CS-1201 Object Oriented Programming

Inheritance

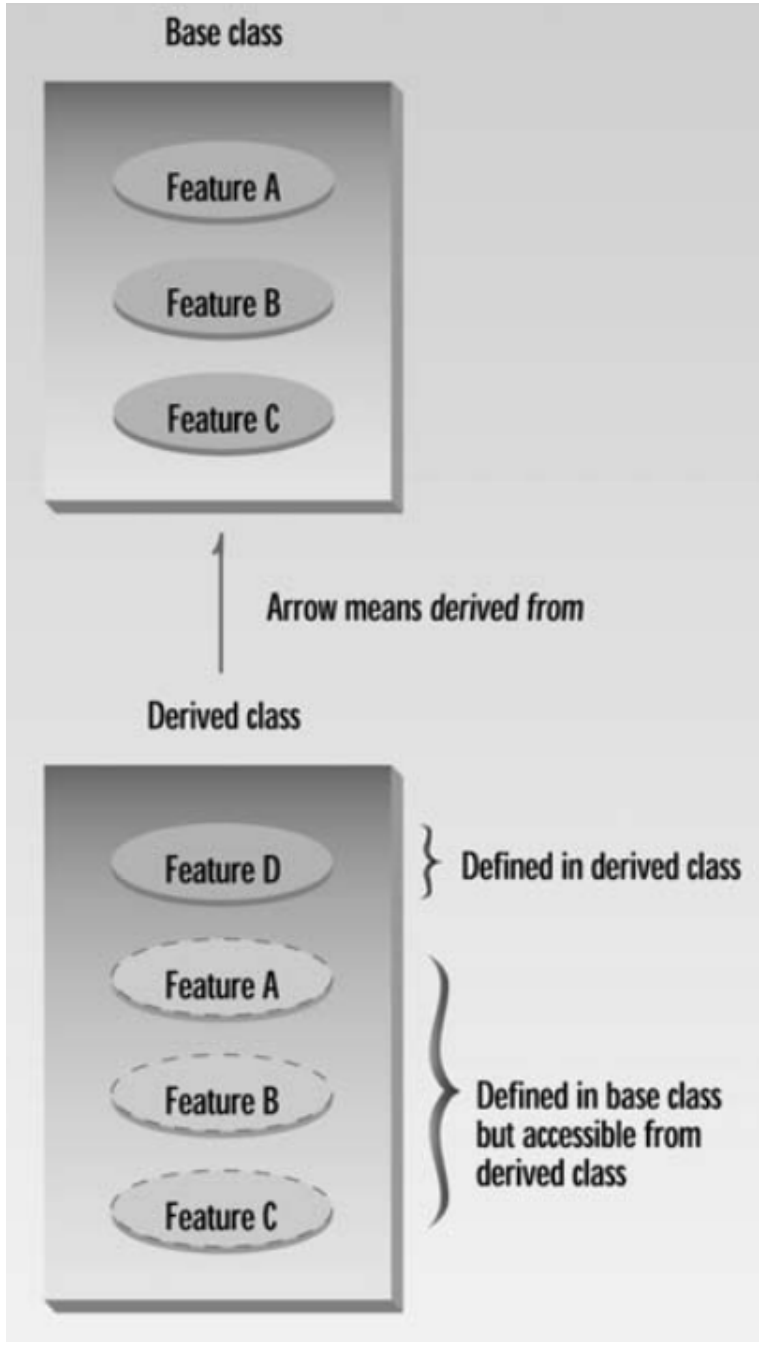
Arbish Akram

Department of Computer Science
Government College University

Introduction to Inheritance

Inheritance is one of the most powerful features of object-oriented programming, alongside classes themselves.

- Inheritance allows for the creation of new classes, known as **derived classes**, from existing or **base classes**.
- The derived class inherits all the capabilities of the base class and can add its own features or refinements.
- The base class remains unchanged by this process.


