Article can be accessed online at http://www.publishingindia.com

A Complete Inter-Class Sharing During the Inheritance to Enhance Reusability of Public Data and Their Access Control Using Dominance

Jyoti Lakhani*, Dharmesh Harwani**

Abstract

Inheritance is used for reusability in an object oriented programming language. The complete reusability is not possible with the simple inheritance process. On inheriting parent class, the child does not get a complete access to the inherited data. Only an instance of the values of inherited data has been accessible by the child class and it is true even for the public data. The present communication is a concept paper in which a special inheritance method has been conceptualised. This proposed method is called backward accessibility inheritance. Once inherited, the public data item of the parent class can be shared by the child class in all sense. This way the inherited data item can be truly reused by child class in terms of memory space, name and value. To control the accessibility of the data during the backward accessibility inheritance, concept of dominance has also been introduced.

Keywords: Inheritance, Object Oriented Programming, Backward Accessibility, Dominance

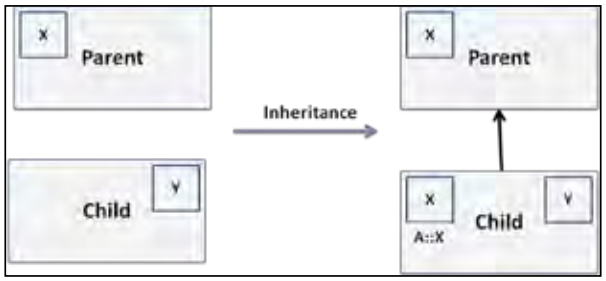
Introduction

The clear motive of opting object oriented programming (OOP) tool by a developer is reusability (en.wikipedia.org). Inheritance is the most common tool for implementing reusability (en.wikipedia.org). It is one of the pillars of object oriented programming. In scientific terminology, inheritance is the process of transferring genetic material from the parent to the child. This is the process through which a child has got the characteristics and behaviour similar to his parent. According to Rambaugh (1991), inheritance is used to establish an 'is-a' relationship between two classes. A class is nothing but a template for a real world entity. A real world entity can be generated by making instances of previously defined class. This real world entity is nothing else but an object. An object has some characteristics and behaviour. In object oriented programming language, a characteristic of object is denoted by member variable of the class and behaviour is denoted by member function of that class. So in the process of inheritance both characteristic (member variable) as well as behaviour (member function) of a class have been transferred to the other class. The sole emphasis of this communication is on inheritance of the characteristics or say member variables. This process of transferring characteristics from parent to the child is called as Inheritance. A class which transfer its contents to the other class is called parent class and a class, which acquire characteristics and behaviour from parent is called as child class (Rambaugh, 1991). The process of inheritance is shown in Fig. 1. Both the parent and child classes have their own individual identification. Each class has its own data-items, such as parent class has data-item X and child class has data-item Y. Let us suppose that the data-item X in parent class is public. Here, the term *public* is a quantifier indicating that X can be seen and get transferred from parent class to the child class. It is shown that after inheritance the parent class has data-item X and child class has data-Items X and Y both. This is an example of a simple inheritance.

* Assistant Professor, Department of Computer Science, Maharaja Ganga Singh University, Bikaner, Rajasthan, India. Email: jyotilakhanimgsu@gmail.com

** Assistant Professor, Department of Microbiology, Maharaja Ganga Singh University, Bikaner, Rajasthan, India. Email: dharmesh_harwani@hotmail.com

Fig. 1: Simple Inheritance Process



Inheritance in C++

Program 1:

```
#include <iostream.h>
#include<conio.h>
class parent
{
  public:
  int x;
  parent()
  {
    x=10;
  }
};
class child: public parent
{
  public:
  int y;
  child()
  {
    y=20;
  }
};

int main(void)
{
   child c;
            cout<<"\nc.x\t"<<c.
x<<endl;
  cout<<"c.y\t"<<c.y<<endl;
     cout<<"c.parent::x\t"<<c.
parent::x;
   getch();
   return 0;
}
```

Considering above example in C++ programming language, code snippet for a simple inheritance process is given in Program 1. In the code there are two classes; one parent class and one child class. The parent class has an integer variable x with value 10 and the child class is also having an integer variable y with value 20. By inheriting parent class, the child class has received data variable x from the parent class and now has two data variables x and y. In the main(), an object c for the child class has been declared. In the next statement there is a call for printing value of the x variable of the child class using c.x expression. The next statement prints value of y variable of the child class using c.y expression. These two statements are simple printing statements and both are accessing child class data variables using child class object. The first two lines in the output (values 10 and 20) are the values of x and y respectively. But the next statement in main() is not as simple. By c.parent::x expression in the statement, we try to access the parent class data variable through the child class. The third line in output indicates that we can access a variable in parent even through child class after inheritance. But it is not true. The child kept a mirror reference of parents' data_ items. When we call child's object to print parents data_ item x, it actully uses that copy of the data. The situation will be clear on running the code given in Program 2.

