# CSIS 10B Assignment 1

Read: Appendices A, B, C, D and Chapter 1

Download and unzip the assignment1 source folder. Choose and complete any 10 points from the problems below. Use BlueJ or Eclipse to complete the assignment, then export JAR file and upload to the server using your pass code. You may do more than 10 points of work but the max award will be 11 points.

## 1) (5 points) Coding Bat Review Problems

Log in to coding bat and solve the following problems (1 point each):

| String-1: | Array-1: |
|---|---|
| withoutEnd, | makeEnds, |
| withouEnd2, | sameFirstLast, |
| middleTwo | double23 |

You can read the Javadoc on the String class to find out more about the substring and other methods.

Copy each of your solutions from Coding Bat into the CodingBat class in the Assignment1 project folder.

## 2) (5 points) Classes and Inheritance

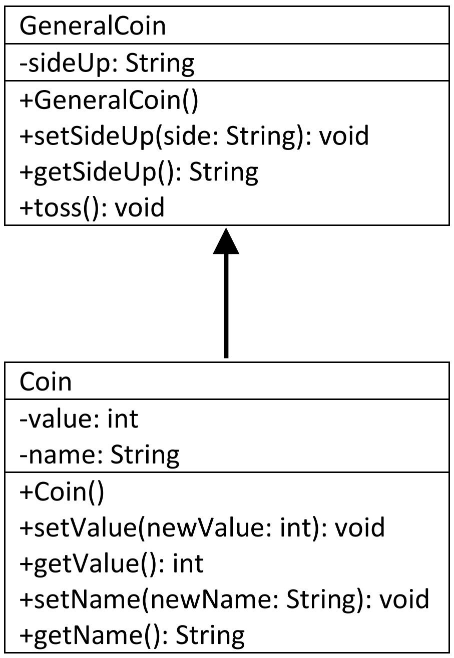
a) Define a class GeneralCoin that represents a coin with no value or name. The coin should have a heads side and a tails side and should be able to tell you which side is up. You should be able to "toss" the coin so that it lands randomly either heads up or tails up. [See Javadoc on Random class.]

Here is a suggested UML for your GeneralCoin class.

| GeneralCoin |
|---|
| -sideUp: String |
| +GeneralCoin()<br>+setSideUp(side: String): void<br>+getSideUp(): String<br>+toss(): void |

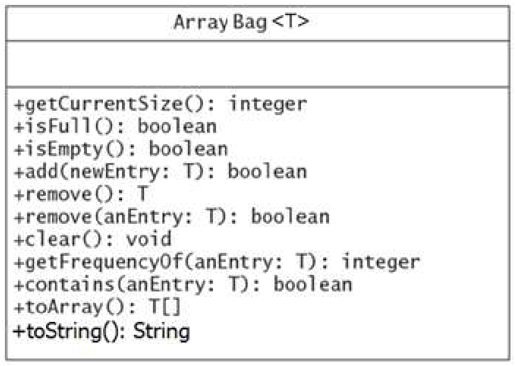
b) Open file CoinApp and activate section 2B (add a / before the /*** ) Write a program that tosses two GeneralCoin coins 50 times each. Record and report how many times each coin lands heads up. Also report which coin landed heads up most often.

c) Now that your GeneralCoin class works, use inheritance to derive a class Coin from GeneralCoin that adds a monetary value and a name as data fields. Provide your class with appropriate methods. A suggested UML inheritance diagram showing the methods of your Coin class is on the next page.



d) In CoinApp, deactivate the prob 2B code and activate prob 2D. Write a small program in the CoinApp file under section 2D that tests your Coin class and shows that your new coins inherit the behaviors of a GeneralCoin object. In particular, your new coins can be tossed.

e) In CoinApp, deactivate the prob 2D code and activate prob2E. To fully appreciate the difference between using Arrays and using Bags to store multiple items, you are now asked to write a program that creates a number of coins, storing some in an ArrayBag and some in an Array. An ArrayBag is a Bag that uses an array inside to keep track of the items. You should have the ArrayBag documentation open to be able to use it properly, or refer to the following UML diagram:


Array Bag <T>
+getCurrentSize(): integer
+isFull(): boolean
+isEmpty(): boolean
+add(newEntry: T): boolean
+remove(): T
+remove(anEntry: T): boolean
+clear(): void
+getFrequencyOf(anEntry: T): integer
+contains(anEntry: T): boolean
+toArray(): T[]
+toString(): String

Create an ArrayBag of Coin objects and an Array of Coin objects:

```
ArrayBag<Coin> headCoins = new ArrayBag<Coin>(40);
Coin []  tailCoins = new Coin[40];
```

Now use a loop to create ten of each kind of coin (for example, ten pennies, ten nickels, ten dimes, and ten quarters), or a total of 410 cents or $4.10. Inside the loop, toss each coin after you create it, and then, if it's heads, put the coin in the headCoins bag, and if it's tails, assign the coin to the next available cell in the tailCoins array. **(NOTE: If you really are stuck on working with arrays, you can use an ArrayBag<Coin> for the tail coins as well.)**

Now, use a second loop to remove all coins from the headCoins bag, and, tally up the total value of all of them.