Base class
Feature A
Feature B
Feature C
Arrow means derived from
Derived class
Feature D
Feature A
Feature B
Feature C
Defined in derived class
Defined in base class but accessible from derived class

Example

```
1  class Animal {
2      // eat() function
3      // sleep() function
4  };
5
6  class Dog : public Animal {
7      // bark() function
8  };
```

Inhertiance Examples

Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount

Example

```
1  class Animal {   // base class
2      public:
3      void eat() {
4              cout << "I can eat!" << endl; }
5      void sleep() {
6          cout << "I can sleep!" << endl; }
7      };
8
9  class Dog : public Animal {   // derived class
10    public:
11     void bark() {
12         cout << "I can bark! Woof woof!!" << endl; }
13 };
14 int main() {
15     Dog dog1; // Create object of the Dog class
16     // Calling members of the base class
17     dog1.eat();
18     dog1.sleep();
19     // Calling member of the derived class
20     dog1.bark();
21     return 0;
22 }
```

Protected Access Specifier

- Member functions can access base class members if they are **public** or **protected**.
- **private** members cannot be accessed.
- Making a member **public** would allow it to be accessed by any function, eliminating data hiding benefits.
- **protected** members are accessible by member functions within its own class and **derived classes**.
- Not accessible from functions outside these classes (e.g., `main()`).

Inheritance and Acessibility

Access Specifier	*Accessible from Own Class*	*Accessible from Derived Class*	*Accessible from Objects Outside Class*
public	yes	yes	yes
protected	yes	yes	no
private	yes	no	no

Example

```
1  class Animal {
2     private:
3          string color;
4     protected:
5          string type;
6     public:
7          void eat() {
8              cout << "I can eat!" << endl;
9          }
10         void sleep() {
11             cout << "I can sleep!" << endl;
12         }
13         void setColor(string clr) {
14             color = clr;
15         }
16         string getColor() {
17             return color;
18         }
19 };
```

Example

```
1  class Dog : public Animal {
2     public:
3      void setType(string tp) {
4          type = tp;
5      }
6      void displayInfo(string c) {
7          cout << "I am a " << type << endl;
8          cout << "My color is " << c << endl;
9      }
10     void bark() {
11         cout << "I can bark! Woof woof!!" << endl;
12     }
13 };
```

Example

```
1  int main() {
2      // Create object of the Dog class
3      Dog dog1;
4      // Calling members of the base class
5      dog1.eat();
6      dog1.sleep();
7      dog1.setColor("black");
8
9      // Calling member of the derived class
10     dog1.bark();
11     dog1.setType("mammal");
12
13     // Using getColor() of dog1 as argument
14     // getColor() returns string data
15     dog1.displayInfo(dog1.getColor());
16     return 0;
17 }
```

Inheritance Access Modes

- We have used the `public` keyword to inherit a class from a base class.
- We can also use `private` and `protected` keywords to inherit classes.
- `class Dog : private Animal`
- `class Dog : protected Animal`

Effect of Access Modes:

- **public:** Members of the base class are inherited as-is.
- **private:** All members of the base class become private members in the derived class.
- **protected:** Public members of the base class become protected members in the derived class.

Inheritance Access Modes

```
1  class Base {
2      public:
3          int x;  // Public member
4      protected:
5          int y;  // Protected member
6      private:
7          int z;  // Private member
8      public:
9          Base(int x_val, int y_val, int z_val) : x(x_val), y(y_val), z(z_val) {}
10         int getZ() const { return z; }
11 };
```

Inheritance Access Modes

```
1  class PublicDerived : public Base {
2      // x is public (from Base)
3      // y is protected (from Base)
4      // z is not accessible directly
5  public:
6      PublicDerived(int x_val, int y_val, int z_val) :
7      Base(x_val, y_val, z_val) {}
8      void printPublicDerived() {
9          // x is public
10         cout << "PublicDerived x: " << x << endl;
11         // y is protected
12         cout << "PublicDerived y: " << y << endl;
13         // Error: z is private in Base
14         // cout << "PublicDerived z: " << z << endl;
15         // Access z through getter
16         cout << "PublicDerived z: " << getZ() << endl;
17     }
18 };
```

Inheritance Access Modes

```
1  class ProtectedDerived : protected Base {
2      // x is protected (from Base)
3      // y is protected (from Base)
4      // z is not accessible directly
5    public:
6      // Constructor to initialize Base members
7    ProtectedDerived(int x_val, int y_val, int z_val) :
8    Base(x_val, y_val, z_val) {}
9    void printProtectedDerived() {
10       // x is protected
11       cout << "ProtectedDerived x: " << x << endl;
12       // y is protected
13       cout << "ProtectedDerived y: " << y << endl;
14       // Error: z is private in Base
15       // cout << "ProtectedDerived z: " << z << endl;
16       // Access z through getter
17       cout << "ProtectedDerived z: " << getZ() << endl;
18   }
19 };
```

