- ➔ Every function call binding with the respective overridden method based on the object reference.
- ➔ When a child class has a definition for a member function of the parent class, the parent classfunction is said to be overridden.
- ➔ The pure polymorphism implemented in the inheritance concept only.

```
class ParentClass
{
        int num = 10;
        void showData()
  {
                System.out.println("Inside ParentClass showData() method");
                System.out.println("num = " + num);
        }
}

class ChildClass extends ParentClass{

        void showData()
  {
                System.out.println("Inside ChildClass showData() method");
                System.out.println("num = " + num);
        }
}

class PurePolymorphism
{
        public static void main(String[] args)
  {
                ParentClass obj = new ParentClass();
                obj.showData();
                obj = new ChildClass();
                obj.showData();
        }
}
```

OUTPUT

```
Inside ParentClass showData() method
num = 10
Inside ChildClass showData() method
num = 10
```

Method Overriding

- The method overriding is the process of re-defining a method in a child class that is alreadydefined in the parent class.
- When both parent and child classes have the same method, then that method is said to be theoverriding method.
- The method overriding enables the child class to change the implementation of the method whichaquired from parent class according to its requirement.
- In the case of the method overriding, the method binding happens at run time.
- The method binding which happens at run time is known as late binding. So, the methodoverriding follows late binding.
- The method overriding is also known as **dynamic method dispatch** or **run time polymorphism** or **pure polymorphism**.

```
class Animal
{
        void move()
         {
                System.out.println ("Animals can move");
         }
}
class Dog extends Animal
{
        void move()
         {
                System.out.println ("Dogs can walk and run");
         }
}
class OverRide
{
        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

```
Animals can move
Dogs can walk and run
```

Abstract Class

- ➔ A method with method body is called **concrete method**. In general any class will have allconcrete methods.
- ➔ A method without method body is called abstract method.
- ➔ A class that contains abstract method is called abstract class.
- ➔ A class prefixed with abstract keyword is known as an abstract class.
- ➔ In java, an abstract class may contain abstract methods (methods without implementation) andalso non-abstract methods (methods with implementation).
- ➔ An abstract class is a class with zero or more abstract methods
- ➔ ·An abstract class contains instance variables & concrete methods in addition to abstract
- ➔ methods.
- ➔ It is not possible to create objects to abstract class.
- ➔ But we can create a reference of abstract class type.
- ➔ All the abstract methods of the abstract class should be implemented in its sub classes.
- ➔ If any method is not implemented, then that sub class should be declared as „abstract".
- ➔ Abstract class reference can be used to refer to the objects of its sub classes.
- ➔ Abstract class references cannot refer to the individual methods of sub classes.
- ➔ A class cannot be both „abstract" & „final".

  e.g.: final abstract class A // invalid

// Example program for abstract class.

```
abstract class Figure
{
    double dim1;
    double dim2;
    Figure (double a, double b)
    {
        dim1 = a;
        dim2 = b;
    }
    abstract double area (); // area is now an abstract method
}
class Rectangle extends Figure
{
    Rectangle (double a, double b)
    {
        super (a, b);
    }
    double area () // override area for rectangle
    {
        System.out.println ("Inside Area of Rectangle.");
        return dim1 * dim2;
    }
}
class Triangle extends Figure
{
    Triangle (double a, double b)
    {
        super (a, b);
    }
    double area() // override area for right triangle
    {
        System.out.println ("Inside Area of Triangle.");
        return dim1 * dim2 / 2;
    }
}
public class AbstractArea {
    public static void main(String args[])
    {
       // Figure f = new Figure(10, 10); // illegal now
        //Rectangle r = new Rectangle(9, 5);
        Figure r = new Rectangle(9, 5);
        Figure  t = new Triangle(10, 8);
        System.out.println("Area of reactangle is " + r.area());
        System.out.println("Area of triangle is " + t.area());
    }
}
```

OUTPUT

```
Area of a circle is : 326.68559999999997Volume
of a circle is : 64.056
```

Object Class

- In java, the Object class is the super most class of any class hierarchy. The Object class in thejava programming language is present inside the **java.lang** package.
- Every class in the java programming language is a subclass of Object class by default.
- The Object class is useful when you want to refer to any object whose type you don't know. Because it is the super class of all other classes in java, it can refer to any type of object.

Methods of Object class

Method	Description	Return Value
getClass()	Returns Class class object	Object
hashCode()	returns the hashcode number for object being used.	int
equals(Object obj)	compares the argument object to calling object.	boolean
clone()	Compares two strings, ignoring case	int
concat(String)	Creates copy of invoking object	object
toString()	returns the string representation of invoking object.	String
notify()	wakes up a thread, waiting on invoking object's monitor.	void
notifyAll()	wakes up all the threads, waiting on invoking object's monitor.	void
wait()	causes the current thread to wait, until another thread notifies.	void
wait(long,int)	causes the current thread to wait for the specified millisecondsand nanoseconds, until another thread notifies.	void

finalize()	It is invoked by the garbage collector before an object is beinggarbage collected.	void

Packages

- In java, a package is a container of classes, interfaces, and sub-packages. We may think of it asa folder in a file directory.
- We use the packages to avoid naming conflicts and to organize project-related classes,interfaces, and sub-packages into a bundle.
- In java, the packages have divided into two types.

1. Built-in Packages
2. User-defined Packages

Built-in Packages

- The built-in packages are the packages from java API. The Java API is a library of pre-definedclasses, interfaces, and sub-packages. The built-in packages were included in the JDK.
- There are many built-in packages in java, few of them are as java, lang, io, util, awt, javax, swing,net, sql, etc.
- We need to import the built-in packages to use them in our program. To import a package, weuse the import statement**.**

User-defined Packages

The user-defined packages are the packages created by the user. User is free to create their ownpackages.

Defining a Package in java

- We use the package keyword to create or define a package in java programming language.
- The package statement must be the first statement in the program.
- The package name must be a single word.
- The package name must use Camel case notation.

Syntax

