



A Thorough Study of Different Types of Inheritance using Object Oriented Programming with JAVA

Shyamapriya Chowdhury
Department of Information Technology,
Narula Institute of Technology, India

Sudip Chatterjee
Department of Computer Science,
Ideal Institute of Technology, India

Abstract— *The concept of inheritance is quite similar to family tree in our daily life. Creation of a new class from an existing one is called inheritance. Without modifying the previous data new features can be added in a class. The newly created class is called child class and from which it is created is known as parent class. Just like human being, child class can automatically access all the properties of Parent class and new methods can also be entered in child class. Concept of reusability is directly supported by JAVA using this inheritance.*

Keywords — *Child class, parent class, redundancy, diamond problem, inheritance*

I. INTRODUCTION

There are three basic properties of object oriented programming language. Inheritance is one of the properties of object orientation by which properties of one object can be directly accessed by another object.[1] The class which is created after accessing properties is known as subclass or derived class or child class and the class from which subclass is created is called super class or parent class or base class. Through the property of inheritance redundancy is reduced. Subclass can directly access all the properties of super class, it doesn't need to create it again. This reusing of previous code also saves time. For example, cricket is an outdoor game which is again a type of sports.[3]

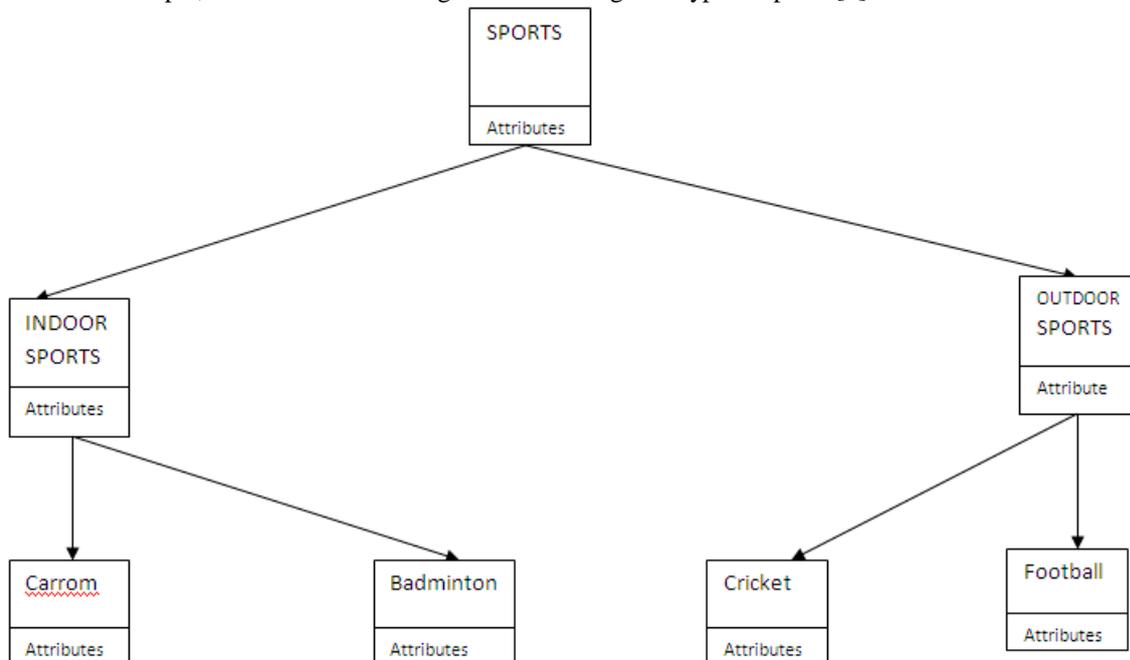


Fig. 1 Inheritance

II. PAGER LAYOUT

So, from Fig. 1 it is clear that new classes can be made from the existing one. New additional features can be inserted in the child class without modifying the previous features. Just like in our real life, child can use all the things of parents, here also, subclass can access all the properties (methods and variables) of superclass. In this way redundancy is totally eliminated.

Inheritance is of four types:

- Single inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Multiple inheritance

III. SINGLE INHERITANCE

This type of inheritance contains only one super class and one subclass. In Fig. 2 child class B is created from parent class A.

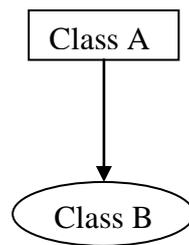


Fig. 2 Single Inheritance

In practical life, suppose a person is father of one child. Here is a short piece of code for demonstrating single inheritance:-

```
Class parent
{
public void access()
{
System.out.println("Son can access my all properties.");
}
}
Class son extends parent
{
public void inherit()
{
super.access();
System.out.println("I have inherited all the properties of my parent");
}
}
class singleinheritance
{
public static void main(String args[])
{
Son s = new son();
s.inherit();
}
}
```

IV. HIERARCHICAL INHERITANCE

In this type of inheritance more than one sub classes are created from one super class.[6] It means that properties of super class will be inherited at least two or more than two sub classes. In Fig.3, A is the super class and B, C, D are the sub classes.