Use a third loop to tally the value of all of the coins in the tailCoins array.

Print the total value in each container (headCoins and tailCoins) and the sum of all coins (you should get a value of 410, if not, check for errors!)

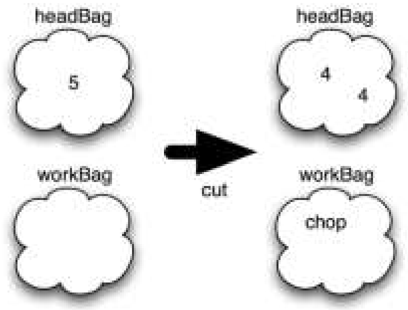
## 3) (1 points) CHALLENGE – Hydra Simulation

In this problem you will complete a simulation of a fight with the mythical Greek hydra. As legend goes, if you were to chop off the head of a hydra, two smaller heads would grow back in its place. In order for our fight to have an end, we will assume that once the size of a head is small enough, no new heads will grow back in its place. The goal of this application is to determine the amount of work required to kill a hydra with a single head, when the size is given as input.

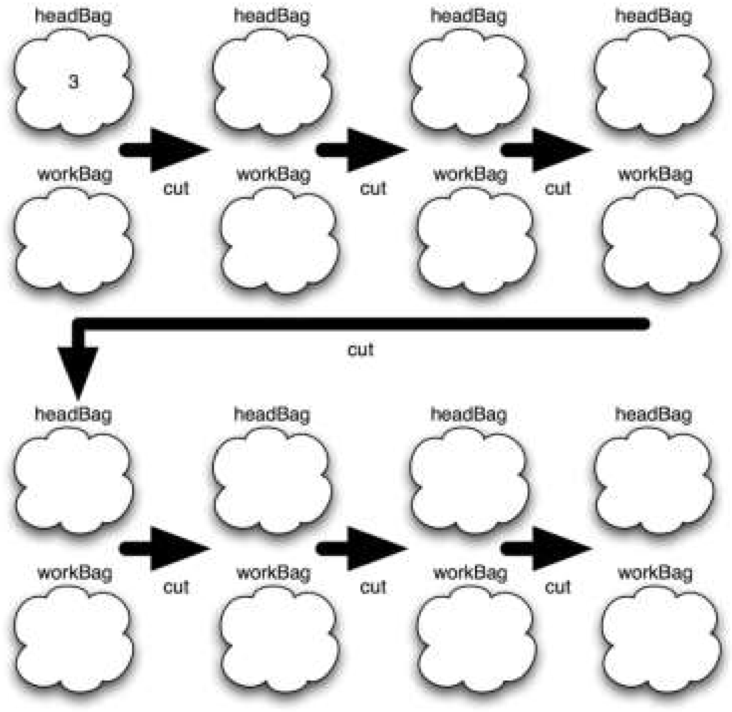
We can view our hydra as a collection of heads, each of which has a size. To indicate the size we will use an integer value. Each time we cut off a head it is replaced by two smaller heads that are one size smaller. For example, if we chop off a head of size 5, two heads of size 4 spring up in its place. The exception to this rule is that a size 1 head does not grow back. (Fortunately for us, otherwise we would never finish.) A bag is perfect to represent the state of the hydra as the fight continues. We need to know what heads the hydra currently has and what the size of each of the heads is, but they are in no particular order. In addition, there can be multiple heads of the same size.

We will use a second bag to accumulate the answer to "How many cuts did it take to kill the hydra?". Each time we cut off a head, we will put the string "chop" into the bag. Again, a bag will work well. We don't care about the order of the strings in the bag and we will certainly have duplicates. At the end of the simulation, the number of strings in the bag will give us the answer to the question.

We want to visualize the process of the simulation as a series of steps and from that determine an algorithm. For example, if we start with one head of size 5, one cut results in the following transition.