Program 2:

```
#include <iostream.h>
#include<conio.h>
class parent
{
  public:
  int x;
  parent()
  {
    x=10;
  }
};

class child: public parent
{
  public:
  int y;
  child()
  {
    y=20;
  }
};

int main(void)
{
   child c;
  cout<<"\nc.x\t"<<c.x<<endl;
  cout<<"c.y\t"<<c.y<<endl;
        cout<<"c.parent::x\t"<<c.
parent::x;
   getch();
   return 0;
}
```

The output of the code will be-

c.x 70

c.y 20

c.parent::x 70

p.x 10

It is clear here by output that object c of class child is not actually accessing the parents' data_item x. If it would, the output of p.x should be 70. It is accessing a copy of parents' data_item x. We can understand this by another example shown in Fig. 2.

Fig. 2: Showing no Inter Class Sharing of Public Data During Simple Inheritance

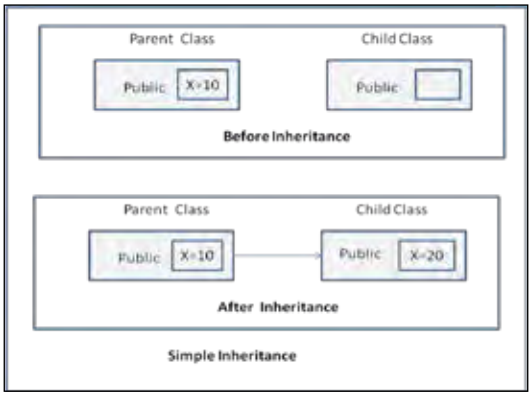
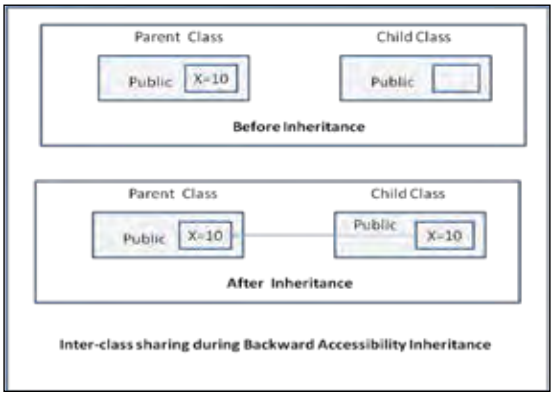


Fig. 2 will further explain the process of simple inheritance from a different angle. As shown in the example, before inheritance the parent class has a public data item X and child class does not have any public data item. After inheritance the child also gets an instance of X. Suppose the value 20 has been assigned to the X in the child class. Fig. 2 is a conceptual image of this situation where parent has X =10 and the child class has X=20. It is clear that data item X in child class is not the same to X in parent class but is a replica of that. It is also clear that changes in values of X in the child class will not be reflected in the parent class. This is not the complete reusability but rather it is partial reusability of data item X. If we are expecting a total reusability of data_items, there should be real access to the data_items from child class to the parent class to implement the complete reusability.

Backward Accessibility Inheritance

In our proposed approach called Backward Accessibility Inheritance, once a child has got permission to reuse parents' data, it should be eligible to change parents' data. The focus of the proposed communication is on public data of the parent class only. After inheritance, parent and child could share the public data item of the parent class. During the inheritance process, a data accessibility link between the parent and the child class has been established. After inheritance a child class could refer the parent class to access its public data directly. If the backward accessibility inheritance has been used in the example shown in Fig. 1, the last output case (p.x), should be 70 in Program 1. It means, once inherited, the object of child class can directly access the parents' public data member x. Consider Fig. 3 which shows the inter-class sharing of public data during the backward accessibility inheritance. Before inheritance, parent class has a public data item X=10 but the child class does not have any public data. After backward accessibility inheritance, the child class has received backward accessibility for public data item X of the parent class. In backward accessibility the child class can directly access the public data item X of the parent class using a backward accessibility pointer and share the address of the same with parent class. If the value of shared data item X has been changed in the child class, this change will be reflected in the parent class.

Fig. 3: Showing Inter Class Sharing of Public Data During Backward Accessibility Inheritance



So this communication is a proposal to amend the process of inheritance in a novel conduct. The newly proposed approach is a special kind of inheritance to resolve the complete reusability issue during the process of inheritance of public data of a class. It is suggested that there should be a choice for the programmer between simple or backward accessibility inheritances.