Inheritance Access Modes

```
1  class PrivateDerived : private Base {
2  // x is private (from Base)
3  // y is private (from Base)
4  // z is not accessible directly
5  public:
6      // Constructor to initialize Base members
7      PrivateDerived(int x_val, int y_val, int z_val) :
8      Base(x_val, y_val, z_val) {}
9
10     void printPrivateDerived() {
11         // x is private
12         cout << "PrivateDerived x: " << x << endl;
13         // y is private
14         cout << "PrivateDerived y: " << y << endl;
15         // cout << "PrivateDerived z: " << z << endl;
16         // Error: z is private in Base
17         cout << "PrivateDerived z: " << getZ() << endl;
18         // Access z through getter
19     }
20 };
```

Inheritance Access Modes

```
1  int main() {
2      PublicDerived publicDerived(1, 2, 3);
3      publicDerived.printPublicDerived();
4
5      ProtectedDerived protectedDerived(4, 5, 6);
6      protectedDerived.printProtectedDerived();
7
8      PrivateDerived privateDerived(7, 8, 9);
9      privateDerived.printPrivateDerived();
10
11     return 0;
12     }
```

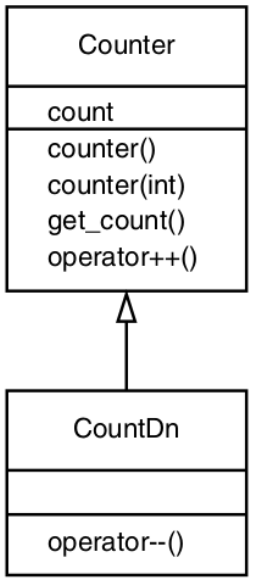
Benefits of Inheritance

- **Code Reusability:** Once a base class is written and debugged, it can be reused in different scenarios without modification.
- **Efficiency:** Reusing existing code saves time and money, and increases a program's reliability.
- **Design Flexibility:** Inheritance helps in conceptualizing and designing the program more effectively.
- **Ease of Distribution:** Inheritance facilitates the distribution and use of class libraries.
- Programmers can utilize classes created by others and derive new classes from them without needing to modify the original code.

Generalization in UML

In UML (Unified Modeling Language), inheritance is referred to as **generalization**.

- Generalization represents a relationship where the parent class is a more **general** form of the child class.
- Conversely, the child class is a more **specific** version of the parent class.



Example

```
1  class Counter   //base class
2  {
3      protected: //NOTE: not private
4          unsigned int count;  //count
5      public:
6          Counter() : count(0) //no-arg constructor
7           { }
8          Counter(int c) : count(c)   //1-arg constructor
9          { }
10         unsigned int get_count() const   //return count
11         { return count; }
12         Counter operator ++ ()    //incr count (prefix)
13         { return Counter(++count); }
14 };
15 class CountDn : public Counter   //derived class
16 {
17     public:
18         Counter operator -- ()  //decr count (prefix)
19         { return Counter(--count); }
20 };
```

Example

```
1  int main()
2   {
3       CountDn c1;    //c1 of class CountDn
4       cout << "\nc1=" << c1.get_count(); //display c1
5       ++c1; ++c1; ++c1;
6       cout << "\nc1=" << c1.get_count(); //increment c1, 3 times
7       //display it
8       --c1; --c1;
9       cout << "\nc1=" << c1.get_count();
10      cout << endl;
11      return 0;
12  }
```

Inheriting from the Counter Class

Following the 'Counter' class, we define a new class called 'CountDn':

- The 'CountDn' class introduces a new function, `operator--()`, which decrements the count.
- `CountDn` inherits all features from the 'Counter' class.
- `CountDn` does not need to redefine the constructor, `get_count()`, or `operator++()` functions, as these are already provided by 'Counter'.

The 'CountDn' class is derived from the 'Counter' class as follows:

```
class CountDn : public Counter
```

- This line uses a single colon (not to be confused with the double colon for scope resolution).
- The keyword `public` followed by the base class name `Counter` sets up the inheritance relationship.
- This means that `CountDn` inherits from `Counter`.

