

- Examples

To get a deeper understanding of inheritance, we'll look at a few examples in this lesson.

WE'LL COVER THE FOLLOWING

- Example 1: Inheritance
 - Explanation
- Example 2: Abstract base class inheritance
 - Explanation
- Example 3: Access rights in inheritance
 - Explanation
- Example 4: Base class inheritance
 - Explanation

Example 1: Inheritance

```
#include <iostream>

class Account{

public:
    Account(double b): balance(b){}

    void deposit(double amt){
        balance += amt;
    }

    void withdraw(double amt){
        balance -= amt;
    }

    double getBalance() const {
        return balance;
    }

private:
    double balance;

};
```



```

class BankAccount: public Account{

public:
    // using Account::Account;
    BankAccount(double b): Account(b){}

    void addInterest(){
        deposit( getBalance()*0.05 );
    }
};

int main(){

    std::cout << std::endl;

    BankAccount bankAcc(100.0);
    bankAcc.deposit(50.0);
    bankAcc.deposit(25.15);
    bankAcc.withdraw(30);
    bankAcc.addInterest();

    std::cout << "bankAcc.getBalance(): " << bankAcc.getBalance() << std::endl;

    std::cout << std::endl;

}

```



Explanation

- We have created two classes, i.e., `Account` and `BankAccount`.
- The `BankAccount` class inherits the `Account` class publicly in line 25.
- The `public` member functions of the `Account` class are available to the `BankAccount` class and we can access them using the `.` operator (line 46).

Example 2: Abstract base class inheritance

```

#include <iostream>
#include <string>

class Abstract{
public:
    virtual ~Abstract() = 0;
};

Abstract::~~Abstract(){}

class Concret: public Abstract{};

```



```

class HumanBeing{
public:
    HumanBeing(const std::string n): name(n){

        std::cout << name << " created." << std::endl;
    }

    virtual std::string getSex() const= 0;

private:
    std::string name;

};

class Man: public HumanBeing{
public:

    // using HumanBeing::HumanBeing;
    Man(const std::string n): HumanBeing(n){}

    std::string getSex() const{
        return "male";
    }
};

class Woman: public HumanBeing{
public:

    // using HumanBeing::HumanBeing;
    Woman(const std::string n): HumanBeing(n){}

    std::string getSex() const{
        return "female";
    }
};

int main(){

    std::cout << std::endl;

    // Abstract abstract;    // ERROR
    Concret concret;

    // HumanBeing humanBeing("grimm");    // ERROR

    Man schmidt("Schmidt");
    Woman huber("Huber");

    std::cout << "schmidt.getSex(): " << schmidt.getSex() << std::endl;
    std::cout << "huber.getSex(): " << huber.getSex() << std::endl;

    std::cout << std::endl;

}

```



- We have created a pure virtual class `Abstract` and we cannot make an instance of this class as it will give an error.
- The classes `Man` and `Woman` inherit publically from the class `HumanBeing`.
- `getSex` function is pure virtual so we cannot make an instance of `HumanBeing` in `main`.
- The instances of the `Man` and `Woman` classes can access the `getSex` function by using the `.` operator and they must be overridden in derived classes.

Example 3: Access rights in inheritance

```
class Account{
public:
    int pub{0};
protected:
    int prot{0};
private:
    int pri{0};
};

class PubAccount: public Account{
public:
    PubAccount(){
        pub + prot; // public + protected
    }
};

class ProtAccount: protected Account{
public:
    ProtAccount(){
        pub + prot; // protected + protected
    }
};

class PriAccount: private Account{
public:
    PriAccount(){
        pub + prot; // private + private
    }
};

int main(){

    PubAccount pubAccount;
    ProtAccount proAccount;
    PriAccount priAccount;
```

```
pubAccount.pub;  
}
```



Explanation

In this example, we figured out the different access rights available to classes when inheriting a parent class using `public`, `protected`, and `private` keywords.

- Only in the case of the `pubAccount` class, can we access the publicly available member of the base class in `main`.
- For the `priAccount` and `proAccount` classes, we can access public and protected members' variables in derived classes, but not in `main`.

Example 4: Base class inheritance

```
#include <iostream>  
#include <string>  
  
class Account{  
public:  
  
    Account() = default;  
  
    Account(double amt, std::string c): amount(amt), cur(c){  
        std::cout << "Account:amount: " << amount << std::endl;  
        std::cout << "Account:cur: " << cur << std::endl;  
    }  
  
private:  
    double amount;  
    std::string cur;  
};  
  
class BankAccount: public Account{  
public:  
    BankAccount(double amt, std::string n): Account(amt, "EUR"), name(n){  
        std::cout << "BankAccount:name: " << name << std::endl;  
    }  
private:  
    std::string name;  
};  
  
int main(){  
  
    std::cout << std::endl;  
  
    Account acc(100.0, "USD");  
}
```

```
Account acc(100.0, USD );

std::cout << std::endl;

BankAccount bankAcc(200.0, "grimm");

std::cout << std::endl;

}
```



Explanation

- We have created two classes,i.e., `Account` and `BankAccount` .
- The class `BankAccount` inherits publically from the `Account` class.
- We have created two instances of these classes in `main` .
- For the `BankAccount` class, if the `cur` parameter is not passed, then the default `EUR` is passed to the `Account` class.

In the next lesson, we'll solve a few exercises.