Casino Walkthrough

The goal this week is for us to work on all the skills we have covered over the past 2 weeks in one project.

The project we are going to walkthrough and create is the Casino project with three games in it.

1. Craps
2. Blackjack
3. Go Fish

The first thing we need to do is to create a fork of the project we want to work on:

Git

Step 1: Lets go to the following url: https://github.com/Zipcoder/TC-Casino

Step 2: Lets create a fork of that repository by clicking the “Fork” button

Open Source

The term “open source” refers to something people can modify and share because its design is publicly accessible.

The term originated in the context of software development to designate a specific approach to creating computer programs. Today, however, “open source” designates a broader set of values — what we call “the open source way.” Open source projects, products, or initiatives embrace and celebrate the principles of open exchange, collaborative participation, rapid prototyping, transparency, meritocracy, and community- oriented development.

Open source software is software with source code that anyone can inspect, modify, and enhance. This is a key concept in “Agile Development” and the Community Coding process that a lot of companies have adopted.

Even though the source code you maybe working on will not be available to the public, a lot of the modules that you incorporate into your programs will consist of open source projects. For example “JUNIT”.

The purpose of using repositories is so that when working on a project with a group of developers, you can

control the creation process, and keep a history of the entire life span of a project. To contribute to a project directly, you would have to be given access and permissions to make changes to the state of the project.

For projects that you do not have access to contribute directly to, you can “FORK” the repository. Creating a fork of a repo will take a snap shot of the code base in its current state at the time you create the fork, and make an exact copy of that repository on your account in GITHUB. The copy of the project on your account you have complete access to make changes to and modify.

Step 3: Cloning the Repository

When you create a repository on GitHub, it exists as a remote repository. You can clone your repository to create a local copy on your computer and sync between the two locations. This procedure assumes you have already created a repository on GitHub, or have an existing repository owned by someone else you'd like to contribute to.

1. On GitHub, navigate to the main page of the repository.
2. Under the repository name, click Clone or download.
3. In the Clone with HTTPs section, click to copy the clone URL for the repository.
4. Open Terminal.
5. Change the current working directory to the ‘dev’ folder in your home directory.
6. Type git clone git clone https://github.com/YOUR-USERNAME/TC-Casino

Lets get to work

When you open the project there should be two files for us to start to work with.

1. Casino.java
2. CasinoTest.java

A couple of points to cover here:

What you are training to do right now is Object Oriented Programming (OOP). Software development can be very difficult to do, and also to explain. One of the ways we can simplify the process is thinking with the problems we face in software development in the same manor that we think about problems in real life. As you start to deconstruct the problem into smaller more manageable components, it is easier to think about those in the context of the problem we are trying to solve.

If we look in our project we should see a this file structure folders:

- src
    - main

        - java
            - io.zipcoder.casino
                - Casino.java

        - test
            - io.zipcoder.casino
                - CasinoTest.java

Here we have two classes Casino and CasinoTest.java.

Casino.java - This is the main file for our application, everything will start here. Any objects that we need to have at the very beginning of our application will have to be created here for the application to do its job.

CasinoTest.java - Let me be clear. EVERY CLASS that you create needs to have a corresponding Test class with it. Every class that you create (Atleast for the time being) should be in a file by itself and be paired with a matching Testing file.

So if you are creating an class called “RocketShip” it would be in a file called “RocketShip.java”

- src
    - main
        - java
            - flight
                - RocketShip.java

You would then immediately create a testing file called “RocketShipTest” which would be in a file called “RocketShipTest.java”

- src
    - main
        - test
            - flight
                - RocketShipTest.java

Make sure that you do not put your testing files in the same place as your application files. When you eventually send your application to production, you won’t need to have the testing files there. It will make the overall size of your application smaller. Most of the time you will have more code in your unit test than in your actual application.

Why do we test?

Object oriented programing is all about having objects talking to other objects to do a job. The same way that in

the real world we have to talk to each other to complete a task, we want our objects to do the same thing. In doing this we can segment our code into logical components, and isolate functionality. This is called Single Responsibility every object we create will be in charge of one thing, and one thing only.

