

System Design and Programming II

CSCI – 1943

David L. Sylvester, Sr., Professor



Chapter 15

Inheritance, Polymorphism,
And Virtual Functions

What is Inheritance?

Inheritance allows a new class to be based on an existing class. The new class inherits all the member variables and functions (except the constructors and destructors) of the class it is based on.

Inheritance and the “is a” Relationship

When one object is a specialized version of another object, there is an “is a” relationship between them.

- Ex:
- A poodle is a dog.
 - A car is a vehicle.
 - A tree is a plant.
 - A rectangle is a shape.
 - A football player is an athlete.

When a “is a” relationship exists between classes, it means that the specialized class has all of the characteristics of the general class, plus additional characteristics, that make it special.

Inheritance involves a base class and a derived class. The base class is the general class and the derived class is the specialized class.

The derived class inherits the member variables and member functions of the base class without any of them being rewritten. Furthermore, new member variables and functions may be added to the derived class to make it more specialized than the base class.

The following page show the GradedActivity class that has the general characteristics of a student's graded activity. Many different types of graded activities exist, such as quizzes, midterm exams, final exams, lab reports, essays, and so on. Because the numeric scores might be determined differently for each of the graded activities, we can create derived classes to handle each one.

Therefore, a FinalExam class can be derived from the Graded Activity class. The FinalExam class has member variables for the number of questions, the number of points each question is worth, and the number of questions missed.

To declare the class FinalExam, you would do the following:

```
class FinalExam : public GradedActivity
```

Class being declared
(the derived class)

Base class

If we want to express the relationship between the two classes, we can say that a FinalExam **is a** GradedActivity.

The word public, which precedes the name of the base class is the base class access specification. It affects how the members of the base class are inherited by the derived class. When you create an object of a derived class, you can think of it as being built on top of the base class. The members of the base class object become member of the derived class object. How the base class members appear in the derived class is determined by the base class access specification.

The GradedActivity class has both private members and public members. The FinalExam class is derived from the GradedActivity class, using public access specification. This means that the public members of the GradedActivity class will become public members of the FinalExam class. The private members of the GradedActivity class cannot be accessed directly by code in the FinalExam class.

Although the private members of the GradedActivity class are inherited, it's as though they are invisible to the code in the FinalExam class. They can only be accessed by the member functions of the GradedActivity class.

Sample Program

```
#include <iostream>
using namespace std;

class GradedActivity
{
private:
double score; // To hold the numeric score
public:

// Default constructor
GradedActivity()
{ score = 0.0; }

// Constructor
GradedActivity(double s)
{ score = s; }

// Mutator function
void setScore(double s)
{ score = s; }

// Accessor functions
double getScore() const
{ return score; }

char getLetterGrade() const;
};
```

Program Output with example input Shown in Bold

Enter your numeric test score: **80** [Enter]

The grade for that test is B

Program Output with Different Examples Input shown in Bold

Enter your numeric test score: **75** [Enter]

The grade for that test is C

```
char GradedActivity::getLetterGrade() const
{
char letterGrade; // To hold the letter
grade
if (score > 89)
letterGrade = 'A';
else if (score > 79)
letterGrade = 'B';
else if (score > 69)
letterGrade = 'C';
else if (score > 59)
letterGrade = 'D';
else
letterGrade = 'F';

return letterGrade;
}

int main()
{
double testScore; // To hold a test score

// Create a GradedActivity object for the
test.
GradedActivity test;

// Get a numeric test score from the user.
cout << "Enter your numeric test score: ";
cin >> testScore;

// Store the numeric score in the test
object.
test.setScore(testScore);

// Display the letter grade for the test.
cout << "The grade for that test is " <<
test.getLetterGrade() << endl;

return 0;
}
```

GRADEACTIVITY (.h)

```
#ifndef GRADEACTIVITY_H
#define GRADEACTIVITY_H

// GradedActivity class declaration

class GradedActivity
{
private:
    double score; // To hold the numeric score
public:
    // Default constructor
    GradedActivity()
    { score = 0.0; }

    // Constructor
    GradedActivity(double s)
    { score = s; }

    // Mutator function
    void setScore(double s)
    { score = s; }

    // Accessor functions
    double getScore() const
    { return score; }

    char getLetterGrade() const;
};
#endif
```

Polymorphism
(overloading)

inherited

GRADEACTIVITY (.cpp)

```
#include "C:\Users\David\Documents\Visual Studio
2010\Projects\chapter15b\chapter15b\GradedActivity.h"

//
// Member function GradedActivity::getLetterGrade
//

char GradedActivity::getLetterGrade() const
{
    char letterGrade; // To hold the letter grade

    if (score > 89)
        letterGrade = 'A';
    else if (score > 79)
        letterGrade = 'B';
    else if (score > 69)
        letterGrade = 'C';
    else if (score > 59)
        letterGrade = 'D';
    else
        letterGrade = 'F';
    return letterGrade;
}
```


