TRAILHEAD
TECHNOLOGY PARTNERS

.NET and C# Training

Session 2

Review Assignment 1

Assignment 01

- Create a .NET console “Hello world” application using the code below

```
using System;
namespace MyConsoleApp
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, world");
            Console.ReadKey();
        }
    }
}
```

- Run the application in the IDE
- Build the application and run the .EXE from outside the IDE
- Challenge: Add a Debug.WriteLine() statement to the code, run the application in the debuggger, and find your debug statement in the output pane

https://github.com/jonathantower/learning-dotnet

Today’s Agenda

1. Review Assignment 1 (15 min)
2. Object Oriented Programming Overview (45 min)
3. C# Introduction (2 hr)
4. Assignment 2 (15 min)

Today’s Agenda

1. Review Assignment 1 (15 min)
2. Object Oriented Programming Overview (30 min)
3. C# Introduction (3 hr) *
4. Assignment 2 (15 min)

* You’ll be able to get more hands-on with code once we finish basic C# (after next time?)

Object Oriented Programming

TRAILHEAD
TECHNOLOGY PARTNERS

What is Object Oriented Programming?

- Objects contain data and code
- Analogy: programs running on a computer (data and code in memory)
- Data in the form of fields and properties
- Code in the form of procedures/methods
- Unique functionality happens in how objects communicate with each other

What is Object Oriented Programming?

- Shared with procedural coding
  - Functions/procedures
  - Variables
  - Types
  - Parameters and return types
  - Loops and conditionals
- Switch is overwhelming at first
  - “Where should I draw the lines when splitting functions between classes?”

Objects and Classes

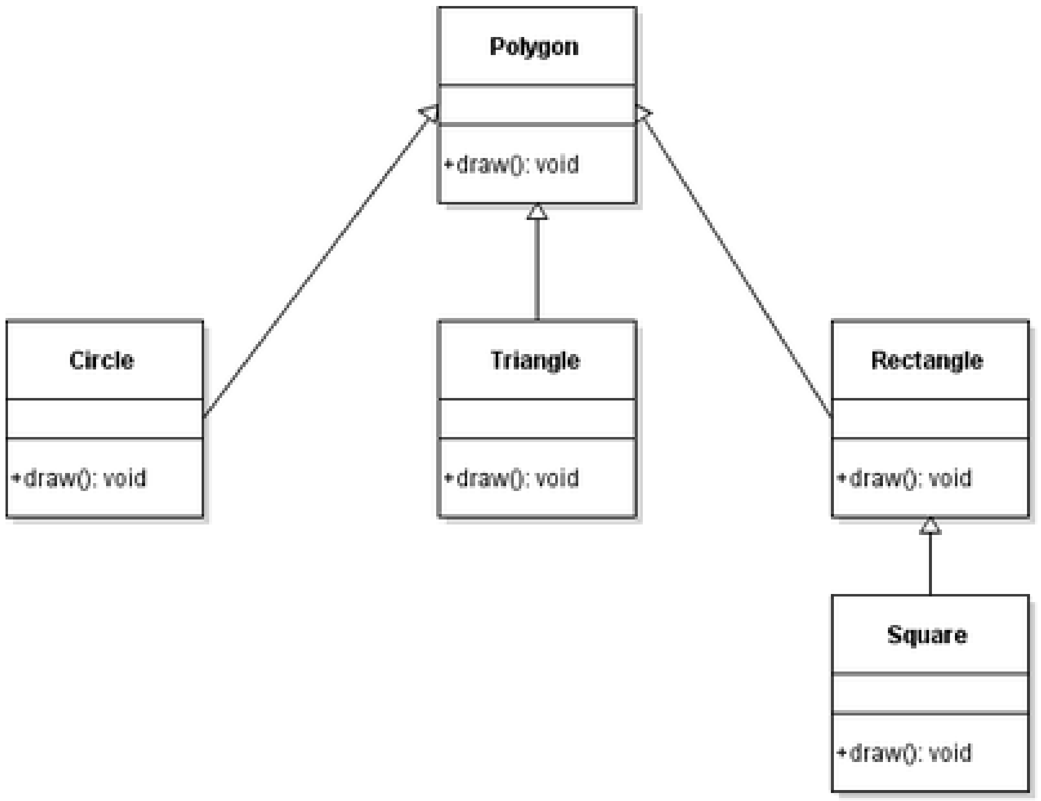
- Class = template for an object
- Object = instance of class
- Analogy
  - Cookie cutter :: cookies as class :: object
- Instantiation (“new”) is to create an instance
  - Uses a special method called a constructor