```
package  packageName;
```

// Java program to create a package mypackage with Addition class

```
package mypackage;
public class Addition
{
   private double d1,d2;
   public Addition(double a,double b)
   {
      d1 = a;
      d2 = b;
   }
```

```
        public void sum()
        {
            System.out.println ("Sum of two given numbers is : " + (d1+d2) );
        }
    }
```

Now, save the above code in a file **Addition.java**, and compile it using the following command.

javac -d . Addition.java

The –d option tells the Java compiler to create a separate directory and place the .class file in that directory (package). The (.) dot after –d indicates that the package should be created in the currentdirectory. So, our package ***mypackage*** with Addition class is ready.

Importing packages:

- In java, the import keyword used to import built-in and user-defined packages. When a packagehas imported, we can refer to all the classes of that package using their name directly.
- The **import** statement must be after the package statement, and before any other statement.
- Using an import statement, we may import a specific class or all the classes from a package.
- Using one import statement, we may import only one package or a class.
- Using an import statement, we can not import a class directly, but it must be a part of a package.
- A program may contain any number of import statements.
- The import statement imports only classes of the package, but not sub-packages and its classes.
- We may also import sub-packages by using a symbol '.' (dot) to separate parent package andsub-package.

import java.util.*;

- The above import statement util is a sub-package of java package. It imports all the classes of util package only, but not classes of java package.

Importing specific class : Using an importing statement, we can import a specific class.

Syntax

import packageName . ClassName;

Importing all the classes: Using an importing statement, we can import all the classes of a package.To import all the classes of the package, we use * symbol.

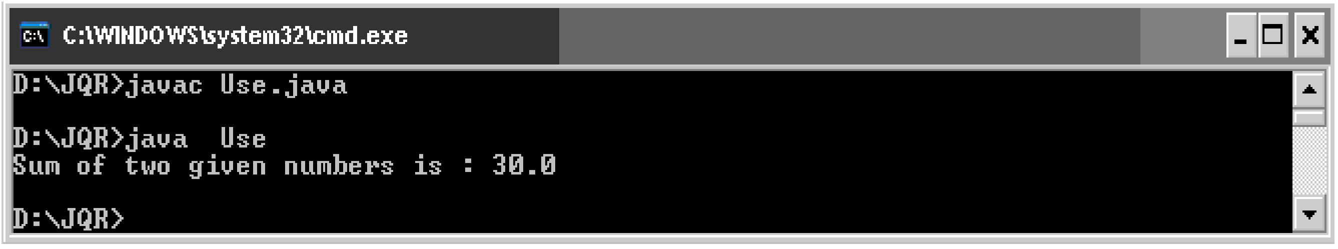
Syntax

import packageName.*;

// Java program to import the Addition class of package mypackage.

```
import mypackage.Addition;
class Use
{
    public static void main(String args[])
    {
        Addition ob1 = new Addition(10,20);
        ob1.sum();
    }
}
```

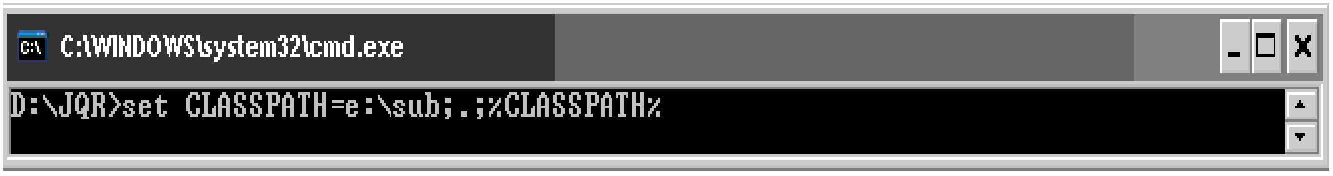
Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Use.java

D:\JQR>java  Use
Sum of two given numbers is : 30.0

D:\JQR>
```

CLASSPATH

- The CLASSPATH is an environment variable that tells the Java compiler where to look for class files to import.
- If the package pack is available in different directory, in that case the compiler should be given information regarding the package location by mentioning the directory name of the package in the classpath.
- If our package exists in e:\sub then we need to set class path as follows:



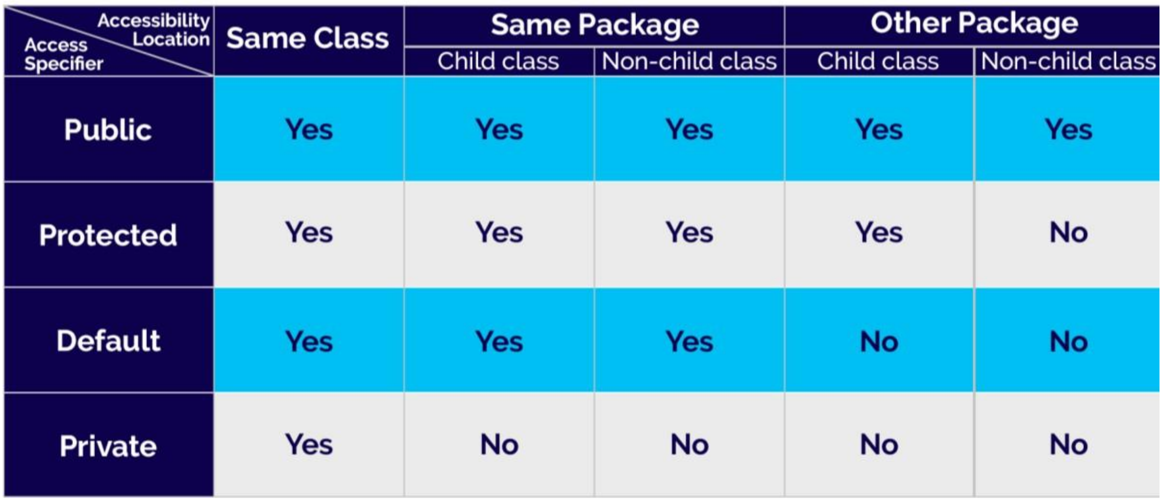
```
C:\WINDOWS\system32\cmd.exe
D:\JQR>set CLASSPATH=e:\sub;.;%CLASSPATH%
```

We are setting the classpath to **e:\sub** directory and current directory (.) and %CLASSPATH% meansretain the already available classpath as it is.

Access protection in java packages

- In java, the access modifiers define the accessibility of the class and its members. For example,private members are accessible within the same class members only.
- Java has four access modifiers, and they are default, private, protected, and public.
- In java, the package is a container of classes, sub-classes, interfaces, and sub-packages.
- The class acts as a container of data and methods.
- So, the access modifier decides the accessibility of class members across the differentpackages.
- In java, the accessibility of the members of a class or interface depends on its access specifiers.
- The following table provides information about the visibility of both data members and methods.

Access control for members of class and interface in java



Access Specifier \ Accessibility Location	Same Class	Same Package		Other Package	
		Child class	Non-child class	Child class	Non-child class
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

- The **public** members can be accessed everywhere.
- The **private** members can be accessed only inside the same class.
- The **protected** members are accessible to every child class (same package or other packages).
- The **default** members are accessible within the same package but not outside the package.

// Example Program

```
class ParentClass
{
    int a = 10;
    public int b = 20;
    protected int c = 30;
    private int d = 40;

    void showData()
    {
        System.out.println("Inside ParentClass");
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
    }
}

class ChildClass extends ParentClass
{
    void accessData()
    {
        System.out.println("Inside ChildClass");
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        //System.out.println("d = " + d);  // private member can't be accessed
    }
}
public class AccessModifiersExample
{

    public static void main(String[] args)
    {
        ChildClass obj = new ChildClass();
        obj.showData();
        obj.accessData();
    }

}
```

OUTPUT

```
Inside ParentClassa
= 10
b = 20
c = 30
d = 40
Inside ChildClassa
= 10
b = 20
c = 30
```

Interfaces

- An interface is a specification of method prototypes.
- In java, an ***interface*** is similar to a class, but it contains abstract methods and static finalvariables only.
- The interface in Java is another mechanism to achieve abstraction.
- All the methods of an interface, implemented by the class that implements it.
- An interface contains zero or more abstract methods.
- All the methods of interface are public, abstract by default.

- An interface may contain variables which are by default public static final.
- Once an interface is written any third party vendor can implement it.
- All the methods of the interface should be implemented in its implementation classes.
- If any one of the method is not implemented, then that implementation class should be declaredas abstract.
- We cannot create an object to an interface.
- We can create a reference variable to an interface.
- An interface cannot implement another interface.
- An interface can extend another interface.
- A class can implement multiple interfaces.

Defining an interface in java

Defining an interface is similar to that of a class. We use the keyword interface to define an interface. Allthe members of an interface are public by default. The following is the syntax for defining an interface.

Syntax

```
interface  InterfaceName
{
   ...
   members declaration;
   ...
}
```

Implementing an Interface in java

- In java, an **interface** is implemented by a class.
- The class that implements an interface must provide code for all the methods defined in theinterface, otherwise, it must be defined as an abstract class.
- The class uses a keyword ***implements*** to implement an interface.
- A class can implement any number of interfaces.
- When a class wants to implement more than one interface, we use the ***implements*** keyword isfollowed by a comma-separated list of the interfaces implemented by the class.