Using the above as a model, complete the seven steps in the simulation for a hydra starting with a single head of size 3.



Examine your sample simulation, and define an algorithm for what to do during a single step.

Single Step:

Given your previous algorithm, come up with an algorithm that performs the simulation. Don't forget initializations and reporting the results.

Hydra:

There is one issue that we need to be aware of with the bag ADT. The add() method may not always succeed. If there is not enough space in the bag to add the item, the add method will return false and the item will not be added into the bag. Obviously, this will have an effect on our simulation. Every time we add an item into a bag, we need to examine the returned value. If we ever get false, we can immediately stop the simulation and report that there was a problem.

Modify your single step algorithm so that it is a method which returns true if the step was successful and false otherwise.

Single Step:

Modify the program algorithm so that it will end early if there is a bag overflow.

Hydra:

## *Directed Lab Work*

Pieces of the Hydra class already exist and are in *Hydra.java*. Take a look at that code now if you have not done so already. Also before you start, make sure you are familiar with the methods available to you in the ArrayBag class. The documentation for ArrayBag/BagInterface is available here.

Compile the classes Hydra and ArrayBag. Run the main method in Hydra.

*Checkpoint: If all has gone well, the program will run and accept input. It will report that the head bag is null and then generate a null pointer exception. The goal now is to create and initialize the two bags.*

Create a new ArrayBag<Integer> and assign it to headBag.

Add the initial head into headBag.

Create a new ArrayBag<String> and assign it to workBag.

*Checkpoint: Compile and run the program. Enter 4 for the size of the initial head. The program should print out Bag [ 4 ] for the head bag. It should then report that the number of chops required is 0. The next goal is to cut off a single head. It will be encapsulated in the method simulationStep().*

Complete the **simulationStep()** method. Refer to the algorithm you developed in the pre-lab exercises to guide your code development. For now, don't worry about overflow.

Call **simulationStep(headBag, workBag)** in main just after the comment ADD CODE HERE TO DO THE SIMULATION.

Print out headBag just after the call to simulationStep.

*Checkpoint: Compile and run the program. Enter 4 for the size of the initial head. The program should print something similar to*

```
The head bag is Bag[ 4 ]
The head bag is Bag[ 3 3 ]
The number of chops required is 1
```

*We see the head bag before and after the simulation step. The next goal is to do multiple steps of the simulation.*

Wrap the lines of code from the previous two steps in a while loop that continues as long as there is an item in the head bag.

*Checkpoint: Compile and run the program. Enter 3 for the size of the initial head. The program should print something similar to*

```
The head bag is Bag[ 3 ]
The head bag is Bag[ 2 2 ]
The head bag is Bag[ 2 1 1 ]
The head bag is Bag[ 2 1 ]
The head bag is Bag[ 2 ]
The head bag is Bag[ 1 1 ]
The head bag is Bag[ 1 ]
The head bag is Bag[  ]
The number of chops required is 7
```

*Check the sequence of steps and verify that the work done was 7.*

*Run the program again using 4 for the size of the initial head. The work done should be 15.*
*Run the program again using 5 for the size of the initial head. The work done should be 25.*
*Run the program again using 6 for the size of the initial head. The work done should be 25.*

*Run the program again using 7 for the size of the initial head. The work done should be 25.*

*Notice that for any input over 4, the work done is always 25. This is caused by an overflow of the work bag.*

*For the final checkpoint we want to fix the code so that if there is an overflow, we stop the simulation and report the issue.*

Modify the simulationStep() method so that it will return false if either bag overflows. You will need to capture the return value of eac call to the add() method.

Modify the while loop in the main() method so that it only continues if noOverflow is true.

Capture the return value from the call to the simulationStep() methods and use it to set noOverflow correctly.

*Final checkpoint: Compile and run the program. Enter 3 for the size of the initial head. The program should print something similar to*

```
The head bag is Bag[ 3 ]
The head bag is Bag[ 2 2 ]
The head bag is Bag[ 2 1 1 ]
The head bag is Bag[ 2 1 ]
The head bag is Bag[ 2 ]
The head bag is Bag[ 1 1 ]
The head bag is Bag[ 1 ]
The head bag is Bag[  ]
The number of chops required is 7
```

*Run the program again using 4 for the size of the initial head. The work done should be 15.*
*Run the program again using 5 for the size of the initial head. You should get the computation ended early.*
*Run the program again using 6 for the size of the initial head. You should get the computation ended early.*
*Run the program again using 7 for the size of the initial head. You should get the computation ended early.*