Object Oriented Programming

Inheritance

Mr. Usman Wajid

usman.wajid@nu.edu.pk

April 12, 2023

National University
of Computer & Emerging Sciences

What is Inheritance?

Inheritance

It is a concept in objected-oriented programming that allows a class to inherit properties and behaviors from another class

What is Inheritance?

Inheritance

It is a concept in objected-oriented programming that allows a class to inherit properties and behaviors from another class

- Inheritance is an ”is a” relationship. Example, ”every employee is a person”

What is Inheritance?

Inheritance

It is a concept in objected-oriented programming that allows a class to inherit properties and behaviors from another class

- Inheritance is an ”is a” relationship. Example, ”every employee is a person”
- It allows us to create new classes from existing classes
  - New classes are called the derived classes
  - Existing classes are called the base classes

What is Inheritance?

Inheritance

It is a concept in objected-oriented programming that allows a class to inherit properties and behaviors from another class

- Inheritance is an ”is a” relationship. Example, ”every employee is a person”
- It allows us to create new classes from existing classes
  - New classes are called the derived classes
  - Existing classes are called the base classes
- Derived classes inherit the properties of the base classes

Inheritance: Basic Syntax

```
class DerivedClass : member-access-specifier BaseClass
{
        // members of derived class
};
```

Inheritance: Basic Syntax

```
class DerivedClass : member-access-specifier BaseClass
{
        // members of derived class
};
```

- Where `member-access-specifier` can be `public`, `protected` or (by default)

Inheritance: Basic Syntax

```
class DerivedClass : member-access-specifier BaseClass
{
        // members of derived class
};
```

- Where `member-access-specifier` can be `public`, `protected` or (by default)
- The `private` members of a base class are always private to the derived class

Inheritance: Basic Syntax

```
class DerivedClass : member-access-specifier BaseClass
{
        // members of derived class
};
```

- Where `member-access-specifier` can be `public`, `protected` or (by default)
- The `private` members of a base class are always private to the derived class
- Therefore, derived class objects can not directly access the `private` members of the base class

Some Real-life Examples

Base class	**Derived class**
Student	Graduate Student Undergraduate Student
Shape	Circle Rectangle
Employee	Faculty Member Staff member

Inheritance continued . . .

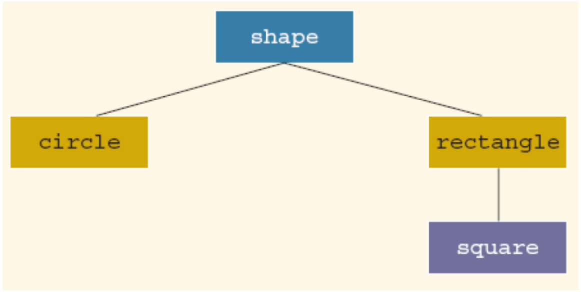
- Inheritance can be viewed as a tree-like, or hierarchical structure

Inheritance continued . . .

- Inheritance can be viewed as a tree-like, or hierarchical structure
- The base class is shown at the top with respective derived classes arranged in a hierarchical order

Inheritance continued . . .

- Inheritance can be viewed as a tree-like, or hierarchical structure
- The base class is shown at the top with respective derived classes arranged in a hierarchical order

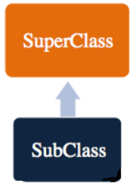


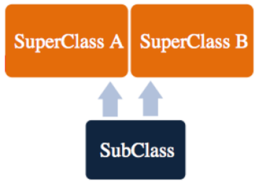
Inheritance continued . . .

1. **Single Inheritance:** When a new class is derived from only a single base class.
2. **Multiple Inheritance:** When a new class is derived from multiple base classes.