Base-class member-access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by member functions, `friend` functions and nonmember functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.

Types of inheritance

- Single Inheritance: A class is derived from **one** base class.
- Multiple Inheritance: A derived class inherits **simultaneously** from two or more (possibly unrelated) base classes.
- Multilevel Inheritance: A class is derived from a **base class**, which in turn is derived from another base class, forming a chain.
- A class having more than one parent class, such inheritance is called Multilevel Inheritance.

Single inheritance: Example I

```
1  class Vehicle  // base class
2  {
3  public:
4      Vehicle() {
5          cout << "This is a Vehicle" << endl;
6      }
7  };
8  class Car: public Vehicle // first sub class
9  {
10 };
11 class Bus: public Vehicle // second sub class
12 {
13 };
14 int main() // main function
15 {
16     // creating object of sub class will
17     // invoke the constructor of base class
18     Car obj1;
19     Bus obj2;
20     return 0;
21 }
```

Single inheritance: Example II

```
1  class Vehicle {
2  public:
3      string brand;
4      int year;
5      Vehicle(string b, int y) : brand(b), year(y) {}
6      void displayInfo() {
7          cout << "Brand: " << brand << endl;
8          cout << "Year: " << year << endl;
9      }
10 };
11 class Car : public Vehicle {
12 public:
13     int doors;
14     Car(string b, int y, int d) : Vehicle(b, y), doors(d) {}
15     void displayCarInfo() {
16         displayInfo();
17         cout << "Doors: " << doors << endl;
18     }
19 };
20 int main() {
21     Car myCar("Toyota", 2020, 4);
22     myCar.displayCarInfo();
23     return 0;
24 }
```

Multiple inheritance: Example I

```
1  class Vehicle {  // base class
2    public:
3      Vehicle() {
4        cout << "This is a Vehicle" << endl;
5      }
6  };
7  class FourWheeler { // second base class
8    public:
9      FourWheeler() {
10       cout << "This is a 4 wheeler Vehicle" << endl;
11     }
12 };
13 // sub class derived from two base classes
14 class Car: public Vehicle, public FourWheeler
15 {
16     public:
17         Car() {
18             cout << "This is a car" << endl;
19     }
20 };
21 int main() {
22         Car obj;
23         return 0;
24 }
```

Multiple inheritance: Example II

```
1  class Vehicle {  // base class
2     public:
3       Vehicle() {
4         cout << "This is a Vehicle" << endl;
5       }
6       void display() {
7           cout << "disp of Vehicle" << endl;
8       }
9   };
10  class FourWheeler { // second base class
11    public:
12      FourWheeler() {
13        cout << "This is a 4 wheeler Vehicle" << endl;
14      }
15      void display() {
16          cout << "disp of Wheeler Vehicle" << endl;
17      }
18  };
```

Multiple inheritance: Example II

```
1  // sub class derived from two base classes
2  class Car: public Vehicle, public FourWheeler
3  {
4      public:
5      Car() {
6          cout << "This is a car" << endl;
7      }
8      void display() {
9          cout << "disp of Car" << endl;
10     }
11 };
12 int main() {
13         Car obj;
14         obj.display();
15         return 0;
16 }
```

Output:
This is a Vehicle
This is a 4 wheeler Vehicle
This is a car
disp of Car

Multiple inheritance: Example III

```
1  class Vehicle {  // base class
2     public:
3       Vehicle() {
4         cout << "This is a Vehicle" << endl;
5       }
6       void display() {
7           cout << "disp of Vehicle" << endl;
8       }
9   };
10  class FourWheeler { // second base class
11    public:
12      FourWheeler() {
13        cout << "This is a 4 wheeler Vehicle" << endl;
14      }
15  };
```

Multiple inheritance: Example III

```
1  // sub class derived from two base classes
2  class Car: public Vehicle, public FourWheeler
3  {
4      public:
5      Car() {
6          cout << "This is a car" << endl;
7      }
8  };
9  int main() {
10         Car obj;
11         obj.display();
12         return 0;
13 }
```

Output:
This is a Vehicle
This is a 4 wheeler Vehicle
This is a car
disp of Vehicle