If we are creating a application to represent office appliances , and we need to have the ability to perform the following tasks:

- Making Phone Calls
- Printing documents
- Copying documents
- Faxing documents
- Making Coffee

We would create the following objects to do those tasks:

- CoffeeMachine
- Telephone
- MultiPrinter

We would then assign the tasks to the objects that make the most sense:

- CoffeeMachine
    - making coffee

- Telephone
    - making phone calls

- MultiPrinter
    - Printing documents
    - Copying documents
    - Faxing documents

This is an example of Single Responsibility , we assign the responsibility for tasks to the object that is the most logical.

Now I know what you are thinking MultiPrinter is doing more than one thing, in fact its doing three things, how is that single responsibility? Single Responsibility doesn't mean our objects can only do one thing, it means that one single object is going to be responsible for handling this task.

Even though it is completely possible for you to create a class called CoffeeMachine and give it the ability to fax documents. The people using your application would be confused as to why the functionality for sending a fax is located in a object called of the type CoffeeMachine. This wouldn't happen in the real world, so it should

not happen in your code. The way we name things are context clues for the people using our applications, and also the people who will be working along with us as we code for what the intent of our code is. Everything has a place , and a place for everything.

So lets get started and write some code !!!

We are creating a text based console game, all the interaction with the user will be done via the console screen. So lets try getting some input and output going.

Currently in our application all we have in our Casino.java file is the following

```
public class Casino{

}
```

We can not even run this file as it stands right now.

Every java applications starts from a main method. If your program doesn't have a class that contains the has a main method, there is no way for you start the program. (At least at this point in time)

A method is a command that we can ask an object to perform. Each of these methods and commands have a unique signature that distinguish them from each other. The signature for a main method is as follows:

```
public static void main(String[] args){

}
```

Lets try and understand whats going on here.

public

public - this an accessor key word, if something is declared as public, that means any other object that sees it no matter what TYPE/CLASS it is has the right to interact with it. Declaring the main method as public allows any object in memory to call it.

static

static - This is a hard thing to wrap your head around in the beginning of learning java. The technical meaning of static is that you are declaring the field or method as a class level declaration , and does not belong to an instance of an object. WTF??!?? (What The Freckle)

Okay , lets look at it this way. The purpose of a class is to create a blue print for our program to create objects.

There are going to be times when we need to have some functionality that exist not only before we create and object, but also after the object leaves memory and is destroyed.

So lets look at what we are trying to do here, we want to create a application that simulates a casino. The casino in the real world is a building, to play the games the players have to enter the casino. When the player is done playing the game, they exit the casino. All of the players are individuals, and do not need to have any knowledge of each other, in fact they may not even be in the casino at the same exact times.

When a player enters the casino , lets consider that as the player is entering memory. When the player leaves the casino lets consider that as the player exiting memory. The casino needs to exist regardless if a player ever enters. The idea of the casino is static in the context of the players. There will be only one casino regardless of the number of players we have.

If we have 1 player we will have 1 casino. If we have 100 players we will have 1 casino.

This represents a one to many relationship.

```
public static Casino casino = new Casino();
```

The keyword static means that as soon as the application starts, before we create any objects this method is ready to go, and ready to do work!

void

void - every method we call is here to do a job. The expectation is that when a job is completed that an object representing the sum of that job completed.

If this was a mathematical equation f(x) = 4 + x.

4 + x is the function or job that we need to complete.

f(x) is the placeholder for the result of the completed job once x is defined.

When x = 4 , then the function can run and we get this f(4) = 4 + 4 f(4) = 8 8

Every method needs to return something! Even if what it return is nothing. It needs to let other objects know not to expect something. This is done with the key word void, when a method is declared as void this means that it will not produce anything, it will simply complete the task without passing back anything.

Our main method represents the start and end of our program, it is also a static method, so it exist before we create any objects, and will end once there are no more objects in memory. So there is no point of returning anything, because there is nothing left in memory to pass anything to.

main

main- is the name of the method, how can we call something if it doesn't have a name…. DUH!