Encapsulation

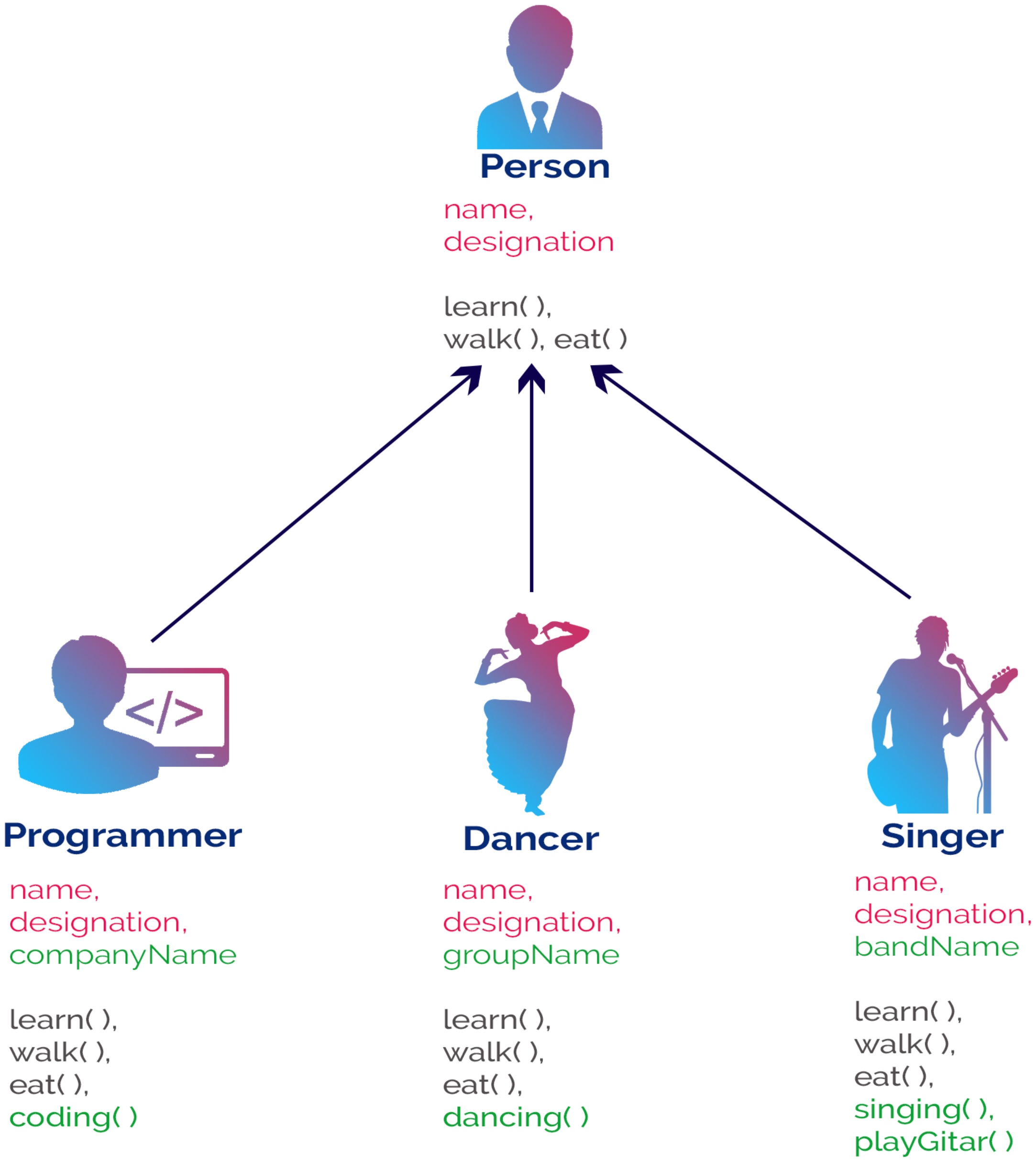
- “Black box” vs “white box”
- Analogy: your cable box
- Simplifies complexity
- Prevents external code from accessing internal implementations
- Allows for refactoring
- Single-responsibility principal

