LAB DEMO Constructors

Defining Classes

# Open previous project

1. Open the “StudentClassDemo” project.
2. View the student class.
3. Notice that we have one constructor defined.

# Paramaterless Constructor

1. Add a parameterless constructor:

//default constructor

public Student()

{

studentNumber = "Number Pending";

studentFirstName = "TBD";

studentLastName = "TBD";

major = "Undeclared";

}

1. Notice that this initializes some of our fields.
2. Replace the Main method with:

static void Main(string[] args)

{

Student myStudent = new Student();

Console.WriteLine("Student Number: {0}", myStudent.GetStudentNumber());

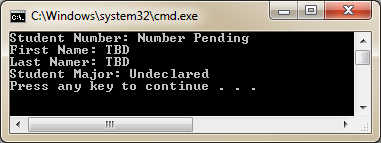
Console.WriteLine("First Name: {0}", myStudent.GetFirstName());

Console.WriteLine("Last Namer: {0}", myStudent.GetLastName());

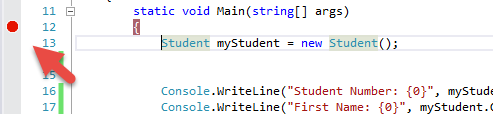
Console.WriteLine("Student Major: {0}", myStudent.GetMajor());

}

1. Test the code with Ctrl-F5 (Start without debugging).



1. Notice that all our fields have been initialized after calling the parameterless constructor.
2. Set a break point above the constructor and run it in debug mode (F5) and trace through what happens. Use F11 for “Step into” so you can watch control transfer from Main to the classes methods back and forth.



# Overloaded Constructors

1. Add the following overloaded constructors:

//single parameter constructor

public Student(string sID)

{

studentNumber = sID;

studentFirstName = "TBD";

studentLastName = "TBD";

major = "Undeclared";

}

//first overloaded constructor

public Student(string sID, string firstName, string lastName)

{

studentNumber = sID;

studentFirstName = firstName;

studentLastName = lastName;

major = "Undeclared";

}

//second overloaded constructor

public Student(string sID, string firstName, string lastName, int s1, int s2, int s3, string maj)

{

studentNumber = sID;

studentFirstName = firstName;

studentLastName = lastName;

score1 = s1;

score2 = s2;

score3 = s3;

major = maj;

}

1. Change main to:

static void Main(string[] args)

{

Student myStudent = new Student("S001","John","Smith",90,80,100,"Computer Programming");

Console.WriteLine("Student Number: {0}", myStudent.GetStudentNumber());

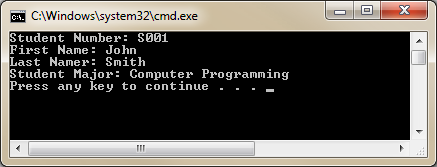
Console.WriteLine("First Name: {0}", myStudent.GetFirstName());

Console.WriteLine("Last Namer: {0}", myStudent.GetLastName());

Console.WriteLine("Student Major: {0}", myStudent.GetMajor());

}

1. Run with Ctrl-F5



1. Notice that the values we entered in the constructor now come out through our get methods.

# Chained Constructors

1. Change the constructors to:

//-------------------------------------

//Constructors

//-------------------------------------

//default constructor

public Student() : this ("Number Pending","TBD","TBD",-1,-1,-1,"Undeclared")

{

}

//single parameter constructor

public Student(string sID)

: this(sID, "TBD", "TBD", -1, -1, -1, "Undeclared")

{

}

//first overloaded constructor

public Student(string sID, string firstName, string lastName)

: this(sID, firstName, lastName, -1, -1, -1, "Undeclared")

{

}

//second overloaded constructor

public Student(string sID, string firstName, string lastName, int s1, int s2, int s3, string maj)

{

studentNumber = sID;

studentFirstName = firstName;

studentLastName = lastName;

score1 = s1;

score2 = s2;

score3 = s3;

major = maj;

}

1. What is the advantage of this approach? Will it be easier to maintain? Does it better implement the “Do not repeat yourself” or DRY concept?
2. Change main to:

static void Main(string[] args)

{

Student myStudent = new Student("S001","John","Smith");

Console.WriteLine("Student Number: {0}", myStudent.GetStudentNumber());

Console.WriteLine("First Name: {0}", myStudent.GetFirstName());

Console.WriteLine("Last Namer: {0}", myStudent.GetLastName());

Console.WriteLine("Student Major: {0}", myStudent.GetMajor());

}

1. Set a break point at the top of main and run in debug mode. Step through with F11 and watch how execution jumps from one constructor to the next.
2. Destructors
3. Add a destructor:

//-------------------------------------

//Destructor

//-------------------------------------

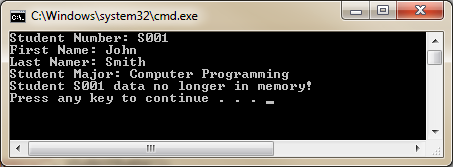
~Student()

{

Console.WriteLine("Student {0} data no longer in memory!", studentNumber);

}

1. Run with Ctrl-F5:



# Try To Override Finalize:

1. Comment out your destructor and try to enter the following code:

protected override void Finalize()

{

try

{

// Cleanup statements...

}

finally

{

base.Finalize();

}

}

1. What errors to do you get?
2. Do you get the impression Microsoft does not want you to touch the finalize method?