Syntax

```
class  className  implements InterfaceName
{
   ...
   boby-of-the-class
   ...
}
```

// Example Program to implement interface.

```
interface Human {

    void learn(String str);
    void work();

    int duration = 10;

}

class Programmer implements Human{
    public void learn(String str) {
        System.out.println("Learn using " + str);
    }
    public void work() {
        System.out.println("Develop applications");
    }
}

public class HumanTest {

    public static void main(String[] args) {
        Programmer trainee = new Programmer();
        trainee.learn("coding");
        trainee.work();
    }
}
```

OUTPUT

```
Learn using coding
Develop applications
```

The above code defines an interface **Human** that contains two abstract methods learn(), work() and oneconstant duration. The class Programmer implements the interface. As it implementing the Human interface it must provide the body of all the methods those defined in the Human interface.

// Java Program to implement multiple inheritance using interfaces.

```
interface Father
{
    double PROPERTY = 10000;
    double HEIGHT = 5.6;
}
interface Mother
{
    double PROPERTY = 30000;
    double HEIGHT = 5.4;
}
class MyClass implements Father, Mother
{
    void show()
    {
        System.out.println("Total property is :" +(Father.PROPERTY + Mother.PROPERTY));
        System.out.println ("Average height is :" + (Father.HEIGHT  +  Mother.HEIGHT)/2 );
    }
}
class InterfaceDemo
    {
        public static void main(String args[])
    {
        MyClass ob1 = new MyClass();
        ob1.show();
    }
}
```

OUTPUT

```
Total property is :40000.0
Average height is :5.5
```

Nested Interfaces

- The interface that defined inside another interface or a class is known as nested interface.
- The nested interface is also referred as inner interface.
- The nested interface declared within an interface is public by default.
- The nested interface declared within a class can be with any access modifier.
- Every nested interface is static by default.
- The nested interface cannot be accessed directly.
- The nested interface that defined inside another interface must be accessed as **OuterInterface.InnerInterface**.
- The nested interface that defined inside a class must be accessed as **ClassName.InnerInterface**.

```
interface OuterInterface{
   void outerMethod();

   interface InnerInterface{
      void innerMethod();
   }
}

class OnlyOuter implements OuterInterface{
   public void outerMethod() {
      System.out.println("This is OuterInterface method");
   }
}

class OnlyInner implements OuterInterface.InnerInterface{
   public void innerMethod() {
      System.out.println("This is InnerInterface method");
   }
}

public class NestedInterfaceExample {
    public static void main(String[] args) {
        OnlyOuter obj_1 = new OnlyOuter();
        OnlyInner obj_2 = new OnlyInner();

        obj_1.outerMethod();
        obj_2.innerMethod();
    }

}
```

OUTPUT

```
This is OuterInterface method
This is InnerInterface method
```

Extending an Interface

- An interface can extend another interface.
- An interface can not extend multiple interfaces.
- An interface can implement neither an interface nor a class.
- The class that implements child interface needs to provide code for all the methods defined inboth child and parent interfaces.

// Java Program to demonstrate extending an interface.

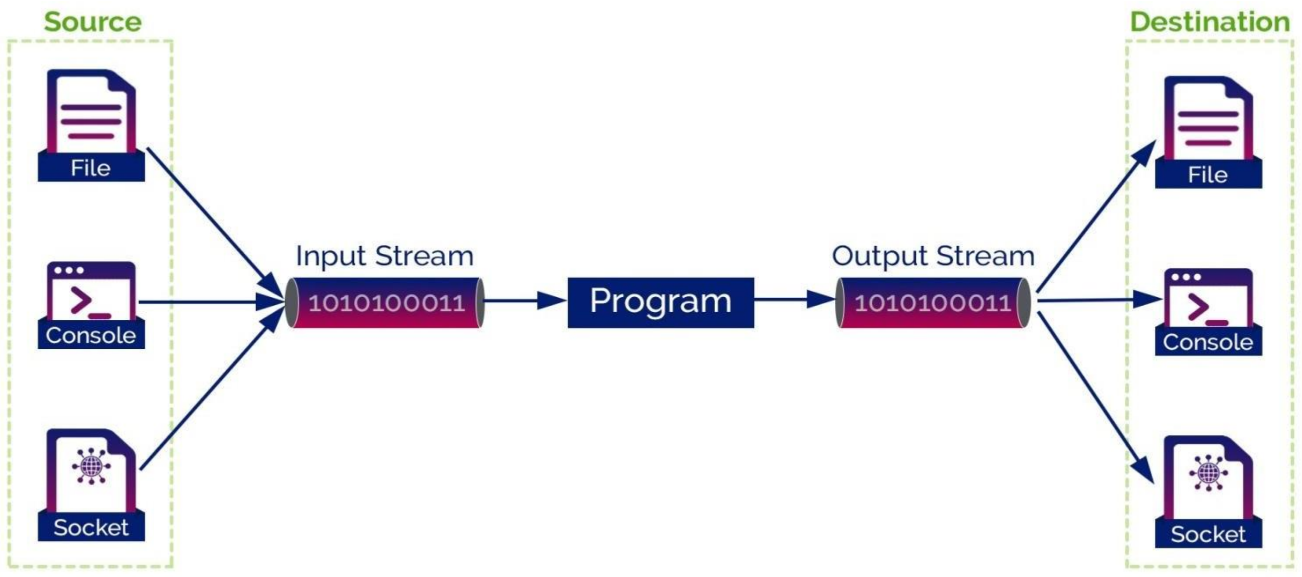
```
interface ParentInterface{
    void parentMethod();
}

interface ChildInterface extends ParentInterface{
    void childMethod();
}

class ImplementingClass implements ChildInterface{

    public void childMethod() {
        System.out.println("Child Interface method!!");
    }

    public void parentMethod() {
        System.out.println("Parent Interface mehtod!");
    }
}

public class ExtendingAnInterface {

    public static void main(String[] args) {

        ImplementingClass obj = new ImplementingClass();

        obj.childMethod();
        obj.parentMethod();

    }
}
```