Access Control of Public Data in Backward Accessibility Inheritance Using Dominance

It is recommended that there should be a conduct of dominance law during the backward accessibility inheritance process to control the accessibility of the public data. The concept of dominance was given by the father of modern genetics "Gregor Mendel". He proposed one of the best described natural inheritance phenomenon (Mendel, 1965). Mendel was a German-speaking Silesian scientist and Augustinian friar who achieved posthumous recognition as the founder of genetics. He gave the law of inheritance which holds that one allele (one member of a pair of genes occupying a specific place on a chromosome)

Fig. 4: Backward Accessibility with Same Dominance of Parent and Child Class (Sharing Mode)

Fig. 5: Backward Accessibility with Higher Dominance of Parent Class

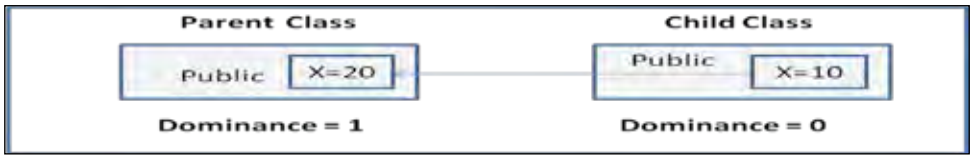
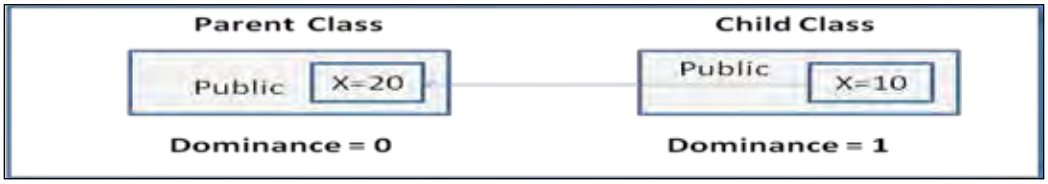


Fig. 6: Backward Accessibility with Higher Dominance of Child Class



either from male or female parent, from the pair of alleles coding for a particular trait (an observable, phenotypic character) is expressed whereas the other is unexpressed in a child. The allele expressed for a particular trait is regarded as the dominant (Mendel, 1965; Blumberg, 1997) whereas the other unexpressed allele is considered recessive (Mendel, Harwani, & Lakhani, 2015). In the proposed approach once the child class has inherited the parent class, the public and protected data items have been co-controlled by parent as well as child. If it is required that data item should be under controlled by only one class then one class should be given dominance over the other class. The dominance could be given to either class without any partiality and as per the requirement. Programmer should have privilege to set the dominance. By default dominance should be in 1:1 ratio means it should equal for both classes so the data will be in shared mode between parent and child (Fig. 4).

If one of the classes is dominated over the other class then the data will be under control and accessible by the dominant class only. Consider Fig. 5, parent class has dominance =1 and child class has dominance =0. So the parent class has the control over the shared data item X hence the actual value of the X will be 20.

In Fig. 6, the child class is dominant over parent class hence it has control over the shared data item X. So the actual value of the X will be 10 after inheritance.

Conclusion

The present communication is an effort to raise the reusability ratio during the process of inheritance. In order to acquire complete reusability, a novel approach has been proposed in this paper. In the proposed method once the child class has inherited the parent class, it will have to have a shared control over the parents' data. In that case the child class object could reuse the public and protected data items in its real sense. Although not implemented and tested yet, this concept could be useful to provide complete reusability. It is also proposed that if complete reusability is not applicable in some case, it would be the programmers' court of conduct to implement this or not. The concept of dominance has been suggested to control reusability. The dominant class will have the solicit control over the data. The limitation of backward accessibility inheritance is that it is considering inheritance of characteristics only. Behavioural inheritance is not considered yet in this study and is a subject to research next.

References

Blumberg, R. B. (1997). Mendel Web, Edition 97.1, Retrieved from www.mendelweb.org.

Inheritance. Retrieved from http://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)

Mendel G. (1965). Centenary of Mendel's paper-Experiments in plant hybridization. *British Medical Journal*, 368-374.

Mendel, G., Harwani, D., & Lakhani, J. (2015). Disambiguation of multiple inheritance in C++ using biological law of genetics. *Academic Journal of Science*, 3(3), 361-371.

Object Oriented Programming. Retrieved from http://en.wikipedia.org/wiki/Object-oriented_programming.

Rambaugh, J. (1991). *Object oriented modeling and design*. Prentice Hall.