Inheritance

- Classes defined in hierarchies
- Can share data and implementation
- Can override or replace implementation



Inheritance



Person

name,
designation

learn( ),
walk( ), eat( )

Programmer

name,
designation,
companyName

learn( ),
walk( ),
eat( ),
coding( )

Dancer

name,
designation,
groupName

learn( ),
walk( ),
eat( ),
dancing( )

Singer

name,
designation,
bandName

learn( ),
walk( ),
eat( ),
singing( ),
playGitar( )

Source: http://www.btechsmartclass.com/java/java-inheritance-basics.html

Data Abstraction

- Objects show relevant data and actions (methods)
- Objects hide (or abstract) all unnecessary or internal data

Polymorphism

- Literally: “many shape”ism
- Related to inheritance
- Treat “this” as a more-general “that”

```
public class Shape { }
public class Circle : Shape { }
public class Square : Shape { }

public void Draw(Shape shape) { }

Circle circ = new Circle();
Draw(circ);
Circle sq = new Square();
Draw(sq);
```

TRAILHEAD
TECHNOLOGY PARTNERS

Open Recursion

- Using members of the same class
- this keyword
- Late-bound by compiler

SOLID Principals

Single-responsibility principle: "There should never be more than one reason for a class to change.” In other words, every class should have only one responsibility.

Open–closed principle: "Software entities ... should be open for extension, but closed for modification."

Liskov substitution principle: "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it." See also design by contract.

Interface segregation principle: "Many client-specific interfaces are better than one general-purpose interface."

Dependency inversion principle: "Depend upon abstractions, [not] concretions.”

C# Introduction

C# Type System

- Garbage collection
  - No alloc and dealloc
- No pointers (sort of)
- Value types vs reference types

C# Type System

- Value type examples:
  - bool
  - byte
  - char
  - decimal
  - double
  - enum
  - float
  - int
  - long
  - sbyte
  - short
  - struct
  - uint
  - ulong
  - ushort