OUTPUT

```
Child Interface method!!
Parent Interface mehtod!
```

Stream based I/O (java.io)

- ➔ A Stream represents flow of data from one place to another place.
- ➔ In java, the IO operations are performed using the concept of streams.
- ➔ A stream means a continuous flow of data.
- ➔ In java, a stream is a logical container of data that allows us to read from and write to it.
- ➔ A stream can be linked to a data source, or data destination, like a console, file or networkconnection by java IO system.
- ➔ The stream-based IO operations are faster than normal IO operations.
- ➔ The Stream is defined in the java.io package.
- ➔ In java, the stream-based IO operations are performed using two separate streams input streamand output stream.
- ➔ The input stream is used for input operations, and the output stream is used for outputoperations.
- ➔ The java stream is composed of bytes.

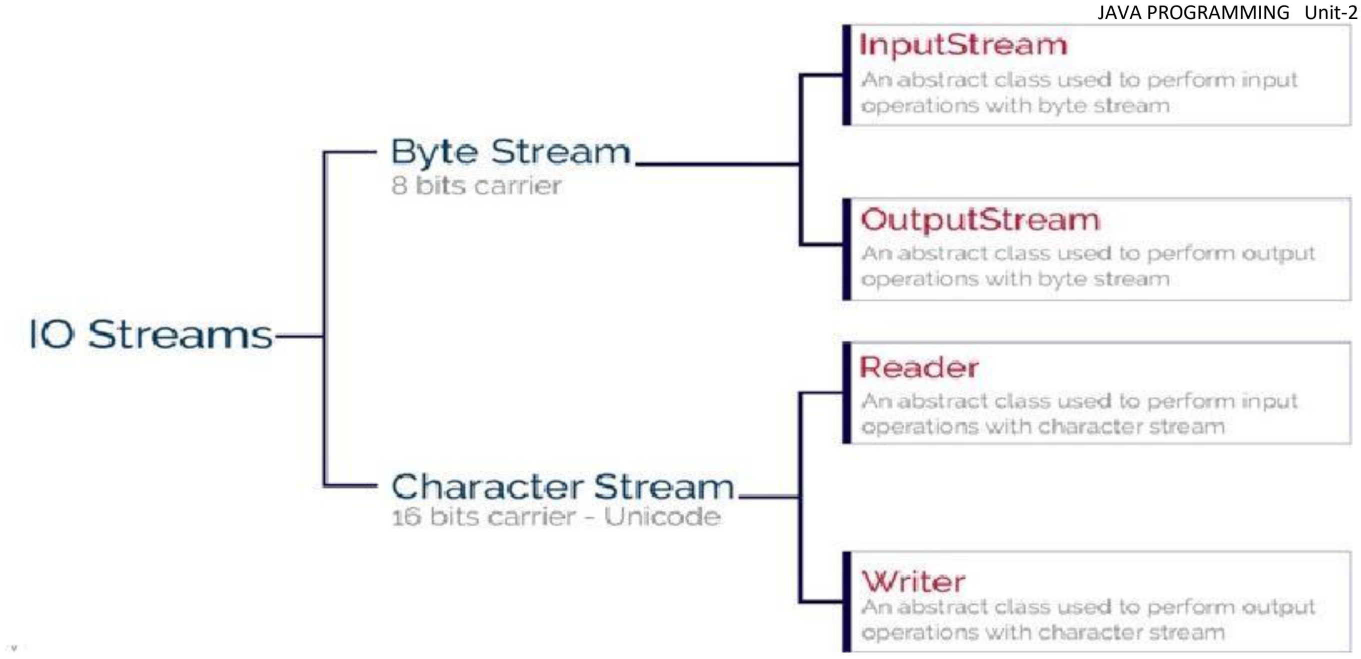


In Java, every program creates 3 streams automatically, and these streams are attached to the console.

