**Casino with Methods and a Class**

**Understand the Application**

**What it Looks Like to the User**

The program will loop, asking the user for a bet amount from 0 to 100 (assume dollars, you can use **ints** or **longs**).  If the user types a 0 that means she wants to quit.  Otherwise, accept the amount as their bet and simulate a ***slot machine pull***.  Your program will print out a line that looks like a slot machine result containing three strings.  Some examples are:**BAR  7  BAR, 7  7  cherries, cherries  BAR  space, space  BAR  BAR, or cherries  cherries  BAR**.

* Each of the three positions in the string could be one of the following:  "**BAR**", "**7**", "**cherries**" or "**space**".
* Each of the three output positions must be generated by your program randomly with probabilities:
  + **BAR**          1/2   (50%)
  + **cherries** 1/4  (25%)
  + **space**       1/8  (12.5%)
  + **7**               1/8  (12.5%)
  + Therefore, **BAR** should be the most frequent symbol seen and **space** or **7** the least frequent.
* The following combinations should pay the bet as shown (note ***ORDER MATTERS***):
  + **cherries  [not cherries]  [any]** pays **5** × **bet** (5 ***times*** the bet)
  + **cherries  cherries  [not cherries]** pays **15** × **bet**
  + **cherries  cherries  cherries** pays **30** × **bet**
  + **BAR  BAR  BAR**pays **50** × **bet**
  + **7  7  7** pays **100** × **bet**
* After the pull, display the three strings regardless of the outcome.  If the user did not win, tell him/her "Sorry, you lose.”  If he won, pay him by displaying his winnings (his original bet ***times*** the winning factor from the above table).  Then, repeat the whole process by requesting another bet amount.

**Position counts!** If you read the above bullet that contains the warning "ORDER MATTERS", you will see that **cherries bar cherries** pays 5× while **cherries cherries bar** pays 15× and **bar cherries cherries** pays nothing.

**A Class that will keep track of the three strings:  TripleString**

We create a new data type to use for this assignment: class **TripleString**.  **TripleString**will consist of three private member **Strings** as its basic data: (**string1**, **string2**, and **string3**).  There will be few instance methods to support that data. The class will be very modest.  Once defined, we will use it to instantiate **TripleString** objects that can be used in our **main()**method and/or the static methods that **main()** invokes to simulate this casino project.

**The Methods to help the main().**

Each static method should be in the main class.  Here are the one that you have to write to simulate this casino app plays a special role.  For example, there will be one method that gets the bet from the user and returns it to **main()**:

public static int **getBet()**

Another method will simulate a random pull of the slot machine -- it generates three random strings and returns them as a **TripleString** object to **main()**:

public static **TripleString pull()**

An output method will be used at the end of each loop-pass when the user needs to see the results of her pull, and receive the news about how much she won (or not):

public static void display **(TripleString thePull, int winnings )**

We will describe each method -- and a few others -- in the next section.

\*\*There will be one main class and one other classes.  Submit only one .txt file that has all classes and output.

**The Program Spec**

**Class TripleString Spec**

The first step in writing this program is to create a simple, working class **TripleString**.

**The Data**

It will contain three private member **Strings** as its main data: **string1**, **string2**, and **string3**.   We will also add a public static member which is to be a final int **MAX\_LEN** set to 20. This represents the maximum length that our class will allow any of its strings to be set to.  We can use **MAX\_LEN** in the **TripleString** method whose job it is to test for valid strings (see below).

Additionally, we want to keep track of the winnings in an array and then print them out at the end of the program.  The static int array will be called **pullWinnings** and have a size equal to **MAX\_PULLS**(a static final int), which will be set to 40.

We will also need to keep track of which spot we are on for the array.  This needs to be a static variable since separate instances of the **TripleString** class will be created.  It will be an int called **numPulls.**

In summary, three private instances strings, one private static int array, one private static int variable, and two public static final ints.  That's the data for this class.

**Default Constructor**

TripleString() -- a default constructor that initializes all string members to "" and declare the size of the **pullWinnings** array. We do not need any parameter-taking constructors.

**A Private Helper Method**

**boolean validString( String str )** -- a **private** helper function that the mutators can use to determine whether a **String** is legal. This method returns **true** if both the string  is **not null** and its **length <= MAX\_LEN** and **false**, otherwise.

**Mutators and Accessors**

You need to create one mutator and one accessor for each of the private string member variables. The Mutator should return a boolean according to whether it was successful in updating the private member variable.  The Mutator should call**validString()** and only update the private member variable with the incoming data if it is good.

**toString() method**

Create a **toString()** method that will return all of the strings as one string.

**Winnings Methods**

Create two more methods for the **pullWinnings** array.  One will save the winnings from the round, **saveWinnings(int winnings)**, and the other will use a loop to get the values out of the array  and return a string, **displayWinnings()**.

**Where it All Goes**

You can create the **TripleString** class as a non-public class directly in your client **Assig2.java** file. You type it directly into that file;*do not ask Eclipse to create a new class for you or it will generate a second .java file which we don't want right now.* In other words, the file will look like this:

import java.util.\*;

import java.lang.Math;

public class Assig2

{

// main class stuff ...

}

class TripleString

{

// TripleString class stuff ...

}

As you see, **TripleString** is to be defined ***after***, not *within*, the **Assig2** class definition. This makes it a sibling class, capable of being used by any other classes in the file  (of which there happens to be only one: **Assig2**).

After writing this class, test it using a simple **main()** which instantiates an object, mutates the members, displays the object, etc.  Don't turn this test in.  It's part of your development cycle.

