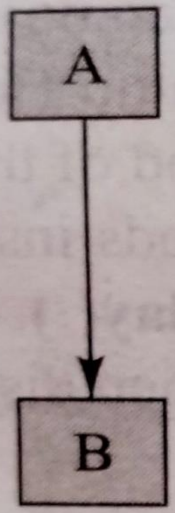
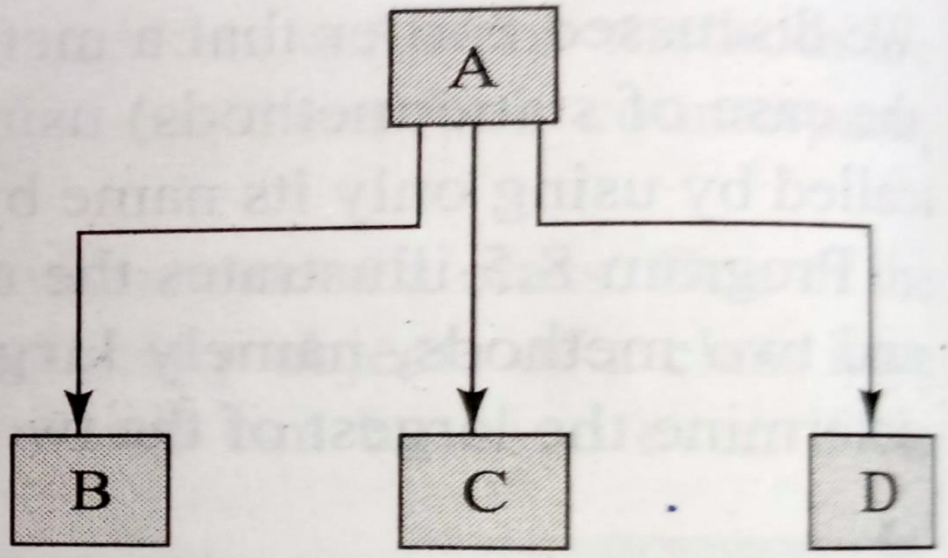
Inheritance

- Reusability
- Deriving a new class from an old one
- Inheritance allows subclasses to inherit all the variables and methods of their parent classes.
- Forms:

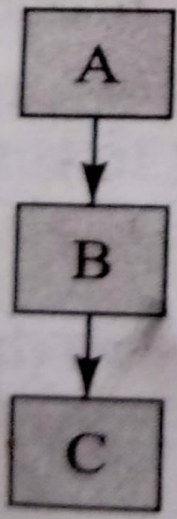
➢ Single inheritance
➢ Multiple inheritance
➢ Hierarchical inheritance
➢ Multilevel inheritance



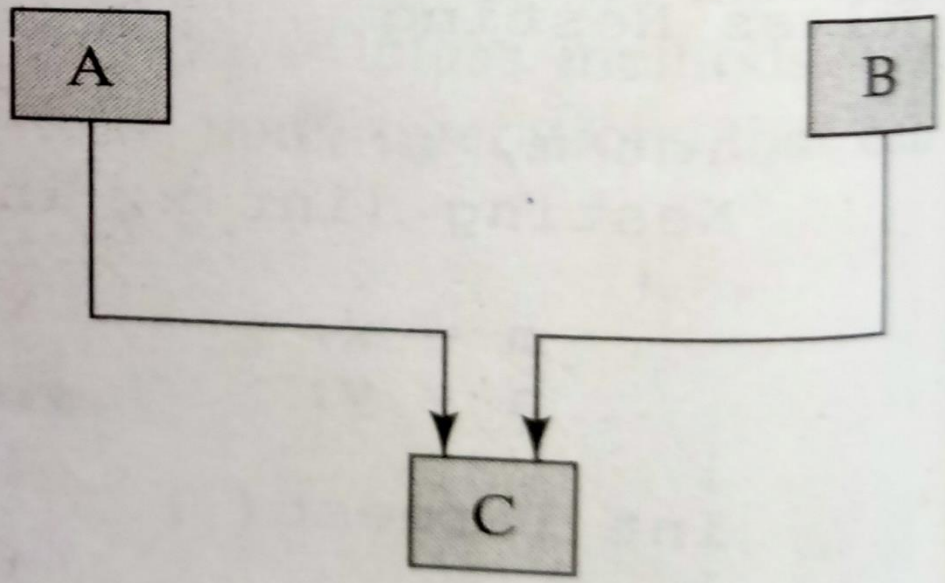
(a) Single inheritance



(b) Hierarchical inheritance



(c) Multilevel inheritance



(d) Multiple inheritance

Defining a subclass

```
class subclassname extends superclassname
{
   variable declaration;
    methods declaration;
}
```

- The keyword extends signifies that the properties of the superclassname are extended to subclassname.
- The subclass will now contain its own variables and methods as well those of the superclass.

