COM326 Object-Oriented Programming

Week 8 – Session 2: Advanced class features

In this laboratory session you are going to explore some advanced class features in OOP programming with C++.

Challenge 1: Preparation

Before you start examining the advanced class features of OOP with C++ you need to do some preparation.

Task 1 Get the source code - duration 5 minutes

Take yourself over to the lab solutions in GitHub and fork the solution for `StudentModule` into your own GitHub account. You can find the repository here:

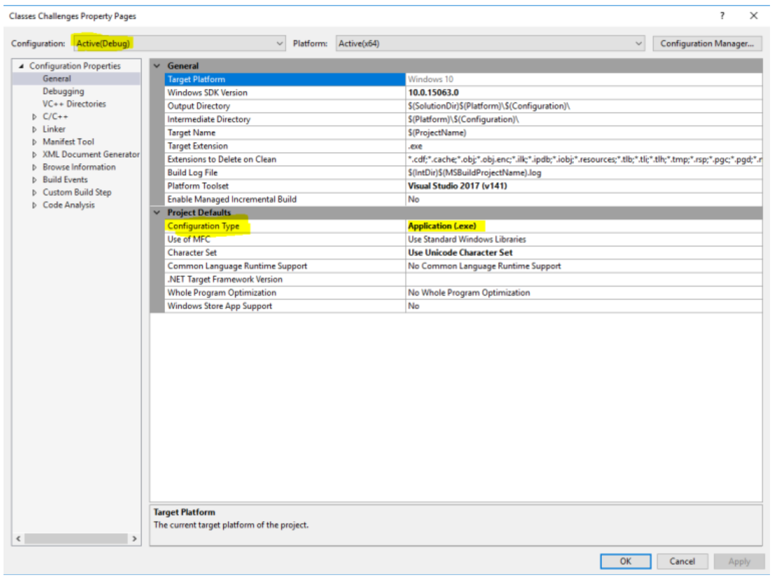
https://github.com/eiramlan/StudentModule

Task 2 Clone the repo - duration 10 minutes

You should clone the repository into your VS. You can either(1) Launch GitKraken and create a local clone (on the PC HD) of the repo from your GitHub account, or (2) Use the clone/check-out option directly in your VS, or (3) Download the zip and extract it locally. Remember where on the local PC you clone the repo to and please open the solution file (*.sln)

Task 3 Review the code - duration 10 minutes

On Tuesday, we focused on testing the student and module classes. Take ten minutes to explore the source code in the `Classes Challenges` project. You will need to change the classes project back to executables.



Classes Challenges Property Pages

Configuration: Active(Debug)   Platform: Active(x64)

General	
Target Platform	Windows 10
Windows SDK Version	10.0.15063.0
Output Directory	$(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory	$(Platform)\$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.r
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	Visual Studio 2017 (v141)
Enable Managed Incremental Build	No
Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use Standard Windows Libraries
Character Set	Use Unicode Character Set
Common Language Runtime Support	No Common Language Runtime Support
.NET Target Framework Version	
Whole Program Optimization	No Whole Program Optimization
Windows Store App Support	No

Target Platform
The current target platform of the project.

Figure 1: Change the output project settings for classes to produce an executable

Challenge 2: Constructors and destructors

Constructors and destructors bookend the lives of objects. Constructors are called when an object is instantiated, and destructors are called when objects are no longer required or go out of scope. In this challenge you will trace the calls to constructors and destructors.

Task 1 Add destructors - duration 15 minutes

Add destructors to both the module and student classes. Within each destructor write to the screen:

```
Destructor of the <insert class> class called on <name> / <title>
```

Replace `<name>` with "empty module/student" as appropriate if the name of the module or student is empty.

For example:

```
Destructor of the Module class called on Graphics programming
```

Task 2 Update constructors - duration 15 minutes

Update the constructors so that they output a similar message. If the student name or module title is available call it.

Task 3 Trace the calls - duration 15 minutes

Build and run the application while you examine the main.cpp. Can you explain why each of the constructor and destructor calls are being made? What about the order of the calls. Is the order correct?

Challenge 3: Static Class members

Add Static class members to the entire class. In this challenge you will learn how to define and use static member functions and data.

Task 1 How many students - duration 5 minutes

Define a variable call `int numberOfStudents_` as a static member of the class student. Write a static function called, `static int GetEnrolled()` that returns the number of student objects currently "alive" in the program. In order to count the number of students, we need to make use of the static member data, and increment it every time a constructor is call, and decrement it in destructor.

Challenge 4: Operator overloading

C++ allows you to overload the standard operators. In this this set of challenges you will overload some of the standard operators on the student class.

Task 1 Overload input operator - duration 20 minutes

Overload the input operator so that the user can input details for a student object. Once you implement the overloaded operator for input call it on a student object that has been created using the default constructor.

```
Student stu3{};
cin >> stu3;
```

Task 2 Overload output operator - duration 20 minutes

With our current implementation, if we want to output the details of a student then we need to call each of the getter methods within out cout statement like so:

```
cout << "Student 1 details" << endl << endl;
cout << "Name: " << stu1.GetName() << "Registration: " <<
stu1.GetRegistrationID() << "Course: " << stu1.GetCourse() << "Year
of study: " << stu1.GetYearofStudy() << endl << endl;
```

If we overload the output operator for the class, then we could just call this in our main like so:

```
cout << stu1;
```

Modify the student class and overload the output operator << so that it prints the details of the student as above.

Task 3 Overload the equality == operator - duration 10 minutes

Overload the equality operator so that you can check if two students are the same. Do not compare the modules that students are enrolled on. Just compare their name, registration ID, course name and year of study.

Task 4 Overload the increment ++ operator - duration 15 minutes

Overload the pre and post increment ++ operator so that you can automatically add module to student depending on their year. For instance, year 1 student will only have "Fundamental of Programming",

if we use `stu++;` or `++stu;`

we will be able to add module "`Object Oriented Programming`" – COM326 into their module list. Year 2 students will be added "`Data Structure – COM328`". Year of study will also increase accordingly.