25 June 2020

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25 June 2020 | Name: | Srinidhi J C |
| Course: | C++ Programming | USN: | 4al16ec078 |
| Topic: | Inheritance & Polymorphism, | Semester & Section: | 8th & b |
| Github Repository: | SrinidhiJC078 |  |  |
| FORENOON SESSION DETAILS | | | | | |
| Image of session  A screenshot of a cell phone  Description automatically generated | | | | | |

Report:

## Inheritance

In C++, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

* **derived class** (child) - the class that inherits from another class
* **base class** (parent) - the class being inherited from

To inherit from a class, use the : symbol.

In the example below, the Car class (child) inherits the attributes and methods from the Vehicle class (parent):

### Example

// Base class  
class Vehicle {  
  public:   
    string brand = "Ford";  
    void honk() {  
      cout << "Tuut, tuut! \n" ;  
    }  
};  
  
// Derived class  
**class Car: public Vehicle** {  
  public:   
    string model = "Mustang";  
};  
  
int main() {  
  Car myCar;  
  myCar.honk();  
  cout << myCar.brand + " " + myCar.model;  
  return 0;  
}

## Multilevel Inheritance

A class can also be derived from one class, which is already derived from another class.

In the following example, MyGrandChild is derived from class MyChild (which is derived from MyClass).

### Example

// Base class (parent)  
class MyClass {  
  public:   
    void myFunction() {  
      cout << "Some content in parent class." ;  
    }  
};  
  
// Derived class (child)  
class MyChild: public MyClass {  
};  
// Derived class (grandchild)   
class MyGrandChild: public MyChild {  
};  
int main() {  
  MyGrandChild myObj;  
  myObj.myFunction();  
  return 0;  
}

## Multiple Inheritance

A class can also be derived from more than one base class, using a **comma-separated list:**

### Example

// Base class  
class MyClass {  
  public:   
    void myFunction() {  
      cout << "Some content in parent class." ;  
    }  
};  
// Another base class  
class MyOtherClass {  
  public:   
    void myOtherFunction() {  
      cout << "Some content in another class." ;  
    }  
};  
// Derived class   
**class MyChildClass: public MyClass, public MyOtherClass** {  
};  
int main() {  
  MyChildClass myObj;  
  myObj.myFunction();  
  myObj.myOtherFunction();  
  return 0;  
}

## Access Specifiers

You learned from the [Access Specifiers](https://www.w3schools.com/cpp/cpp_access_specifiers.asp) chapter that there are three specifiers available in C++. Until now, we have only used public (members of a class are accessible from outside the class) and private (members can only be accessed within the class). The third specifier, protected, is similar to private, but it can also be accessed in the **inherited** class:

### Example

// Base class  
class Employee {  
  **protected: // Protected access specifier**  
    int salary;  
};  
// Derived class  
class Programmer: public Employee {  
  public:  
    int bonus;  
    void setSalary(int s) {  
      salary = s;  
    }  
    int getSalary() {  
      return salary;  
    }  
};  
int main() {  
  Programmer myObj;  
  myObj.setSalary(50000);  
  myObj.bonus = 15000;  
  cout << "Salary: " << myObj.getSalary() << "\n";  
  cout << "Bonus: " << myObj.bonus << "\n";  
  return 0;  
}

## Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; [**Inheritance**](https://www.w3schools.com/cpp/cpp_inheritance.asp) lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a base class called Animal that has a method called animalSound(). Derived classes of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

### Example

// Base class  
class Animal {  
  public:  
    void animalSound() {  
    cout << "The animal makes a sound \n" ;  
  }  
};  
// Derived class  
class Pig : public Animal {  
  public:  
    void animalSound() {  
    cout << "The pig says: wee wee \n" ;  
  }  
};  
// Derived class  
class Dog : public Animal {  
  public:  
    void animalSound() {  
    cout << "The dog says: bow wow \n" ;  
  }  
};

Now we can create Pig and Dog objects and override the animalSound() method:

### Example

// Base class  
class Animal {  
  public:  
    void animalSound() {  
    cout << "The animal makes a sound \n" ;  
  }  
};  
// Derived class  
class Pig : public Animal {  
  public:  
    void animalSound() {  
    cout << "The pig says: wee wee \n" ;  
   }  
};  
// Derived class  
class Dog : public Animal {  
  public:  
    void animalSound() {  
    cout << "The dog says: bow wow \n" ;  
  }  
};  
  
int main() {  
  Animal myAnimal;  
  Pig myPig;  
  Dog myDog;  
  myAnimal.animalSound();  
  myPig.animalSound();  
  myDog.animalSound();  
  return 0;  
}

## C++ Exceptions

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an **exception** (throw an error).

## try and catch

Exception handling in C++ consist of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

### Example

try {  
  // Block of code to try  
  throw exception; // Throw an exception when a problem arise  
}  
catch () {  
  // Block of code to handle errors  
}

Consider the following example:

### Example

try {  
  int age = 15;  
  if (age > 18) {  
    cout << "Access granted - you are old enough.";  
  } else {  
    throw (age);  
  }  
}  
catch (int myNum) {  
  cout << "Access denied - You must be at least 18 years old.\n";  
  cout << "Age is: " << myNum;   
}

#### Example explained

We use the try block to test some code: If the age variable is less than 18, we will throw an exception, and handle it in our catch block.

In the catch block, we catch the error and do something about it. The catch statement takes a **parameter**: in our example we use an int variable (myNum) (because we are throwing an exception of int type in the try block (age)), to output the value of age.

If no error occurs (e.g. if age is 20 instead of 15, meaning it will be be greater than 18), the catch block is skipped:

### Example

int age = 20;

You can also use the throw keyword to output a reference number, like a custom error number/code for organizing purposes:

### Example

try {  
  int age = 15;  
  if (age > 18) {  
    cout << "Access granted - you are old enough.";  
  } else {  
    throw 505;  
  }  
}  
catch (int myNum) {  
  cout << "Access denied - You must be at least 18 years old.\n";  
  cout << "Error number: " << myNum;   
}

## Handle Any Type of Exceptions (...)

If you do not know the throw **type** used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

### Example

try {  
  int age = 15;  
  if (age > 18) {  
    cout << "Access granted - you are old enough.";  
  } else {  
    throw 505;  
  }  
}  
catch (...) {  
  cout << "Access denied - You must be at least 18 years old.\n";  
}

## C++ Files

The fstream library allows us to work with files.

To use the fstream library, include both the standard <iostream> **AND** the <fstream> header file:

### Example

#include <iostream>  
#include <fstream>

There are three objects included in the fstream library, which are used to create, write or read files:

|  |  |
| --- | --- |
| **Object/Data Type** | **Description** |
| ofstream | Creates and writes to files |
| ifstream | Reads from files |
| fstream | A combination of ofstream and ifstream: creates, reads, and writes to files |

## Create and Write To a File

To create a file, use either the ofstream or fstream object, and specify the name of the file.

To write to the file, use the insertion operator (<<).

### Example

#include <iostream>  
#include <fstream>  
using namespace std;  
int main() {  
  // Create and open a text file  
  ofstream MyFile("filename.txt");  
  // Write to the file  
  MyFile << "Files can be tricky, but it is fun enough!";  
  
  // Close the file  
  MyFile.close();  
}

#### Why do we close the file?

It is considered good practice, and it can clean up unnecessary memory space.

## Read a File

To read from a file, use either the ifstream or fstream object, and the name of the file.

### Example

// Create a text string, which is used to output the text file  
string myText;  
// Read from the text file  
ifstream MyReadFile("filename.txt");  
// Use a while loop together with the getline() function to read the file line by line  
while (getline (MyReadFile, myText)) {  
  // Output the text from the file  
  cout << myText;  
}  
// Close the file  
MyReadFile.close();