Subclass Constructor

- A subclass constructor is used to construct the instance variables of both the subclass and the superclass .
- The subclass constructor uses the keyword super to invoke the constructor method of the superclass.
- The keyword **super** is used provided:

➢ Super may only be used within a subclass constructor method.

➢ The call to superclass constructor must appear as the first statement within the subclass constructor.

➢ The parameters in the super class must match the order and type of the instance variable declared in the superclass.

Multilevel Inheritance

```
Class A
{
  -----
  ------
}
Class B extends A
{
------
------
}
Class C extends B
{
-----
------
}
```

Overriding Methods

- A method defined in a super class is inherited by its subclass and is used by the objects created by the subclass.
- If we want an object to respond to the same method but have different behaviour when that method is called. We override the method defined in the superclass.
- This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a method in the superclass.
- When that method is called, the method defined in the subclass is invoked and executed instead of the one in the superclass.
- This is known as **method overriding.**

Final variables and methods

- All methods and variables can be overridden by default in subclasses.
- If we wish to prevent the subclasses from overriding the members of the superclass, we can declare them as final using the keyword **final** as a modifier.

final int SIZE=100;
final void showstatus();

Final classes

- Sometimes we may like to prevent a class being further subclasses for security reasons.
- A class that cannot be subclassed is called a final class.

final class abc{……..}
final class pqr extends someclass{……..}

- Any attempt to inherit these classes will cause an error and the compiler will not allow it.

Abstract methods and classes

- To indicate that a method must always be redefined in a subclass, thus making overriding compulsory.
- This is done using the modifier keyword **abstract** in the method definition.

```
abstract class shape
{
……
    abstract void draw();
……
}
```

- When a class contains one or more abstract methods ,it should be declared **abstract**

Dynamic Method Dispatch

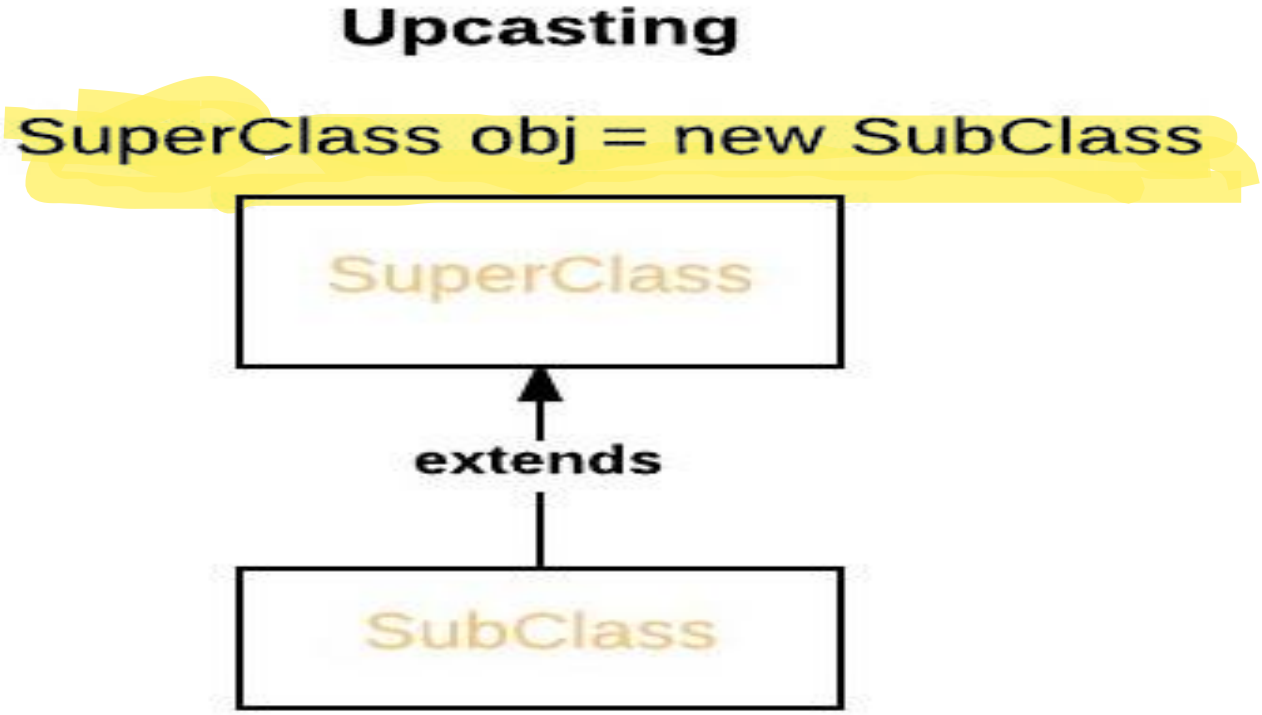
- Method overriding is one of the ways in which Java supports Runtime Polymorphism.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.