**The Assig2 Static Method Specs**

**int getBet()**

This prompts the user for input and returns the bet amount as a functional return.  It should validate the amount before it returns and insist on a legal bet (0 < **bet** < 100) until it gets one from the user.  It must return the legal value to the client and not take any other action besides getting the legal amount.

**TripleString pull()**

This method instantiates and returns a **TripleString** object to the client.   The data of the **TripleString** object has to first be filled with three randomly chosen strings according to the probabilities described in the "Understand the Application" section above. For example, it might return a **TripleString** object that contains the three strings **["cherries",  "BAR" , "space"]**.

The way it determines and loads the three strings is by using a private ***helper method***, described, next, **randString()**.  So this method, **pull()** will call the next method **randString()** three times to get the three strings that will be stored into the**TripleString** object.  Once that's done, **pull()** just returns the **TripleString** object to the client and its job is done.

**String randString()**

This **private** helper method does a little work -- yet is still quite short.  It produces and returns a single random string based on the required probabilities.  It does this by calling the java **Math.random()** function and using the return result of that function as a means of deciding which of the four possible strings to return.  Take this in stages.  **Math.random()** returns a double between 0 and 1.  One idea (but not the only one) is to turn that **double** into an **int**  between 1 and 100 using technqiues from five weeks ago.  Then, decide which of those numbers should trigger a "7", which should trigger a "cherries", etc. based on the desired probabilities.  Since a "Bar" should happen half the time, which numbers would you want to trigger a "Bar"?  Since a "cherries" should happen 25% of the time, which numbers would trigger a "cherries"?  So you see, this is a very simple -- and even short -- function, even though it has to be designed carefully.  Common sense will go a long way here.

**int getPayMultiplier (TripleString thePull)**

After **main()** gets a **TripleString** object from **pull()**, which we will call **pullString**, it needs to know what the payout will be.  That's the job of this function.  **getPayMultiplier()** takes the **pullString**as a parameter, and inspects it to determine what its pay multiplier should be:  5?  15?  100?  0?   It does this by looking at the three strings inside the passed-in **TripleString** object and using *if statements* to determine and return the right value.  For example, if all three of the strings are "cherries", which is easily checked using an*if statement*, then this method returns a pay multiplier of 30.  You can use logic like this to create a sequence of *if* or *else if* statements that will give you the desired multiplier.   However you do it, the method will return one of the values;  0, 5, 15, 30, 50 or 100.

**void display (TripleString thePull, int winnings )**

This method takes the winnings (a dollar amount) and **thePull** as parameters and displays the three strings inside **thePull** along with "  sorry - you lost " or "congrats, you won $X".

**main()'s Workflow**

You can debug each of the above methods individually using a***test main()*** that consists of a statement or two.  That way you will make sure each component works before trying to write the final **main()** client.

**main()** will be a loop controlled by value returned from **getBet()**.  As long as that value is non-zero, we keep playing.

Each time through the loop, we have to call **pull()** to get the **pullString** as a return value. Then we need to pass that to **getPayMultiplier()** to find the multiplier.  We then compute the winnings based on the previous information, and finally we display it all using **display()**.  That's all that each loop pass does.  So **main()**is quite neat and clean.

**Input Errors**

The only place the user can make an input error is in **getBet()**, so that's the method that deals with such errors.  Don't worry about non-numbers.  Assume that a number was entered.  But do test for range and only return to ***main*** after you have a valid range. **getBet()** may not decide about ending the program.  That's up to **main()**.

**Test Run Requirements**:

Submit one run that lasts about 20 to 40 pulls -- enough to see a few wins.  At least once enter an illegal amount to make sure that your program handles it correctly.

**General Requirements**

Communicate all values as ***parameters*** or ***return values***, not through ***globals*** (static class variables). The meaning of these terms and examples are contained in the reading.

Also, I will emphasize that in keeping with the separation of I/O and computation, we would not have any method other than **display()** output results to the screen, and **display()** is called from **main()**, not from any other method. Similarly,**getBet()** is the only method that does input.  The other methods do no input, no output and do not call any methods that do input or output.  Let's keep that idea fresh.

Here is an example of a partial run sample:

/\* -------------------- Sample Run ---------------------------\*

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

cherries (space) BAR

congratulations, you win: 25

How much would you like to bet (1 - 100) or 0 to quit? 55

whirrrrrr .... and your pull is ...

BAR BAR 7

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 555

How much would you like to bet (1 - 100) or 0 to quit? 555

How much would you like to bet (1 - 100) or 0 to quit? -2

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

(space) (space) BAR

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

cherries (space) BAR

congratulations, you win: 25

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

cherries 7 BAR

congratulations, you win: 25

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

cherries cherries cherries

congratulations, you win: 150

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

(space) BAR cherries

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

cherries BAR (space)

congratulations, you win: 25

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

BAR BAR cherries

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

BAR cherries cherries

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

(space) (space) 7

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

(space) (space) BAR

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

BAR BAR cherries

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

cherries cherries BAR

congratulations, you win: 75

How much would you like to bet (1 - 100) or 0 to quit? 5

whirrrrrr .... and your pull is ...

BAR cherries cherries

sorry, you lose.

How much would you like to bet (1 - 100) or 0 to quit? 0

Thanks for playing at the Casino!  
Your individual winnings were:  
25 0 0 25 25 150 0 25 0 0 0 0 0 75 0  
Your total winnings were: $325

-------------------------------------------------------- \*/