C# Type System

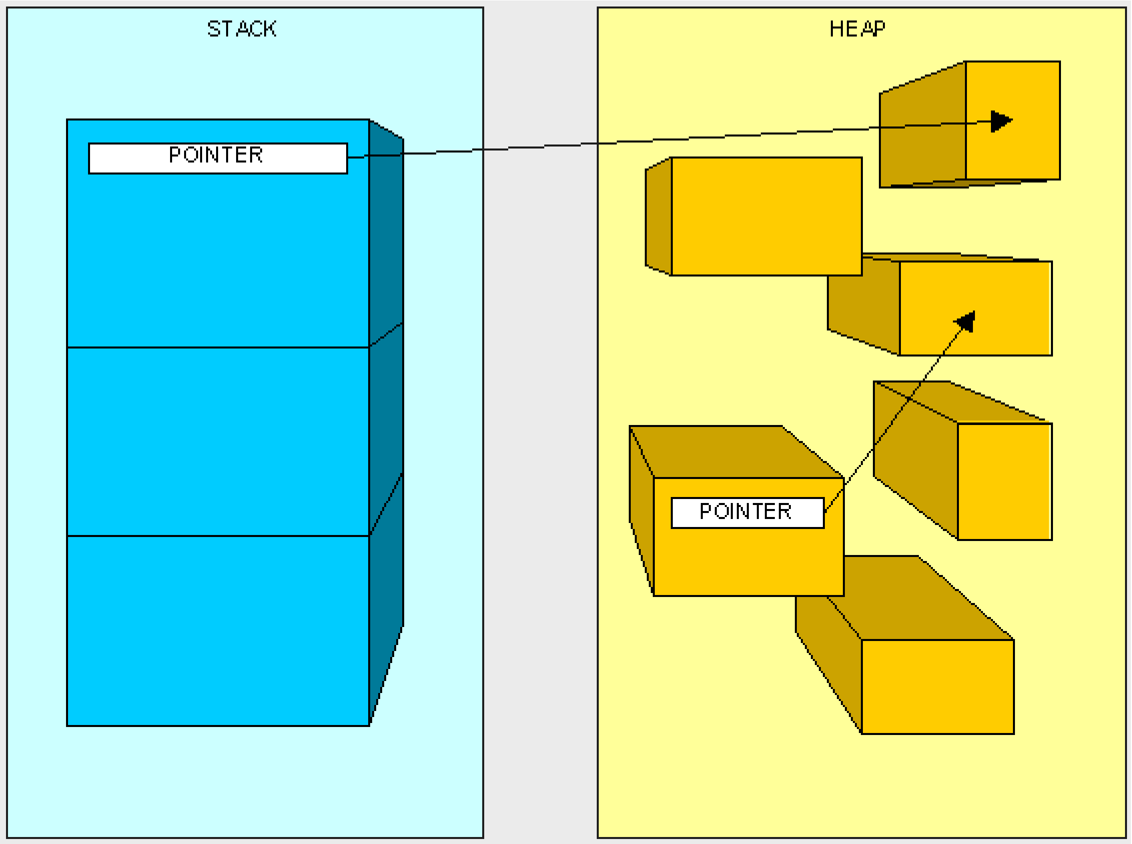
- Value type examples:
  - bool
  - byte
  - char
  - decimal
  - double
  - enum
  - float
  - int
  - long
  - sbyte
  - short
  - struct
  - uint
  - ulong
  - ushort

C# Type System

- Reference type examples:
  - Object
  - string
  - DateTime
  - dynamic
  - Anything you define with “class”

C# Type System

- Stack vs Heap



https://www.c-sharpcorner.com/article/C-Sharp-heaping-vs-stacking-in-net-part-i/

C# Type System

- A Reference Type always goes on the Heap
- Value types go where they are declared
- Implications:
- passing objects (reference types) is actually passing around a pointer to the object
- Passing values (value types) is actually passing around copies of the value

Demo

Structs for Object-like Value Types

Demo

Defining variables with types, var, and dynamic

Demo

Classes in C#

Demo

Constructors and Initializers

Demo

Fields & Properties

Demo

Private, Public, and Protected

Demo

Methods, Parameters, Output Parameters, and Return Types

Demo

Methods, Parameters, and Return Types

Demo

Output Parameters, Param keyword, Named Parameters, Optional Parameters

Demo

Interfaces

Demo

Class and Interface Inheritance

Demo

Abstract, Virtual, Overloading, Shadowing, etc

Demo

Static vs Instance Members

Demo

Polymorphism

Assignment 2

- Create a console application and use the following base class to define a Cat and Dog class that inherit from it

```
public abstract class Animal
{
    public abstract string Talk();
    public string Feed()
    {
        return Talk();
    }
}
```

- Implement the Talk() method in each class
- In the Main method, create dog and cat instances, feed them, and write out the output.
- Challenge 1: Add a protected field “energy’ to the Animal class. Increment it by 1 every time the cat is fed and 2 every time the dog is fed.
- Challenge 2: Add a method to Animal called GetEnergy() that returns the value of the field energy and output the animals new energy every time you feed them

https://github.com/jonathantower/learning-dotnet