Defining a simple subclass

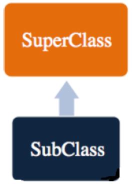
- Each individual class can be used as a base class (or super-class) to derive a new class (sub-class)

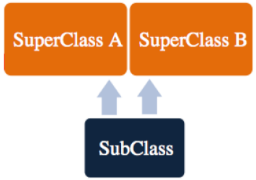




Defining a simple subclass

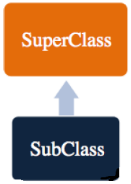
- Each individual class can be used as a base class (or super-class) to derive a new class (sub-class)
- It's also allowed to use more than one superclass to define a subclass.

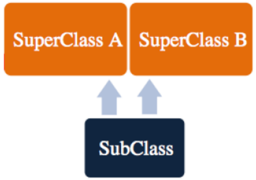




Defining a simple subclass

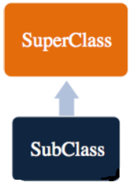
- Each individual class can be used as a base class (or super-class) to derive a new class (sub-class)
- It's also allowed to use more than one superclass to define a subclass.
- Note: the arrows always point to the superclass(es).

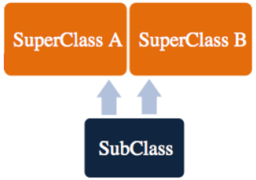




Defining a simple subclass

- Each individual class can be used as a base class (or super-class) to derive a new class (sub-class)
- It's also allowed to use more than one superclass to define a subclass.
- Note: the arrows always point to the superclass(es).
- We can refer super-classes as base classes , and sub-classes as derived classes





Inheritance Example 1

```cpp
class Super{
        private:
        int x;
        public:
        void setX(int x){
                this->x = x;
        }
        int const getX(){
                return x;
        }
};

class Sub: Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

Inheritance Example 1

```
class Super{
        private:
        int x;
        public:
        void setX(int x){
                this->x = x;
        }
        int const getX(){
                return x;
        }
};

class Sub: Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

- This syntax is same as,

```
   class Sub: private Super {
   };
```

Inheritance Example 1

```
class Super{
        private:
        int x;
        public:
        void setX(int x){
                this->x = x;
        }
        int const getX(){
                return x;
        }
};

class Sub: Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

- This syntax is same as,

```
   class Sub: private Super {
   };
```

- This will give a compilation error

Inheritance Example 1

```
class Super{
        private:
        int x;
        public:
        void setX(int x){
                this->x = x;
        }
        int const getX(){
                return x;
        }
};

class Sub: Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

- This syntax is same as,

```
class Sub: private Super {
};
```

- This will give a compilation error
- Because the `private` access specifier manipulates even the public members of Super class as private

Inheritance Example 1 continued . . .

```
class Super{
        private:
                int x;
        public:
                void setX(int x){
                        this->x = x;
                }
                int const getX(){
                        return x;
                }
};

class Sub: public Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

Inheritance Example 1 continued . . .

```
class Super{
        private:
                int x;
        public:
                void setX(int x){
                        this->x = x;
                }
                int const getX(){
                        return x;
                }
};

class Sub: public Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

- In,

  `class Sub:  public Super`

  the keyword `public` doesn't mean that all the members of Super class will become public

Inheritance Example 1 continued ...

```
class Super{
        private:
                int x;
        public:
                void setX(int x){
                        this->x = x;
                }
                int const getX(){
                        return x;
                }
};

class Sub: public Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

- In,

  `class Sub:  public Super`

  the keyword `public` doesn't mean that all the members of Super class will become public

- Instead, it means that only `public` members of Super class can be accessed by the Sub class objects

Inheritance Example 1 continued ...

```
class Super{
        private:
                int x;
        public:
                void setX(int x){
                        this->x = x;
                }
                int const getX(){
                        return x;
                }
};

class Sub: public Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

- In,

  `class Sub:  public Super`

  the keyword `public` doesn't mean that all the members of Super class will become public

- Instead, it means that only `public` members of Super class can be accessed by the Sub class objects

- Hence, statements such as `obj.x = 10` or `cout<<obj.x;` are not allowed here

Inheritance Example 1 continued . . .

```
class Super{
        private:
                int x;
        public:
                void setX(int x){
                        this->x = x;
                }
                int const getX(){
                        return x;
                }
};

class Sub: public Super {
};

int main() {
        Sub obj;
        obj.setX(10);
        cout<<"x = "<<obj.getX()<<endl;

}
```