Multiple inheritance: Example IV

```
1  class Vehicle {  // base class
2     public:
3       Vehicle() {
4         cout << "This is a Vehicle" << endl;
5       }
6       void display() {
7           cout << "disp of Vehicle" << endl;
8       }
9   };
10  class FourWheeler { // second base class
11    public:
12      FourWheeler() {
13        cout << "This is a 4 wheeler Vehicle" << endl;
14      }
15      void display() {
16          cout << "disp of Wheeler Vehicle" << endl;
17      }
18  };
```

Multiple inheritance: Example IV

```
1  // sub class derived from two base classes
2  class Car: public Vehicle, public FourWheeler
3  {
4      public:
5      Car() {
6          cout << "This is a car" << endl;
7      }
8  };
9  int main() {
10         Car obj;
11         obj.display();
12         return 0;
13 }
```

ERROR: display() method exists in both base classes!

Multilevel inheritance: Example I

```
1  class person
2  {
3      char name[100], gender[10];
4      int age;
5      public:
6          void getdata() {
7          cout << "Name: ";
8          cin >> name;
9          cout << "Age: ";
10         cin >> age;
11         cout << "Gender: ";
12         cin >> gender;
13     }
14     void display() {
15         cout << "Name: " << name << endl;
16         cout << "Age: " << age << endl;
17         cout << "Gender: " << gender << endl;
18     }
19 };
```

Multilevel inheritance: Example I

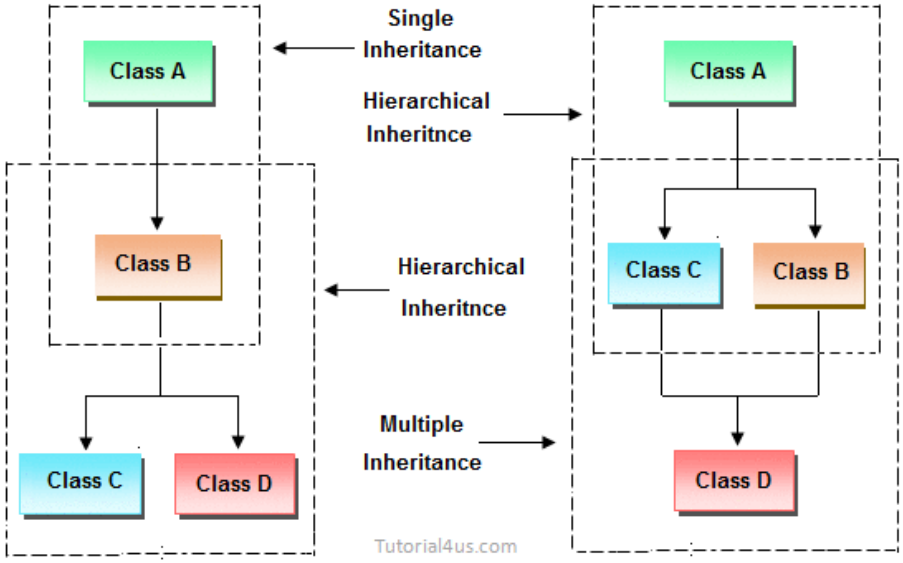
```
1  class employee : public person
2  {
3      char company[100];
4      float salary;
5      public:
6          void getdata() {
7              person::getdata();
8              cout << "Name of Company: ";
9              cin>>company;
10             cout << "Salary: Rs.";
11             cin >> salary;
12         }
13         void display()  {
14             person::display();
15             cout << "Name of Company: " << company << endl;
16             cout << "Salary: Rs." << salary << endl;
17         }
18 };
```

Multilevel inheritance: Example I

```
1  class programmer: public employee
2  {
3      int number;
4      public:
5      void getdata() {
6          employee::getdata();
7          cout << "Number of programming language known: ";
8          cin >> number;
9      }
10     void display() {
11     employee::display();
12     cout << "Number of programming language known: " << number<<endl;
13     }
14 };
15 int main()
16 {
17     programmer p;
18     cout << "Enter data" << endl;
19     p.getdata();
20     cout << endl << "Displaying data" << endl;
21     p.display();
22     return 0;
23 }
```

Hybrid inheritance

- Hybrid inheritance is used when we **mix different types of inheritance** within a single program.
- For example, we can combine:
  - **Single inheritance** with **multiple inheritance**, or
  - **Multiple inheritance** within a single program.



Inheritance hierarchy