- **System.out**: standard output stream for console output operations.
- **System.in**: standard input stream for console input operations.
- **System.err**: standard error stream for console error output operations.Java

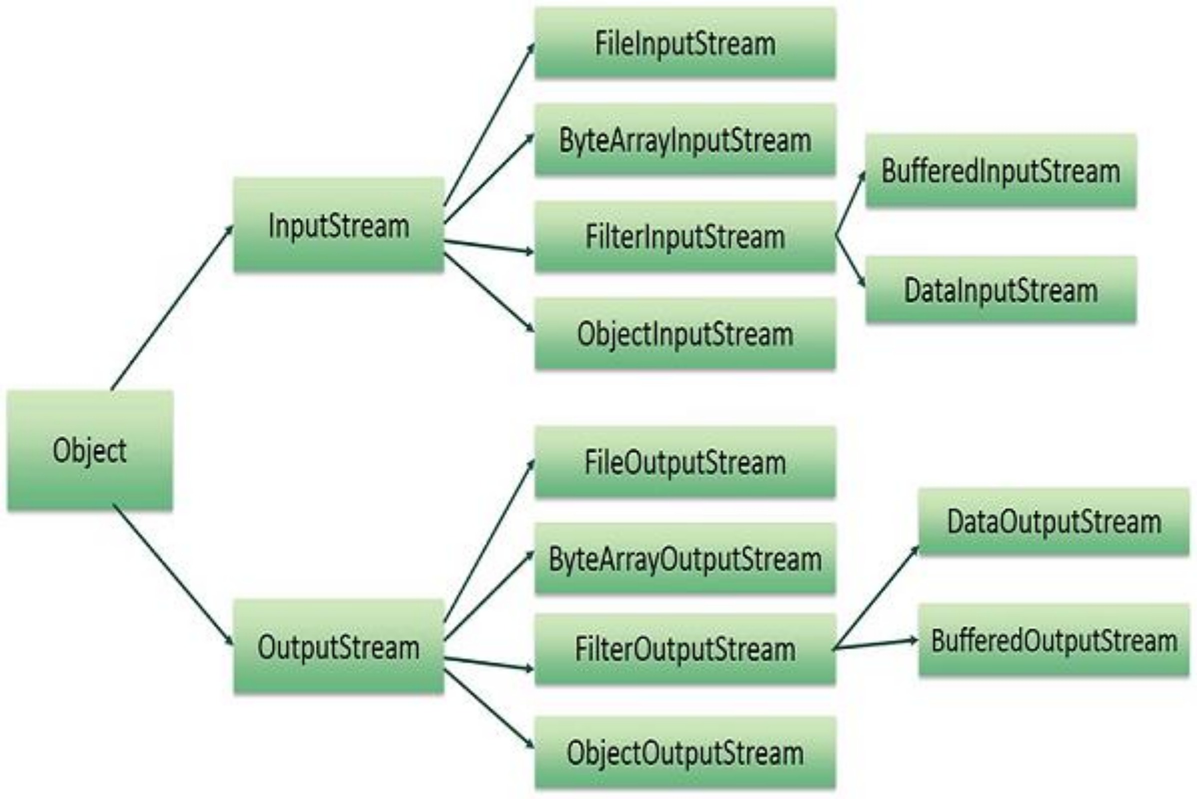
provides two types of streams, and they are as follows.

- **Byte Stream**
- **Character Stream**


IO Streams
Byte Stream
8 bits carrier
InputStream
An abstract class used to perform input operations with byte stream
OutputStream
An abstract class used to perform output operations with byte stream
Character Stream
16 bits carrier - Unicode
Reader
An abstract class used to perform input operations with character stream
Writer
An abstract class used to perform output operations with character stream

Byte Streams

- The byte stream is an 8 bits carrier. The byte stream in java allows us to transmit 8 bits of data.
- The java byte stream is defined by two abstract classes, **InputStream** and **OutputStream.**
- Byte streams are used to handle any characters (text), images, audio and video files.
- FileInputStream/FileOutputStream: They handle data to be read or written to disk files.
- FilterInputStream/FilterOutputStream: They read data from one stream and write it to anotherstream.
- ObjectInputStream/ObjectOutputStream: They handle storage of objects and primitive data.

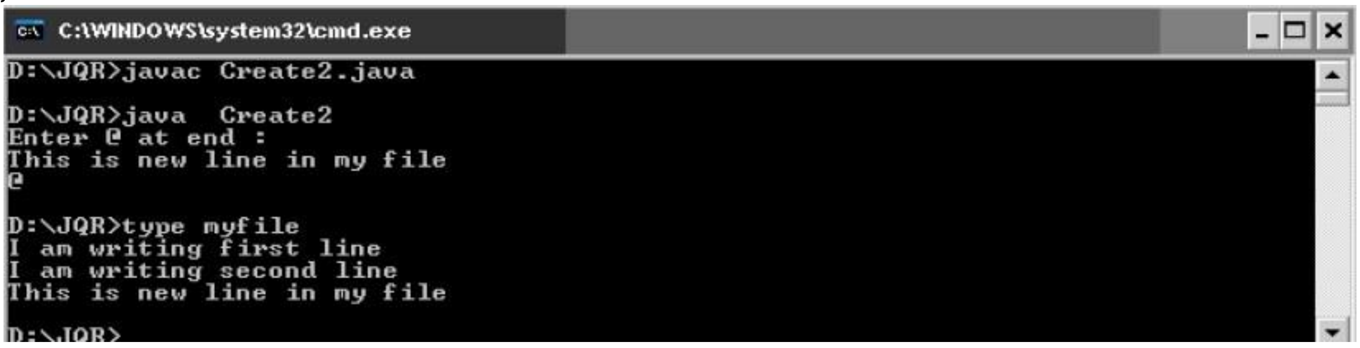


// Java program to read data from the keyboard and write it to a text file using byte stream classes.

```
import java.io.*;
class Create1
{ public static void main(String args[]) throws IOException
   { //attach keyboard to DataInputStream
      DataInputStream dis = new DataInputStream (System.in);
    //attach the file to FileOutputStream
      FileOutputStream fout = new FileOutputStream ("myfile");
    //read data from DataInputStream and write into FileOutputStream
      char ch;
      System.out.println ("Enter @ at end : " ) ;
      while( (ch = (char) dis.read() ) != '@' )
         fout.write (ch);
      fout.close ();
   }
}
```