(String[] args)

(String[] args) - anything inbetween the brackets of a method represents the objects that our method needs to complete its job.

If I was to ask you to go to the store and pick me up everything on a list of items, you could not successfully complete the task without the list of items. Your method signature would look something like the following:

```
public Food[] goToStore(String list)
```

The brackets '[]' means its an array of that TYPE so the FOOD[] is saying that this function will return an array that holds objects of the TYPE/Class FOOD.

When the method **goToStore** is called , to complete its job it needs object called list of **TYPE/Class String**. If we sent you to the store without the list, you would fail to complete the task, because you did not have the list. So we send the list when we call the method, so that it can complete its job.

When the main method is called you have an option to send in arguments at the beginning of the program hence why the signature contains (String[] args).

Lets create a new class called Application.java

- src
  - main
    - java
      - io.zipcoder.casino
        - Application.java

Inside **Application.java** lets create a main method

```
package io.zipcoder.casino;

public class Application {

    public static Casino casino = new Casino();

    public static void main(String[] args) {
        System.out.println("Welcome to the casino!");
    }
}
```

Try running the application now. It should display the message and exit.

Now lets get to work : When you complete each task get it approved by an Instructor before moving to the next task!

Task 1: Create a class called **Player** and a class called **PlayerTest** (make sure you place Player in the right package, and PlayerTest in the testing package)

Task 2: Inside of the **Player** class

- A **private** reference called **name** of type **String**
- A **private** reference called **money** of type **Double**
- A constructor that takes two parameters
    - A reference called **name** of type **String**
    - A reference called **money** of type **Double**
    - Then sets the local values of **name** and **money** to the global values of **name** and **money** (Think of the **this** keyword)

Task 3:

Inside of the **PlayerTest** class

- A **public** method that returns nothing called **getNameTest** that when given a **Player** object it test to see if the **getName** method works correctly.

Inside the **Player** class

- A **public** method that will return the name of the player called **getName** that takes no parameter references and returns a object of type **String**

Task 4:

Inside of the **PlayerTest** class

- A **public** method that returns nothing called **setNameTest** that when given a **Player** object it test to see if the **setName** method works correctly.

Inside the **Player** class

- A **public** method that will set the name of the player called **setName** that takes a parameter called **name** of type **String** and returns **nothing**.

Task 5:

Inside of the **PlayerTest** class

- A **public** method that returns nothing called **getMoneyTest** that when given a **Player** object it test to see if the **getMoney** method works correctly.

Inside the **Player** class

- A **public** method that will return the amount of money the player has called **getMoney** that takes no parameter references and returns a object of type **Double**.

Task 6:

Inside of the **PlayerTest** class

- A **public** method that returns nothing called **setMoneyTest** that when given a **Player** object it test to see if the **setMoney** method works correctly.

Inside the **Player** class

- A **public** method that will set the amount of money the current player has called **setMoney** that takes a parameter reference called **money** of type **Double** and returns **nothing**.

Task 7:

Inside of the **CasinoTest** class

- A **public** method that returns nothing called **createPlayerTest** that when given a **Casino** object it test to see if the **createPlayer** method works correctly.
    - You will need to figure out how to test using a Scanner object. Its a little tricky

Inside of the **Casino** class

- A **public** method that will create a new **Player** object called **createPlayer** that takes no parameters and returns a **Player** object.
    - This method should prompt the user to enter their name, and how much money they have.
    - It should not allow the user to have more than 1000 dollars.
    - If the user enters more than 1000 dollars it should set the value to 1000 dollars and prompt the user that they can only accept up to 1000 dollars.
    - To get the user to enter their username you will need a Scanner object. The issue here is how do you write your method so that its testable. Here is a hint :

```
public class MyClass {
    private InputStream systemIn;

     // default constructor
    public MyClass() {
        this(System.in);
    }

    // overloaded constructor
    public MyClass(InputStream in) {
        systemIn = in;
    }
}
```