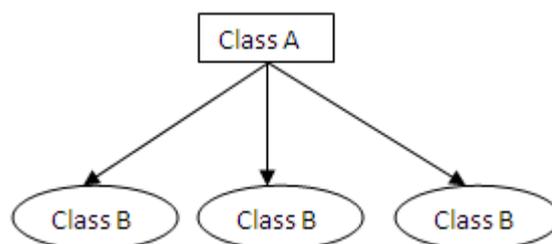


Fig. 3 Hierarchical Inheritance

In real life, suppose a person has two sons, will be the example of hierarchical inheritance. Here is a piece of code to demonstrate the hierarchical inheritance problem in the next page:

```
class parent
{
public void access()
{
System.out.print("My two Sons can access all my properties");
}
}
class elderson extends parent
```

```
{
public void inherit()
{
super.access();
System.out.print("I am inheriting all my parent's properties.");
}
}
class youngerson extends parent
{
public void acquire()
{
super.access();
System.out.print("I am also inheriting all my parent's properties.");
}
}
class hierarchicalinheritance
{
public static void main(String arg[])
{
elderson es=new elderson();
youngerson ys=new youngerson();
es.inherit();
ys.acquire();
}
}
```

V. MULTILEVEL INHERITANCE

In this type of inheritance, sub class is created from a subclass. Presence of intermediate class is mandatory in this type of inheritance.[4] Intermediate class is a class which can act as both super class and sub class. In Fig.4, B is the intermediate class, which is super class for C and sub class for A.

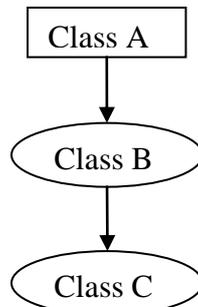


Fig. 4 Multilevel Inheritance

In real life, grandfather-father-son is an example of multilevel inheritance. Here is a piece of code to demonstrate the multilevel inheritance problem:

```
class grandfather
{
public void access()
{
System.out.println("My son and his son will access my all properties");
}
}
class father extends grandfather
{
public void inherit()
{
super.access();
System.out.print("I am accessing all my father's properties");
}
}
class son extends father
{
public void acquire()
{
```

```
super.inherit();
System.out.print("I am accessing my father's as well as grandfather's properties");
}
}
class multilevelinheritance
{
public static void main(String arg[])
{
Son s=new son();
s.acquire();
}
}
```

VI. MULTIPLE INHERITANCE

If one sub class is created by inheriting the properties of more than one super class then this type of inheritance is called multiple inheritance. In this type of inheritance, more than one super class is present and one subclass will access the properties of these subclasses.[5] In Fig 5, class A and class B represents super class and class C is the sub class.

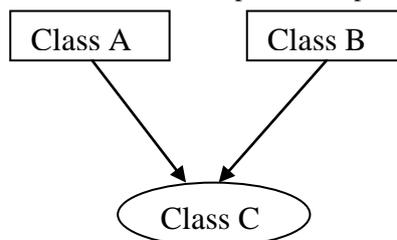


Fig. 5 Multiple Inheritance

In our practical life, generally a child acquires all the properties of their parents. This is an example of multiple inheritance.

But some problem arises with this type of inheritance due to which JAVA can't support multiple inheritance directly. Suppose two same name methods (inherit() in the following program) are present in the super classes father and mother. The child class is accessing this method.[7] The problem is JAVA compiler cant understand which version of inherit() to call, whether of father class or mother class. That's why JAVA can't directly support multiple inheritance. Though interface is the method by which JAVA can partially support the concept of multiple inheritance.

```
class father
{
void inherit()
{
System.out.println("My sibling will inherit my properties");
}
}
class mother
{
void inherit()
{
System.out.println("My sibling will inherit my properties also");
}
}
class son extends father, mother
{
void inherit() //method from which class, class father or class mother
{
}
}
```

VII. CONCLUSIONS

Reusability is a main feature of object orientation programming language. In JAVA this feature is supported by the properties of inheritance. Use of non redundant data is also a feature of inheritance property. The property of accessing all data and methods is called inheritance. As child class acquires all the properties of parent class, it doesn't need to define all the previously defined things of super class, thus redundancy stops and reusability increases.

REFERENCES

- [1] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M.: Pattern-oriented Software Architecture: A System of Patterns. Wiley 1996.
- [2] J. Campione M., Walrath K.: The Java Tutorial, 2nd edition, Addison-Wesley, 1998.
- [3] M. Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley 1995.
- [4] J.: A Systematic Approach to Multiple Inheritance Implementation. SIGPLAN Notices 28 (4): 61-66
- [5] Mössenböck H.: Objektorientierte Programmierung in Oberon-2. Springer-Verlag 1993.
- [6] Pree W.: Design Patterns for Object-Oriented Software Development. Addison-Wesley 1995.
- [7] Stroustrup B.: Multiple Inheritance for C++. Proceedings EUUG Spring Conference, Helsinki, May 1989.