GRAND CIRCUS

C# .NET BOOTCAMP

ADVANCED C#

FILE I/O

FILE I/O IN C#

- C# has libraries to deal with data stored in file.
- When files are opened, we can read or write to them using streams
- A Stream is a sequence of bytes
- Main namespace: System.IO

FILE I/O CLASSES

- File: Static class that provides basic methods for creating, copying, deleting, and opening files.
- FileInfo: Can get information about files, and can be used to create file streams.
- FileStream: Provide a Stream to a file to do read and write.
- Directory: Static class that provides basic methods for creating, copying, deleting, and opening directories.
- draft

FILE I/O CLASSES

- DirectoryInfo: Can get information about directories.
- StreamReader: Provides support to read characters from a stream.
- StreamWriter: Provides support to write characters to a stream.

HOW TO GET INFORMATION ABOUT FILES AND DIRECTORIES

HOW TO WRITE TEXT TO FILES USING STREAMWRITER

```
er writer =  new StreamWriter("file1.txt");
te("Word ");
teLine("word 2");
teLine("Line");
```

HOW TO READ TEXT FROM FILES USING STREAMREADER

```
er reader = new StreamReader("file.txt");
e)
ne = reader.ReadLine();
= null)

teLine(line); // print line
```

TEST DRIVEN DEVELOPMENT (TDD)

WHAT IS TDD?

TDD is a method of developing software that involves a very quick cycle. It stresses simplicity of design as well as speed of implementation.

BASIC TDD STEPS:

1. Write a test
2. Run tests
3. Write implementation code
4. Run tests
5. Refactor

APPROACHING TESTING

TDD is not as focused as you may think on the tests themselves. It's more about having the confidence to reorganize our code knowing that all the previously completed features are working properly.

DOCUMENTATION USING TDD

Tests provide an excellent record of what the application should do as well as what it actually does.

TESTING OPTIONS

- Visual Studio's Test Explorer
- NUnit
- xUnit.net

VISUAL STUDIO TEST EXPLORER

provides overloaded assertion methods for all primitive types and objects.

VISUAL STUDIO TEST EXPLORER

EXAMPLES ON METHODS IN THE ASSERT CLASS:

- Assert.AreSame: Checks that they are the exact same object, memory reference and all.
- Assert.AreEqual: Checks that objectOne.Equals(objectTwo).

VISUAL STUDIO TEST EXPLORER

We can receive immediate feedback test results within VS.

VISUAL STUDIO TEST EXPLORER

DEMO!

TDD DEMO!

DESIGN PATTERNS

WHAT ARE DESIGN PATTERNS?

Design patterns are predefined solutions to general software development problems.

Design pattern solutions represent best practices.

Design patterns are tried and tested solutions developed by experienced object-oriented developers.

WHY USE DESIGN PATTERNS?

STANDARDIZE SOLUTIONS

A standard solution for a particular scenario allows developers to build an application around a commonly know pattern

WHY USE DESIGN PATTERNS?

BEST PRACTICES

Design patterns have been developed and tested by experienced developers over a long period of time and increases the speed and ease of software development of less experienced developers

MAIN DESIGN PATTERN TYPES

CREATIONAL PATTERNS

Provides a way to create objects without directly using the new operator.

STRUCTURAL PATTERNS

Defines new ways to compose objects to obtain new functionality.

BEHAVIORAL PATTERNS

Define ways to communicate between objects.

MAIN DESIGN PATTERN TYPES

BEHAVIORAL PATTERNS

Define ways to communicate between objects.

COMMON DESIGN PATTERNS

- Factory
- Singleton
- MVC

FACTORY PATTERN

This pattern can use an instance to create other objects.

In other words, we can design a class that will produce other objects of other classes.

FACTORY PATTERN IMPLEMENTATION

- Create an interface.
- Create concrete classes implementing the interface.
- Create a factory to generate an object based on a given parameter.
- Use a factory to get an object of the class by passing in the requested parameter type.

FACTORY PATTERN EXAMPLE

```
erface Shape {
;


s Rectangle:Shape {

void draw() {
e.WriteLine("Inside Rectangle draw()
");
```

FACTORY PATTERN EXAMPLE

```
;s ShapeFactory {
iape method to get object of type shape
)e getShape(string ShapeType){
:reate different object types
```

FACTORY PATTERN EXAMPLE

```
e1 = shapeFactory.getShape("CIRCLE");
r method of Circle
r();
bject of Rectangle and call its draw method.
e2 = shapeFactory.getShape("RECTANGLE");
r method of Rectangle
r();
```

SINGLETON PATTERN

The Singleton pattern enables programmers to control how many instances (mostly one) that are allowed to be made.

When those instances are created, we can get global access to them.

SINGLETON PATTERN IMPLEMENTATION

- Create an Singleton class
- Create Singleton static object declaration
- Make a private constructor
- Create a static get instance method

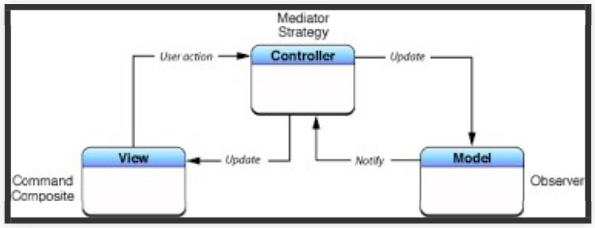
SINGLETON PATTERN EXAMPLE

```
ss SingleObject

tic SingleObject instance = new SingleObject();
gleObject(){}
ic SingleObject getInstance(){return instance;}
```

MVC PATTERN

The Model-View-Controller pattern is used to separate the layers of an interactive application



MVC PATTERN

MODEL

Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

VIEW

View represents the visualization of the data that model contains

MVC PATTERN

CONTROLLER

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT:

- What is a design pattern.
- Know when to use a design pattern.
- How three main design patterns.
- How how to implement the three main design patterns.