# Module 1 Day 10

Week 2 Review

# Encapsulation

- The action of enclosing something in or as if ***in a capsule***
- From Wikipedia
  - A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.
  - A language mechanism for restricting direct access to some of the object's components.
- Mike's words
  - Bundling stuff together which goes together (as in classes)
    - Models real-world as closely as possible - **maintainable**
  - Not showing outsiders any more than they need to know (access modifiers)
    - **Loosely couples** your system; makes system more **flexible**

# Encapsulation

- Access modifiers help us encapsulate

```
public int Property { get; set; }          //public set
public int Property { get; }               //readonly set w/in constructor
public int Property { get; private set; } //private set
```

- Read-only Properties
  - Value can *only* be set by the object in its constructor
  - After that, the value cannot change
- Private methods also hide details

# Static

- Member belongs to the class/type, *not* to individual object instances
  - "Class variable or method" vs. "instance variable or method"
- Property or field
  - The data is stored once, regardless of how many instances there are
  - "Shared" by all instances
  - You don't need an instance (object) to access the property
    - Console.ForegroundColor
    - DateTime.Now
- Method
  - You don't need an instance (object) to access the method
    - Console.Writeline
    - Math.Round
- Class
  - Only has static members; cannot be created (new'd)

# Lecture Code Goals

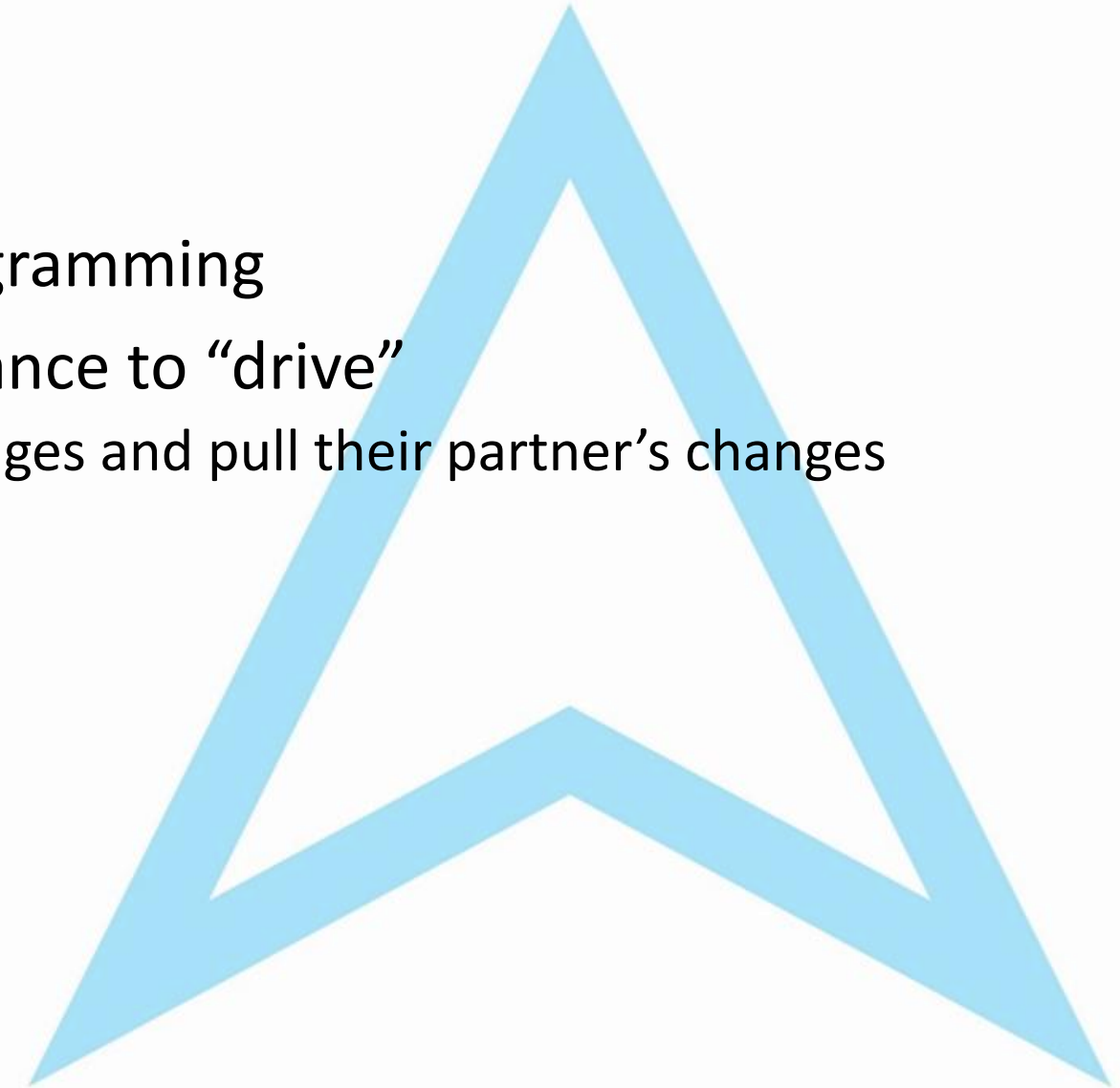

- Card
  - Create a Card class to represent a standard American playing card
  - Properly encapsulate the Card class
- Deck
  - Create a Deck class to represent a standard deck of cards
  - How should we encapsulate the members of the Deck?
- Card Game
  - Deal a hand to each of 2 players
  - Print out the two hands
- Relationships
  - As important as the classes themselves is how these classes relate to one another
    - Reference

# Card Game

- Design
  - What are the classes?
  - What are the properties and methods of each class? What is private and what is public?
  - What are the relationships between the classes?
- UML
  - Unified Modelling Language
  - Class Diagram expresses design

# Pairs Exercises

- Pairs assigned
- Clone your repository
- Pair programming vs. parallel programming
- Make sure both partners get a chance to “drive”
  - Each partner should push their changes and pull their partner’s changes



# Week 2 Pairs Exercise

| Team | Student | Room |
|---|---|---|
| 0 | Clayton Cross | EUCLID |
| 0 | Jess Wilson-Woodrow | |
| 1 | Ahmad Doleh | GARAGE |
| 1 | Timothy Becker | |
| 2 | Kameron Howes | GOSLING |
| 2 | Paul Vanden Broeck | |
| 3 | Amir Farhat | HOPPER |
| 3 | Andrew Dorscheid | |
| 4 | Aklile Kebede | JOHNSON |
| 4 | Edward Uhl | |
| 5 | Lucinda Verplanck | LOVELACE |
| 5 | Robin Blake | |
| 6 | Colin Leverette | ONTARIO |
| 6 | Marissa Buss | |
| 7 | Bill Michaels | PARTICIPATE |
| 7 | Dejan Cancar | |
| 8 | Brandon Oleksik | PROSPECT |
| 8 | Todd Zucker | |