**Class Notes 7**

Today's Agenda:
1. Epoch Dates
2. Q/A on Methods
3. Calendar Project Solution

## Epoch Dates

Different civilizations throughout human history have used different events to begin their record keeping, a point in time we call an *epoch*.

In China, the current year is 4716, the year of the Dog, which began on the Gregorian calendar on February 16, 2018. This year is derived from the recorded beginning of the first dynasty.

The Gregorian calendar, used today in most of the world, was not introduced until October 1582 by Pope Gregory XIII. It was introduced as a way to correct the issues with the Julian calendar, which had issues with leap years and the fact that the earth takes 365.2422 days to complete a revolution around the sun.

## Methods

It's entirely possible to call one method from another method. When we do this, we say that the called method is a *helper* method. A method that changes a variable is called a *mutator* or *modifier* method. A method that returns variable data is called an *accessor* method. These are informal names for these types of methods that aren't enforced by the programming language, but are useful when discussing how code works with other programmers. It is part of the *code conventions* that we discussed previously.

Making methods *private* prevents them from being used by other callers outside of the program. Usually, we will do this to protect or *encapsulate* the interior methods of our code. In the Solution for Calendar, I have made all of the methods *public* for testing purposes.

## Testing

Testing is broken up into two categories:

1. **Unit Testing** is where we examine individual functions and determine if they are accepting the proper kind of data, and returning the proper kind of data. You can imagine it as working with a kind of *black box*, where after having implemented the device, you no longer worry about how it is actually achieving its goal, but only care if the proper input/output is achieved.

2. **Integration Testing** is where we combine all of the functions that we have tested together and determine whether the complete system accomplishes the desired task.

--

In order for formally cover all the testing necessary, it is vital to come up with a list of **test cases.** This list of test cases should be comprehensive, and should verify every single possible input to the program, and resulting output. **Software Validation** is an important component of all modern software engineering. There is criticism of some current software companies for releasing software that is not completely finished, but requires many updates each week to fix bugs. On the flip side, this attention to updating can extend the life of a piece of software significantly.

For example, some video games have very short lifespans in the market. However, being able to update them with new content and gameplay can extend their sales life significantly...Blizzard Entertainment, the creators of Warcraft, Starcraft, and Diablo have enjoyed this sort of continued support of products they have made over 20+ years ago in this regard.

## Organization of Code

At this point, you realize that even if we break complex ideas down into multiple functions, we still end up with programs that can get into hundreds or thousands of lines of code. We still have the same problem as before...a single error in any portion of the program will cause the entire program to

fail to compile. This is akin to having an airplane where a light bulb goes out, and the entire plane falls from the sky and explodes.

**Object Oriented Programming**

OOP is a programming paradigm ("way of doing things") that is less focused on the order in which code is written, but is more concerned with the relationship that different parts of the code have with one another.

In OOP, we compartmentalize blocks of code in such a way that they conceal the details of their operation from one another, in a process called *encapsulation*. Encapsulation is a fancy word for "information hiding," and it is done not for malicious purposes, but to provide a way to allow programmers to focus on functionality as opposed to individual details.

**Classes**

We've been typing the word Class a lot...and I bet you've not had a lot of chance to consider why we keep using it.

A **Class** in Java is a lot of things:
1. the design for an object
2. container that stores variables and methods
3. abstract data type

Because a Class can store variables and methods, we can create **Objects** that are **instances** of each class.

In these objects, the variables determine the **state** and the methods determine the **behavior.**

From a single Class, one can instantiate multiple objects.

To summarize:

A **Class** is the design for an Object.
An **Object** is an instance of a class.
An **instance** is a single occurrence of something.

A Class is like the *blueprint* for an object. It does not actually exist on its own, but rather provides a design for us to create an instance of it.

An Object is the realization of the design, like a home that is built using an architectural blueprint. In the City of Irvine, we have many homes (the objects) that are all built using the exact same designs (the class), leading to many square miles of places that all look exactly the same, all buildings vaguely the same color of thousand-island dressing, trees of the same height, and appearance, disturbingly identical down to each leaf.

**Designing a Class**

We are going to design a class to represent a simple geometric concept: A circle.

What sort of variables does a circle have?

```
PI
radius
```

What sort of methods does a circle have?

```
getRadius()
setRadius( double newRadius )
area()
circumference()
```

## Method and Constructor Names

Methods and Constructors can share the same name as long as their parameters are different! When we have two methods or constructors with the same name, we say that they are *overloaded*.

## Methods and Behavior

Methods define the behavior of classes. When you write methods, you are determining what an object instantiated from a class is capable of doing. As a Java programmer, your primary job is to write methods!

## Programming Exercise No. 1

Write a class that defines a Rectangle.
Provide the required variables for the object.
Provide a single constructor that accepts the variables to be saved.
Provide methods that get the variables, set the variables, and calculate the area and perimeter.

*Why do we make instance variables private?*

We keep instance variables private whenever possible to prevent them by being changed by programs outside of the class/object. One of the more important principles of designing classes is not only to hide the data, but to ensure that the only way that the data is changed is by the methods that you provide...thus, **you** define the behavior that can change the state of the object.

**Programming Exercise No. 2**

Create a class that defines an equilateral **n-gon**. An equilateral n-gon is a polygon with n number of equal length sides, with n being greater than 2.

One should be able to instantiate this object using two variables in the constructor:
n: number of sides
size: length of the side

Your methods should include get and set methods for n and size, as well as a method to determine the sum of the interior angles of the polygon. The formula for calculating the of the interior angles of a regular polygon is (n-2) * 180 degrees.

**Client Code**

Client Code is where we actually interact with the classes and objects that we have created. This is where the **main()** function goes in a program!

You can see here that the classes we have created are used as Abstract Data Types (or ADT's), similar to the Scanner and Random classes we have already been using.

The *new* keyboard calls the constructor of a class in order to generate a single new instance of the class, called an object.

**HOMEWORK**

Part 1:
Read the textbook excerpt on methods and the introductory material for classes and objects.

Part 2:
Write a class that is the design for a robot.
The robot should have a name, age, and favorite color.

The robot's behavior should include sharing its name, age, and favorite color, as well as functions to change the robot's name and favorite color. There should also be:

1. A random greeting method by the robot, selected from four different greeting messages.

2. a method that calls for a ray gun effect where the robot tries to vaporize things (just print something to screen like PEW PEW PEW).

3. a method that implements some kind of algorithm of your choosing to determine the robot's current mood. Select from happy, sad, angry, depressed, psychotic, apathetic. There should be a supporting method that returns the robot's mood as well.

When complete, please provide client code that instantiates a new robot, and tests all of its methods.