/* Java program to to improve the efficiency of writing data into a file using BufferedOutputStream. */

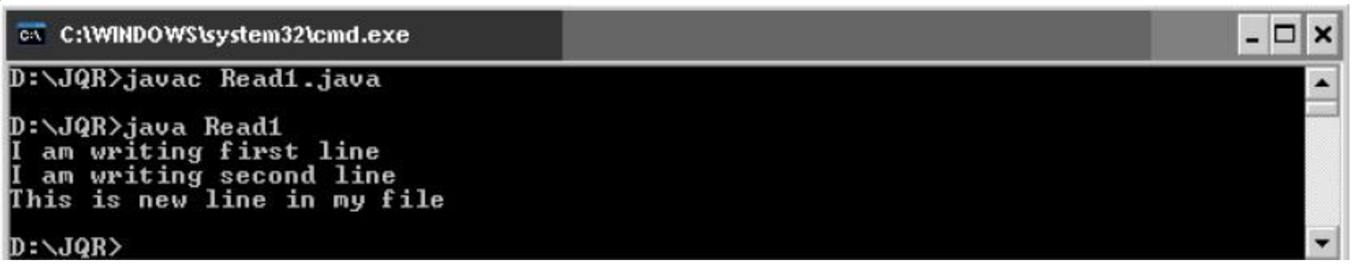
```
import java.io.*;
class Create2
{ public static void main(String args[]) throws IOException
   {
           //attach keyboard to DataInputStream
      DataInputStream dis = new DataInputStream (System.in);
           //attach file to FileOutputStream, if we use true then it will open in append mode
      FileOutputStream fout = new FileOutputStream ("myfile", true);
      BufferedOutputStream bout = new BufferedOutputStream (fout, 1024);
            //Buffer size is declared as 1024 otherwise default buffer size of 512 bytes is used.
             //read data from DataInputStream and write into FileOutputStream
      char ch;
      System.out.println ("Enter @ at end : " ) ;
      while ( (ch = (char) dis.read() ) != '@' )
         bout.write (ch);
      bout.close ();
      fout.close ();
   }
}
```



```
D:\JQR>javac Create2.java

D:\JQR>java  Create2
Enter @ at end :
This is new line in my file
@

D:\JQR>type myfile
I am writing first line
I am writing second line
This is new line in my file

D:\JQR>
```

// Java program program to read data from myfile using FileInputStream
//Reading a text file using byte stream classes

```
import java.io.*;
class Read1
{
   public static void main (String args[]) throws IOException
   { //attach the file to FileInputStream
      FileInputStream fin = new FileInputStream ("myfile");
      //read data from FileInputStream and display it on the monitor
      int ch;
      while ( (ch = fin.read() ) != -1 )
         System.out.print ((char) ch);
      fin.close ();
   }
}
```



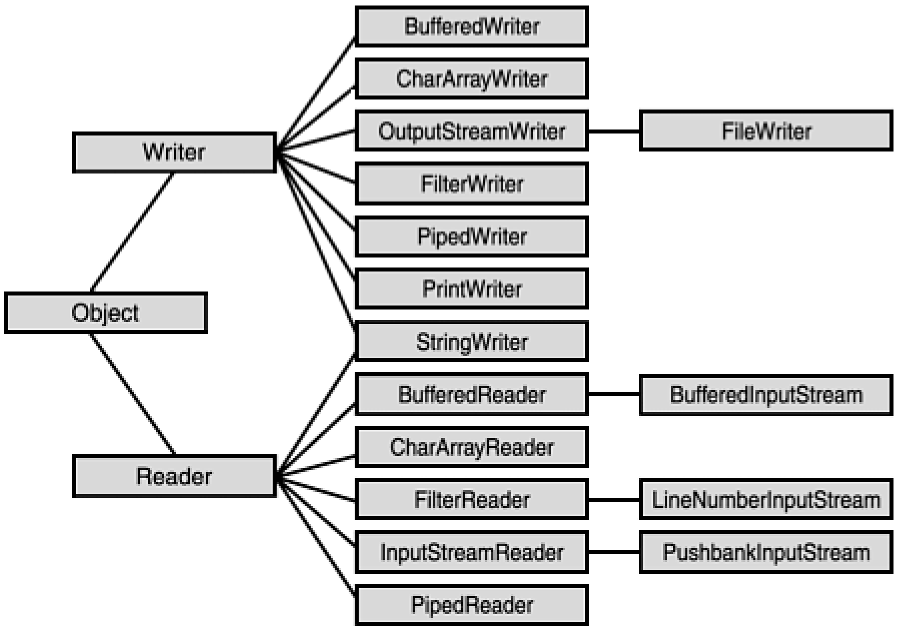
```
D:\JQR>javac Read1.java

D:\JQR>java Read1
I am writing first line
I am writing second line
This is new line in my file

D:\JQR>
```

// Java program to improve the efficiency while reading data from a file using BufferedInputStream.

