Classes Deep Dive

- **Review of Previous Week**
- **Revisit Class Definition**
- **Revisit Method Definition**
- **References**
- **Encapsulation**
- **this and null keywords**
- **Initial Object State**
- **Homework**

But first a review of last week

`git add`: Use this command to tell Git to start tracking the changes you make to a file. Example: git add HelloWorld.java

`git commit`: Use this command once you are done with a set of updates to your code and you are ready to package them so that you can share them with others. git commit -m "Fixing a bug"

`git push` : Use this command to make your commits available to the rest of your team. Example: git push origin master

`git status` : Use this command to get information about your repository. Tells you what files Git is not tracking and which ones you have modified.

`git clone` : Use this command to download a copy of a repository to your own computer. Only need to do this once.

Revisit Class Definition

`access modifier` class `Name`

{

`class body`

}

```
public class Ball {
    private double xVelocity;
    private double yVelocity;

    public double getyVelocity() {
        return yVelocity;
    }

    public double getxVelocity() {
        return xVelocity;
    }
}
```

Revisit Method Definition

`access modifier` `return type` `name` (`parameter list`)

{

`method body`

}

```
    public void incrementAngle() {
        angle = angle + 1;
    }
```

`Video reference` : **Method Basics**: 2:05 - 2:30

Exiting Methods

3 Ways to exit a method:

- the end of the method is reached
- encounter a return statement
- exception thrown

```
private Game createGame() {
        if(alreadyCreated){
                return existingGame;
        }
        Ball ball = new Ball();
        Player player = new Player(200);
        player.hold(ball);
        return new Game(player, ball, new Hoop());
    }
```

References

References point to a place in memory, they are not values in themselves.

`Video reference`: **Using Classes**: 1:30 - 2:55

Encapsulation

Users of a class should be concerned with what they want the class to do, not how the class will do it.

Control Acess to data and methods using **Access Modifiers**

- no modifier (called package private)
- public
- private

`Video reference`: **Applying Access Modifiers**: 1:05 - 2:10

Accessors and Mutators

Common practice to make fields private and create methods to access and update them

```
public class Player {
    private double yVelocity = 0;

    public double getyVelocity() {
        return yVelocity;
    }

    public void setyVelocity(double yVelocity) {
        this.yVelocity = yVelocity;
    }
}
```

this and null

the `this` keyword is a reference to the current object.

`Video reference`: **Special References: this and null**: 0:30 - 1:25

the `null` keyword indicates an uncreated object

Initial Object State

- **Field initial state**
- **Field Initializer**

```
public class AmountLimiter {
    private int currentAmount;
    private int maxAmount = 10;
    private int minAmount = calcMinAmount();

    private int calcMinAmount(){
        return -10;
    }
}
```

Constructors

- **Implicit empty constructor**
- **constructors can be chained**

```
public class GameEngine {
    private GravityEngine gravityEngine;
    private MovementEngine movementEngine;

    public GameEngine(GravityEngine gravityEngine,
        MovementEngine movementEngine) {
        this.gravityEngine = gravityEngine;
        this.movementEngine = movementEngine;
    }
}
```

`Video reference` : **Chaining Constructors**: 1:35 - 2:20

Initialization Blocks

- Shared across all constructors
- Executed as if the code were placed at the start of each constructor
- Enclose statments in brackets outside of any method or constructor

`Video reference`: **Initialization Blocks**: 2:40 - 3:40

Summary

- A class is a template for creating an object
- Classes are reference types
- Use access modifiers to control encapsulation
- Methods manipulate state and perform operations
- Fields store object state
- Field initializers provide an initial value as part of the declaration

Summary - Continued

- Every class has at least 1 constructor
- If no explicit constructor, java provides one with no arguments
- You can provide Multiple constructors with differing parameter lists
- One constructor can call another
- Must be first line of constructor
- Initialization blocks share code across constructors