Dynamic Method Dispatch

- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

Dynamic Method Dispatch

- Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.



Program

Create an abstract class SalaryAccount with an abstract method bank_name() and normal method status_employee(). Create a class PermEmployee with 3 constructors default constructor, parametrized constructor with 2 arguments name, id and another parametrized constructor with 3 arguments name, id and salary. Class PermEmployee should extend SalaryAccount and define bank_name() displaying name of bank of Permanent employee salary account is present. Override status_employee().Create another class TempEmployee with 1 constructor which initializes name, id and salary also overrides bank_name() and status_employee(). Display the details of 2 classes of employee using constructor overloding and dynamic method dispatch.

Interfaces: Multiple Inheritance

- Java does not support multiple inheritance.
- Java provides interfaces to support multiple inheritance.
- Although a java class cannot be a subclass of more than one superclass, it can implement more than one interface, thereby enabling us to create classes that build upon other classes without the problems created by multiple inheritance.

Interfaces: Multiple Inheritance

- Defining Interfaces:

- An interface is basically a kind of class.
- Like classes , interfaces contain methods and variables but with a major difference.
- Interfaces define only abstract methods and final fields.
- Interfaces do not specify any code to implement these methods and data fields contains only constants.
- Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

Interfaces: Multiple Inheritance

- Syntax for defining an Interface:

```
interface InterfaceName
{
        variable declaration;
         methods declaration;
}
```

- **static final type variableName=value;**
  All variables are declared as constants.
- **return-type methodName1(parameter_list);**

Interfaces: Multiple Inheritance

- Example of an Interface:

```
interface Item
{
    static final int code=1001;
    static final String name="Fan";
    void display();
}

interface Area
{
    static final float pi=3.14;
    void show();
    float compute(float x, float y);
}
```

Interfaces: Multiple Inheritance

- Extending Interfaces:

- Like classes, interfaces can also be extended.
- An interface can be subinterfaced from other interfaces.
- The new subinterface will inherit all the members of the superinterface.

```
interface name2 extends name1
{
    body of name2
}
```

Interfaces: Multiple Inheritance

- Extending Interfaces:

```
interface ItemConstants
{
    int code=1001;
    String name=“Fan”;
}

interface Item extends ItemConstants
{
        void display();
}
```

Interfaces: Multiple Inheritance

Implementing Interfaces:

- Interfaces are used as “superclasses” whose properties are inherited by classes.
- It is therefore necessary to create a class that inherits the given interface.

class classname **implements** interfacename
{
    body of classname
}

Interfaces: Multiple Inheritance

Implementing Interfaces:

- More general form of implementation:

class classname extends superclass implements interface1,interface2,…..
{
  body of classname
}

Interfaces: Multiple Inheritance

- example

Difference in Interface and class

Sr No	Class	Interface
1.	The members of a class can be constant or variables.	The members of an interface are always declared as constant that is their values are final.
2.	The class definition can contain the code for each of its methods. That is, the methods can be abstract or non-abstract.	The methods in an interface are abstract in nature i.e. there is no code associated with them. It is later defined by the class that implements the interface.
3.	It can be instantiated by declaring objects.	It cannot be used to declare objects . It can only be inherited by a class.
4.	It can use various access specifiers like public, private or protected.	It can only use public access specifier.

Interfaces: Multiple Inheritance

Create a class Student which defines and accepts roll_no of a student.

Create one more class Test which extends Student class which defines and accepts students marks in two tests T1 and T2.

Create an interface sports, which has a variable name sportWt=6.0F and void putwt() method.

Create class Result which extends Test class and implements Sports interface to display total marks of a student including sports marks if a student is a sports person.

Create one more class which has main method, which will get roll no, test1, test2 marks and displays the average marks including sports marks if applicable in the result.