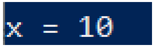
- In,

  `class Sub: public Super`

  the keyword `public` doesn't mean that all the members of Super class will become public

- Instead, it means that only `public` members of Super class can be accessed by the Sub class objects

- Hence, statements such as `obj.x = 10` or `cout<<obj.x;` are not allowed here

- Output: 

```
x = 10
```

Inheritance Example 2

```cpp
class Person {
        private:
                string name;
                int age;
        public:
                void setName(string name){
                        this->name = name;
                }
                string getName(){ return name; }

                void setAge(int age){
                        this->age = age;
                }
                int getAge(){ return age; }
};
class Student: public Person {
        private:
                int rollNo;
                char sec;
        public:
                void setRollNo(int rollNo){
                        this->rollNo = rollNo;
                }
                int getRollNo(){ return rollNo; }
                void setSec(char sec){
                        this->sec = sec;
                }
                char getSec(){ return sec; }
};
```

```cpp
int main() {
        Student ali;
        ali.setName("Ali Imran");
        ali.setAge(12);
        ali.setRollNo(01);
        ali.setSec('A');

        cout<<"name:\t"<<ali.getName()<<endl;
        cout<<"age:\t"<<ali.getAge()<<endl;
        cout<<"rollno:\t"<<ali.getRollNo()<<endl;
        cout<<"sec:\t"<<ali.getSec()<<endl;
}
```

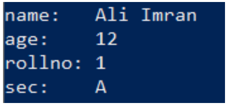
Inheritance Example 2

```cpp
class Person {
        private:
                string name;
                int age;
        public:
                void setName(string name){
                        this->name = name;
                }
                string getName(){ return name; }

                void setAge(int age){
                        this->age = age;
                }
                int getAge(){ return age; }
};
class Student: public Person {
        private:
                int rollNo;
                char sec;
        public:
                void setRollNo(int rollNo){
                        this->rollNo = rollNo;
                }
                int getRollNo(){ return rollNo; }
                void setSec(char sec){
                        this->sec = sec;
                }
                char getSec(){ return sec; }
};
```

```cpp
int main() {
        Student ali;
        ali.setName("Ali Imran");
        ali.setAge(12);
        ali.setRollNo(01);
        ali.setSec('A');

        cout<<"name:\t"<<ali.getName()<<endl;
        cout<<"age:\t"<<ali.getAge()<<endl;
        cout<<"rollno:\t"<<ali.getRollNo()<<endl;
        cout<<"sec:\t"<<ali.getSec()<<endl;
}
```



```
name:   Ali Imran
age:    12
rollno: 1
sec:    A
```

Inheritance Example 3:

```
class Person {
        private:
                string name;
                int age;
        public:
                Person(string name="N/A",int age=0)
                     {
                        setName(name);
                        setAge(age);
                }
                void setName(string name){
                        this->name = name; }
                string getName(){ return name; }

                void setAge(int age){
                        this->age = age; }
                int getAge(){ return age; }
};
```

```
class Student: public Person {
        private:
                int rollNo;
                char sec;
        public:
                Student(string name="N/A", int age=0,
                        int rollNo=0, char sec='-') :
                        Person(name, age)
                {
                        setRollNo(rollNo);
                        setSec(sec);
                } // end Student constructor
                void setRollNo(int rollNo){
                        this->rollNo = rollNo; }
                int getRollNo(){ return rollNo; }
                void setSec(char sec){ this->sec = sec; }
                char getSec(){ return sec; }
};
```

```
int main() {
        Student ali;

        cout<<"name:\t"<<ali.getName()<<endl;
        cout<<"age:\t"<<ali.getAge()<<endl;
        cout<<"rollno:\t"<<ali.getRollNo()<<endl;
        cout<<"sec:\t"<<ali.getSec()<<endl;
}
```

Inheritance Example 3:

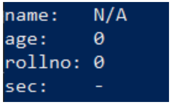
```
class Person {
        private:
                string name;
                int age;
        public:
                Person(string name="N/A",int age=0)
                     {
                        setName(name);
                        setAge(age);
                }
                void setName(string name){
                        this->name = name; }
                string getName(){ return name; }

                void setAge(int age){
                        this->age = age; }
                int getAge(){ return age; }
};
```

```
class Student: public Person {
        private:
                int rollNo;
                char sec;
        public:
                Student(string name="N/A", int age=0,
                        int rollNo=0, char sec='-') :
                        Person(name, age)
                {
                        setRollNo(rollNo);
                        setSec(sec);
                } // end Student constructor
                void setRollNo(int rollNo){
                        this->rollNo = rollNo; }
                int getRollNo(){ return rollNo; }
                void setSec(char sec){ this->sec = sec; }
                char getSec(){ return sec; }
};
```

```
int main() {
        Student ali;

        cout<<"name:\t"<<ali.getName()<<endl;
        cout<<"age:\t"<<ali.getAge()<<endl;
        cout<<"rollno:\t"<<ali.getRollNo()<<endl;
        cout<<"sec:\t"<<ali.getSec()<<endl;
}
```



```
name:    N/A
age:     0
rollno:  0
sec:     -
```

Inheritance Example 3 with Default constructors

```cpp
class Person {
        private:
                string name;
                int age;
        public:
                Person(string name="N/A",int age=0)
                     {
                        setName(name);
                        setAge(age);
                }
                void setName(string name){
                        this->name = name; }
                string getName(){ return name; }

                void setAge(int age){
                        this->age = age; }
                int getAge(){ return age; }
};
```

```cpp
class Student: public Person {
        private:
                int rollNo;
                char sec;
        public:
                Student(string name="N/A", int age=0,
                        int rollNo=0, char sec='-') :
                        Person(name, age)
                {
                        setRollNo(rollNo);
                        setSec(sec);
                } // end Student constructor
                void setRollNo(int rollNo){
                        this->rollNo = rollNo; }
                int getRollNo(){ return rollNo; }
                void setSec(char sec){ this->sec = sec; }
                char getSec(){ return sec; }
};
```

```cpp
int main() {
        Student ali={"Ali Imran", 12, 01, 'A'};

        cout<<"name:\t"<<ali.getName()<<endl;
        cout<<"age:\t"<<ali.getAge()<<endl;
        cout<<"rollno:\t"<<ali.getRollNo()<<endl;
        cout<<"sec:\t"<<ali.getSec()<<endl;
}
```

Inheritance Example 3 with Default constructors

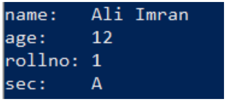
```cpp
class Person {
        private:
                string name;
                int age;
        public:
                Person(string name="N/A",int age=0)
                     {
                        setName(name);
                        setAge(age);
                }
                void setName(string name){
                        this->name = name; }
                string getName(){ return name; }

                void setAge(int age){
                        this->age = age; }
                int getAge(){ return age; }
};
```

```cpp
class Student: public Person {
        private:
                int rollNo;
                char sec;
        public:
                Student(string name="N/A", int age=0,
                        int rollNo=0, char sec='-') :
                        Person(name, age)
                {
                        setRollNo(rollNo);
                        setSec(sec);
                } // end Student constructor
                void setRollNo(int rollNo){
                        this->rollNo = rollNo; }
                int getRollNo(){ return rollNo; }
                void setSec(char sec){ this->sec = sec; }
                char getSec(){ return sec; }
};
```

```cpp
int main() {
        Student ali={"Ali Imran", 12, 01, 'A'};

        cout<<"name:\t"<<ali.getName()<<endl;
        cout<<"age:\t"<<ali.getAge()<<endl;
        cout<<"rollno:\t"<<ali.getRollNo()<<endl;
        cout<<"sec:\t"<<ali.getSec()<<endl;
}
```



```
name:    Ali Imran
age:     12
rollno:  1
sec:     A
```