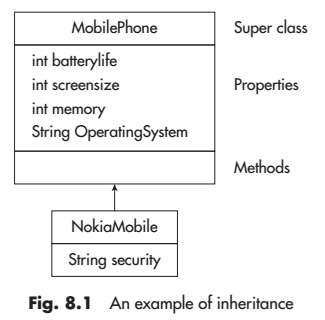
**Introduction, Process of Inheritance**

It is the mechanism by which a class can acquire properties and methods of another class. Using inheritance, an already tested and debugged class program can be reused for some other application. This class is often termed as base class or super class. Another class that is created in this application inherits the fields and methods of base class and extends the existing class to include special characteristics of objects required in the program. In this connection, the following terminologies are often used.

**Super class:** This is the existing class from which another class, that is, the subclass is generally derived. In C++, the super class is called the base class. In Java, several derived classes can have the same super class. However, multiple super classes are not allowed as it may lead to an ambiguous situation.

**Subclass**: A class that is derived from another class is called subclass. In C++, a subclass is called derived class. In Java, a subclass can have only one super class. This restriction is not present in C++, which supports several base classes to a derived class.

Let us consider an example of inheritance in which a super class Mobile Phone is created. Different types of mobile phones have different features and properties. However, some of the properties are general and common to all. This may include battery, screen size, memory, operating system, and so on. The subclass of this super class can be created and it can be any type of mobile phone and will inherit the common features of the super class. For instance, class Best Mobile is a subclass that will inherit all the features of class Mobile Phone. Some additional features and attributes can also be defined in this subclass that makes it different from the other subclasses.



Benefits of Inheritance

1. It allows the reuse of already developed and debugged class program without any modification. Further, the subclass inherits it as it is; besides, the subclass can have its own variables and methods to take care of special characteristics of sub-section of objects.

2. The super class is more general in its scope, whereas a subclass is more specialized. Thus, it allows a number of subclasses to fulfil the needs of several subgroups.

3. A large program may be divided into suitable classes and subclasses that may be developed by separate teams of programmers and vice versa.

4. The process of inheritance may or may not stop with one derived class. In fact, another class may be derived from a previously derived class. For instance, class B is derived from class A and class C is derived from class B (see Fig. 8.2). In that case, class B is the derived class with respect to class A and is the super class with respect to class C. Class A may also be called indirect super class of class C and class C may also be called indirect subclass of class A.

**Process of Inheritance**

In simple words, inheritance means deriving some characteristics from something that is generic. In the context of Java, it implies deriving a new class from an existing old class, that is, the super class. We may create a super class that describes general characteristics of a class of objects. A subset of these objects may have characteristics different from others. There are two ways of dealing with this problem. Either, make a separate class for the subset to include all the characteristics or, to have another class that inherits the existing class, extend this class to include the special characteristics. For example, we may have a general class of vehicles. We may have other classes that extend this class to include specific characteristics of vehicles such as cars, trucks, and scooters, as illustrated in Fig.

A diagram of a vehicle

Description automatically generated

The class that is inherited is called a superclass. The class that is inheriting the properties is called a subclass. Therefore the subclass is the specialized version of the superclass. The subclass inherits all the instance variables and methods using the extends keyword, and adds its code. Superclass is also known as Parent class, and Base class. The subclass is also known as the Child class and Derived class.

**Aggregation** is the process of making an object by combining several other objects. The behaviour of the bigger object is defined by the behaviour of its component objects. For example, cars contain several other components such as engines, clutches, breaks, starters, etc.

**Substitutability and Subtyping**

The concept of substitutability is fundamental to OOP. Under this concept, the type given to variable in the declaration statement may not match with the type of the value the variable is holding. Let us assume that we have two classes A and B, such that class B is subclass of class A. Then, according to this concept, we can substitute the instances of class B for instance of class A, irrespective of any situation.

**subtype**

The term subtype is used to describe a subclass relationship in which the principle of substitution is maintained.

According to this concept, we can say that type B is a subtype of A if

1. B’s specification implies A’s specification. In other words, any object that satisfies B’s specification also satisfies A’s specification.

2. for each method defined in super type, the subtype must have a corresponding method.

Furthermore, for type B to be a subtype of A, the relationship must be declared using either keyword, extends or implements. The subtype may have additional methods that may not be defined in super type. In addition, for each method in subtype that corresponds to super type, it must have the following:

1. Same types of arguments

2. Same type of result

3. No declared exceptions

In symbolic representation, we can write type B as a subtype of A as B v A. For instance, we have types int and float. It can be seen that int is a subtype of float. This can be represented in symbolic form as int v float. In Java, subtyping occurs when inheritance is implemented. In most of the cases, a subclass is also a subtype. For instance, consider the previous example in which class Vehicle is the super class and class Four Wheelers is the derived class that will reuse some of the codes defined in super class.

The following types of inheritances are supported by Java.

1. Single inheritance

2. Multilevel inheritance

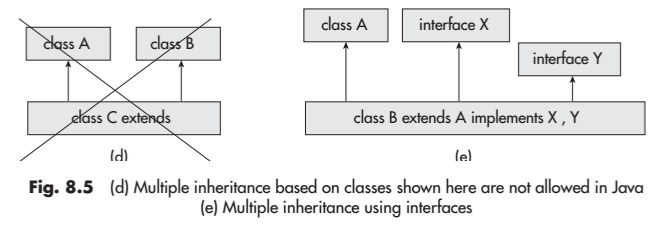
3. Hierarchical inheritance

4. Multiple inheritance using interfaces

1. Single inheritance: It is the simple type of inheritance. In this, a class extends another one class only. As shown in Fig. 8.5 (a), class B extends only one class, that is, class A. Here, class B is the sub class and class A is the super class.
2. Multilevel inheritance: In this type of inheritance, a derived class inherits a parent or super class; the derived class also acts as the parent class to other class. As shown in Fig. 8.5 (b), class C inherits from its immediate super class, that is, class B and also from class A. As a result, class C possesses all the properties and methods of class A as well as class B and also has its own. In Java, there is no limit to this chain of inheritance.
3. Hierarchical inheritance: In this type of inheritance, one class is inherited by many sub classes. As can be seen in Fig. 8.5 (c), class B, D and C inherits from class A.
4. Multiple inheritance: In this, a class is extending more than one class. Java does not support multiple inheritance. This implies that a class cannot extend more than one class. From the figure, it is seen that class C extends both the class A and class B. Suppose there is a method in class A. This method is overridden in class B and class C in their own way. Since class C extends both the classes A and B. So, if class C uses the same method, then there will be ambiguity as which method is called. This is covered in detail in section 8.11. The first three types are illustrated in Fig. 8.5(a)–(c). It should be noted that Java does not support multiple inheritance in which a class extends a number of classes as is the case in C++.

A diagram of a class

Description automatically generated



Single Inheritance

class Box

{

double width, height, depth;

Box(double w, double h, double d)

{

width = w; height = h; depth = d;

}

}

class BoxVolume extends Box

{

BoxVolume(double w,double h,double d)

{

super(w, h, d); //calling the super class constructor

}

void boxVolume()

{

double v = width \* height \* depth;

System.out.println("The volume of the Box is " + v);

}

}

class BoxTest

{

public static void main(String args[])

{

BoxVolume bv = new BoxVolume(12.3, 13.2, 14.3);

bv.boxVolume();

}

}

Output

A black screen with white text

Description automatically generated