//Reading a text file using byte stream classes

```
import java.io.*;
class Read2
{
   public static void main(String args[]) throws IOException
   {
      //attach the file to FileInputStream
      FileInputStream fin = new FileInputStream ("myfile");
      BufferedInputStream bin = new BufferedInputStream (fin);
      //read data from FileInputStream and display it on the monitor
      int ch;
      while ( (ch = bin.read() ) != -1 )
         System.out.print ( (char) ch);
      fin.close ();
   }
}
```

CharcterStream Classes



// Java program to create a text file using character or text stream classes

```
import java.io.*;
class Create3
{
   public static void main(String args[]) throws IOException
   {
   String str = "This is an Institute" + "\n You are a student"; // take a String
   //Connect a file to FileWriter
   FileWriter fw = new FileWriter ("textfile");
   //read chars from str and send to fw
   for (int i = 0; i<str.length () ; i++)
      fw.write (str.charAt (i) );
   fw.close ();
   }
}
```

// Java program to read a text file using character or text stream classes.

```
import java.io.*;
class Read3
{
   public static void main(String args[]) throws IOException
   {
          //attach file to FileReader
           FileReader fr = new FileReader ("textfile");
```

```
            //read data from fr and display
            int ch;
            while ((ch = fr.read()) != -1)
               System.out.print((char)ch);
            //close the file
            fr.close ();
   }
}
```

RandomAccessFile

- ➔ In java, the **java.io** package has a built-in class **RandomAccessFile** that enables a file to beaccessed randomly.
- ➔ The RandomAccessFile class has several methods used to move the cursor position in a file.
- ➔ A random access file behaves like a large array of bytes stored in a file.

RandomAccessFile Constructors

RandomAccessFile(File fileName, String mode): It creates a random access file stream to read from,and optionally to write to, the file specified by the File argument.

RandomAccessFile(String fileName, String mode): It creates a random access file stream to readfrom, and optionally to write to, a file with the specified fileName.

Access Modes

Using the RandomAccessFile, a file may created in th following modes.

- **r** - Creates the file with read mode; Calling write methods will result in an IOException.
- **rw** - Creates the file with read and write mode.
- **rwd** - Creates the file with read and write mode - synchronously. All updates to file content iswritten to the disk synchronously.
- **rws** - Creates the file with read and write mode - synchronously. All updates to file content ormeta data is written to the disk synchronously.

RandomAccessFile methods: Some of the methods are described below.

int read(): It reads byte of data from a file. The byte is returned as an integer in the range 0-255.
int read(byte[] b, int offset, int len): It reads bytes initialising from offset position upto b.length from the buffer.

String readUTF(): It reads in a string from the file.
void write(int b): It writes the specified byte to the file from the current cursor position.

void writeDouble(double v): It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high bytefirst.

// Java program to demonstrate class RandomAccessFile.

```
import java.io.*;
public class RandomAccessFileDemo
{
   public static void main(String[] args)
```

```
{
    try
    {
        double d = 1.5;
        float f = 14.56f;

        // Creating a new RandomAccessFile - "F2"
        RandomAccessFile f= new RandomAccessFile("C:\\prog\\myfile.txt", "rw");

        // Writing to file
        f.writeUTF("Hello, Good Morning!");

        // File Pointer at index position - 0
        f.seek(0);

        // read() method :
        System.out.println("Use of read() method : " + f.read());

        f.seek(0);

        byte[] b = {1, 2, 3};

        // Use of .read(byte[] b) method :
        System.out.println("Use of .read(byte[] b) : " + f.read(b));

        // readBoolean() method :
        System.out.println("Use of readBoolean() : " + f.readBoolean());

        // readByte() method :
        System.out.println("Use of readByte() : " + f.readByte());

        f.writeChar('c');
        f.seek(0);

        // readChar() :
            System.out.println("Use of readChar() : " + f.readChar());

            f.seek(0);
            f.writeDouble(d);
            f.seek(0);

            // read double
            System.out.println("Use of readDouble() : " + f.readDouble());

            f.seek(0);
            f.writeFloat(f);
            f.seek(0);

            // readFloat() :
            System.out.println("Use of readFloat() : " + f.readFloat());

            f.seek(0);
            // Create array upto geek.length
            byte[] arr = new byte[(int) f.length()];
            // readFully() :
            f.readFully(arr);

            String str1 = new String(arr);
            System.out.println("Use of readFully() : " + str1);

            f.seek(0);

            // readFully(byte[] b, int off, int len) :
            f.readFully(arr, 0, 8);

            String str2 = new String(arr);
            System.out.println("Use of readFully(byte[] b, int off, int len) : " + str2);
        }
        catch (IOException ex)
        {
            System.out.println("Something went Wrong");
            ex.printStackTrace();
        }
    }
}
```