MAIN PROGRAM (.cpp)

```
#include <iostream>
#include "GradedActivity.h"
using namespace std;

int main()
{
    double testScore;// To hold a test score

    // Create a GradedActivity object for the test.
    GradedActivity test;

    // Get a numeric test score from the user.
    cout << "Enter your numeric test score: ";
    cin >> testScore;

    // Store the numeric score in the test object.
    test.setScore(testScore);

    // Display the letter grade for the test.
    cout << "The grade for that test is "
    << test.getLetterGrade() << endl;

    return 0;
}
```

Program Output with example input Shown in Bold

Enter your numeric test score: **80 [Enter]**

The grade for that test is B

Program Output with Different Examples Input shown in Bold

Enter your numeric test score: **75 [Enter]**

The grade for that test is C

Ex: FinalExam class

Private members:

int numQuestions

declared in the FinalExam class

double pointsEach

declared in the FinalExam class

int numMissed

declared in the FinalExam class

Public members:

FinalExam()

defined in the FinalExam class

FinalExam(int, int)

defined in the FinalExam class

set(int, int)

defined in the FinalExam class

getNumQuestions()

defined in the FinalExam class

getPointsEach()

defined in the FinalExam class

getNumMissed()

defined in the FinalExam class

setScore()

inherited from GradedActivity

getScore()

inherited from GradedActivity

getLetterGrade()

inherited from GradedActivity

FINALEXAM (.h)

```
#ifndef FINALEXAM_H
#define FINALEXAM_H
#include "GradedActivity.h"

class FinalExam : public GradedActivity
{
private:
    int numQuestions; // Number of questions
    double pointsEach; // Points for each question
    int numMissed; // Number of questions missed
public:
    // Default constructor
    FinalExam()
    { numQuestions = 0;
      pointsEach = 0.0;
      numMissed = 0; }

    // Constructor
    FinalExam(int questions, int missed)
    { set(questions, missed); }


    // Mutator function
    void set(int, int); // Defined in FinalExam.cpp

    // Accessor functions
    double getNumQuestions() const
    { return numQuestions; }

    double getPointsEach() const
    { return pointsEach; }

    int getNumMissed() const
    { return numMissed; }
};
#endif
```

Declares FinalExam class
derived from GradedActivity



FINALEXAM (.cpp)

```
#include "FinalExam.h"

//
// Set function
// The parameters are the number of questions and the
// number of questions missed.

void FinalExam::set(int questions, int missed)
{
    double numericScore; // To hold the numeric score

    // Set the number of questions and number missed

    numQuestions = questions;
    numMissed = missed;

    // Calculate the points for each question.
    pointsEach = 100.0 / numQuestions;

    // Calculate the numeric score for this exam.
    numericScore = 100.0 - (missed * pointsEach);

    // Call the inherited setScore function to set
    // the numeric score.
    setScore(numericScore);
}
```

MAIN PROGRAM (.cpp)

```
#include <iostream>
#include <iomanip>
#include "FinalExam.h"
using namespace std;

int main()
{
    int questions;// Number of question on the exam
    int missed;// Number of questions missed by the student

    // Get the number of questions on the final exam.
    cout << "How many questions are on the final exam? ";
    cin >> questions;

    // Get the number of questions the student missed.
    cout << "How many questions did the student miss? ";
    cin >> missed;

    // Define a FinalExam object and initialize it with
    // the values entered.
    FinalExam test(questions, missed);

    // Display the test results.
    cout << setprecision(2);
    cout << "Product =: " << setprecision(3) <<
    test.getProduct() << " " << endl;
    cout << "\nEach question counts " << test.getPointsEach()
    << " points.\n";
    cout << "The exam score is " << test.getScore() << endl;
    cout << "The exam gade is " << test.getLetterGrade() <<
    endl;

    return 0;
}
```

Program Output with Example Input Shown in Bold
How many questions are on the final exam? **20 [Enter]**
How many questions did the student miss? **3 [Enter]**

Each question counts 5 points.
The exam score is 85
The exam grade is B

Overloading Constructors

The polymorphism feature of an object-oriented language allows the programmer to provide multiple definitions of an operator or for a function. Another term used to refer to polymorphism is overloading.

More than one function may have the same name, if the argument list is different. When the program is compiled the appropriate occurrence of the function will be linked based on the data type that is used in the call to the function.

Therefore it is possible to have multiple constructor functions within a single class.

Ex:

```
// Default constructor
GradedActivity()
{ score = 0.0; }

// Constructor
GradedActivity(double s)
{ score = s; }
```

Notice that both constructor functions have the same name `GradedActivity()`, the difference is within the parentheses. When an object is declared, the contents in the parentheses determine which function is called.

There may also be multiple functions with arguments, but the arguments are of different types or have a different number of arguments for overloaded functions.

Ex: `GradedActivity();`
 `GradedActivity(float);`
 `GradedActivity(int);`
 `GradedActivity(float, int);`
 `GradedActivity(int, float);`

All of the above are different and could be used within a single class or independently of classes. A function may be redefined with different argument lists using the same function name. It is possible to have different return types. However, the return types cannot be the only differing factor in the function declarations because the compiler would not be able to distinguish between the two functions at the point of the function call.