CIS*3260 W20 – Assignment 1
Questions and Answers

Do you want us to make a GUI or will the interactions be all he through the terminal?

All through the terminal / irb session.

Addendum 1 <from the Thursday/Friday Zoom session Q&A>

The A1 specification is *not* designed as an application, but rather as a library module or package (i.e. a set of interacting classes) that can be used by a programmer to build a dice/coin based game, or to be used interactively by a user.

In other words there is no specified mainline (as there commonly aren't when creating an OO program). Consequently the test script that you are providing is exercising and driving the interactive classes that you will have built. The classes can also be used directly, after loading, through an "irb" session in whatever way, and which order a user desires (creating two players to play against each other in a dice game; only using a cup and bag without any players; etc.)

Addendum 2 <expansion on the Wednesday Zoom lab tutorial>

When using a Ruby script (or irb session) and you need to load a number of classes (not packaged together as a gem), use the "requires" method

```
require filename.rb
```

or

```
require path/filename.rb
```

You would have as many requires as classes you need to load. These are generally put at the top of your scripts, including scripts you might copy and paste into an "irb" session.

Enumeration isn't built-in, so is there a preferred way you like us to implement it?
I'm thinking of create modules for them, but I haven't found any Ruby standards on it.

Enum doesn't exist as a programming language feature. It is a design concept and , just as interfaces, are "defined" in Ruby through use, not through declarations are modules etc.

In other words, an enum is an implicit contract between classes on what discrete "constant values" are to be used by a class.

You just need to use the symbols, they do not have to be separately defined in a class or module.

The reason the design in A1 has them explicitly called an "enum" is because I was reflecting their use in UML when creating a design (not Ruby).

The Bag class's empty function. I was under the impression that the bag doesn't need to be emptied …

You can select items from a Bag object into a Hand object, or you can empty the entire bag into the hand to be loaded into the cup. It depends on how the user wants to use get items from the bag.

… as its always filled with dice and coins …

Until you (i.e. the player) stores some dice or coins from the Randomizer class, the bag starts out empty. If the selection you use selects all of the coins and dice placing them into a Hand object, then the bag is empty again (or if you call the empty method, which empties the bag into a Hand object, of course)

… If it's not in the cup, the throw, or the hand, it's in the bag.

That is the way it should be, but you could empty the hand before you restore it to the bag. If you do that, you lose the pointer to the coins/dice and in effect no longer have them (they will be garbage collected). It is the equivalent of throwing them away.

I see that the RandomizerContainer has an 'empty' method, and so does the bag, clutch, and cup. What was the intention with this?

To show that you can empty any randomizer container, so empty is a method in the abstract class (technically, it need not be written in Ruby if there is no general default way of performing the method ... in Java, it would be labeled an abstract method). This is an indication that any randomizer container will respond to an empty method call, so you could write general code on it, i.e. polymorphism. However, in this case, at least some of the containers require you to override the superclass method, because the behaviour/method-details are different.

Does the hand empty just return nil and not do anything else?

It is implicitly doing one other thing … it is it dropping all references to the coins and dice that the clutch is holding (you would set the instance variable that holds the coins and dice to nil). As these are the only pointers that are pointing to the coins and dice by any object in the system the coins and dice are now effectively eliminated (with the memory taken up by the coin or die eventually being recycled by the garbage collector).

The assignment will be modified to explicitly describe this, instead of just implying it.

I'm just confused about the purpose of some of the empties.

All the other empty methods "empties" the container (bag or cup) into a Hand object, leaving the container completely empty of coins or dice.

Does a RondomizerContainer get emptied out as its randomizers are moved to the new container?

Yes.

I noticed that a lot of the functions had parameters that are less than 3 characters in length. I was made aware that characters less than 3 characters in length are not meeting the coding standards for Ruby. Is it OK that I renamed the parameters that were shorter than 3 characters to longer names?

Parameter names are the same as any other programming language, irrelevant to the functioning of the code. I do not care what you use. The one and two-character representations were only for brevity sake. In part this is because I gave the type information for the parameter (after the : as used in UML), while in Ruby this is not given in the code.

In other words, you can use whatever parameter names you want as they are not part of the signature of the method.

Does this mean we can add additional methods and class variables (single @) to these classes as we see fit (so long as it works within the guideline)?

Absolutely (so long as it works within the guideline)

I am also assuming we cannot add any additional parameters to the methods you listed.

Again correct.

For enums, I have made the assumption that these can be passed as strings either then the number values. So :yellow would be ":yellow" or :H would be ":H" when passed to something. Is this fine? I have mentioned it in my readme as an assumption. I just have finished most of the assignment and don't want to redo it all if I don't need to. If this needs to be changed, how would I pass these values to different methods?

No, the enums are not strings. They are symbols.

:yellow is not the same as "yellow" nor is it the same as ":yellow"

:yellow is a symbol, not a string. You can use a symbol just by typing it as :my_symbol which creates/accesses the symbol as it is. You can pass symbols in as arguments, return symbols, compare symbols to see if they are the same etc.
The difference between a symbol and a string is that a symbol is replaced behind the scenes with a global pointer to a constant location in memory that has the name "yellow" but when you compare symbols you are comparing the pointers, not the contents character by character. Also symbols are unique, so :yellow and :yellow will point to the same memory address, while ":yellow" points to a different memory address than :yellow or a different instance of ":yellow".

You generally use symbols as keys to Hash tables. You can use strings for the keys as well, but you get better speed performance from the Hash table if you use symbols and it is the standard approach.

Is the randomize() method under the Randomizer class suppose to randomize the randomizer item to go in the bag? As in choose whether a coin or a die goes into the bag? AND is it suppose to choose the values of that item? E.g if die was chosen, it would randomly pick a colour and number of sides for the die too?

No. The randmize() method has no parameters. It does not have access to the Randomizer class methods, so it cannot do what you suggest. Furthermore, randomize() just randomizes the Randomizer instance (either a coin or a die). It has nothing to do with the bag. Items in the bag cannot be randomized (there is no such method to call on the bag to do that). You can randomize a coin or die directly, or through the throw method of Cup.

If you want to randomly select coins and dies for a game that you design to test the classes, you may of course do so as long as that is done outside of the classes I designed.

Are we required to add comments to each method?

This is proper programming style. You should always do that.