CS 284: Homework Assignment 1
Due: September 19, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Your code must include a comment with your name and pledge.

Late Policy. There are a total of 10 points available for this assignment. You will lose .5 points for every two hours late you hand in your assignment (rounded down).

2 Assignment

Define a class `CoinPurse`. A `CoinPurse`. entity can store three different kinds of coins: Galleons, Sickles and Knuts. There are 17 Sickles in a Galleon, and 29 Knuts in a Sickle. A `CoinPurse` can hold at most 256 coins regardless of type.

Throughout this assignment, you will be implementing a number of `CoinPurse` operations. They are divided into groups below. 1 point of your score will be given for style, comments, and readability. There is a bonus questions for an extra .5 points. You are not required to answer the bonus question.

2.1 Errors and Exception Handling

For this homework, you have several options for handling errors of incorrect arguments.

- You can throw an Exception in the function. I am not picky about what type of Exception is thrown, but for most cases in this homework consider an IllegalArgumentException. See the ExceptionTest.java example on Canvas.
- You can use a `try-catch` block. In the `catch` block, you should make sure an error message is printed.

- You can directly print an error message using `System.out.println`. This is only acceptable for this first homework. For future homework, you will be expected to use Exceptions.

If you choose one of the last two options for a function with a non-void return type, you will have to ensure that a value of the appropriate type is returned. If the return type of a function is an `int`, return -1 in the case of error. If the return type is `int[]`, return an empty array.

2.2 Basic Operations (2.5 pts total)

- *Constructors (.5 pts)*
  - A constructor `CoinPurse()` for creating an empty coin purse.
  - A constructor `CoinPurse(int g, int s, int k)` for creating a coin purse containing g Galleons, s Sickles, and k Knuts. Make sure to consider whether the `CoinPurse` has enough space to hold the specified coins and output an error message if not. A negative value of coins should also trigger an error message.
- *Adding and withdrawing coins (1 pt)*
  - `depositGalleons(int n), depositSickles(int n)` and `depositKnuts(int n)` functions for adding n Galloens, Sickles or Knuts respectively. If the `CoinPurse` cannot carry n additional coins, none should be deposited and an error message should be printed. A negative number of coins cannot be deposited and an error message should display if n is negative.
  - `withdrawGalleons(int nt), withdrawSickles(int n)` and `withdrawKnuts(int n)` functions to withdraw n Galleons, Sickles or Knuts respectively. If there are not enough coins in the `CoinPurse` to complete this request, no coins should be removed and an error message should be printed. A negative number of coins cannot be withdrawn and an error message should display if n is negative.
- *Cumulative operations (1 pt)*
  - `numCoins()` returns the number of coins in the `CoinPurse`.
  - `totalValue()` returns the value (in Knuts) of the `CoinPurse`. There are 29 Knuts in a Sickle and 493 Knuts in a Galleon.
  - `toString()` returns a String denoting the current contents of the `CoinPurse`. It should output the number of Galleons, Sickles and Knuts.

2.3 Exact Change (3 pts total)

- `exactChange(int n)` returns true if there is some subset of the coins in `CoinPurse` whose combined value is exactly n. It returns false otherwise. As an example, consider a `CoinPurse` that contains two Galleons, five Sickles and ten Knuts. `exactChange(559)` should evaluate to true because one Galleon, two Sickles and eight Knuts have a value of exactly 559. On the other hand, `exactChange(564)` should return false as no subset of the coins has a value of exactly 564. (*1.5 pts*)

- `withdraw(int n)` *(1.5 pts)*
  - If `exactChange(int n)` is true, `withdraw(int n)` withdraws a subset of coins from the `CoinPurse` whose total value is exactly n. It returns an integer array of length 3. The first value of the array is the number of Galleons withdrawn, the second the number of Sickles, and the last the number of Knuts. Using the example above, `withdraw(559)` should return $[1, 2, 8]$.
  - If `exactChange(int n)` is false, `withdraw(int n)` withdraws a subset of coins from the `CoinPurse` with the smallest value that is larger than n. If we consider the example above, `withdraw(564)` should return $[1, 3, 0]$. If n is larger than the total value of the `CoinPurse`, an error message should be printed and no coins should be withdrawn.

2.4 A Game of Chance (3.5 pts total)

We use the following operations to define a game of luck that can be played with `CoinPurse` entities. You can import the `java.util.Random` class for this section.

- `drawRandCoin()` draws a coin at random from the `CoinPurse`. It return 0 if the coin if a Knut, 1 is the coin is a Sickle, and 2 is the coin if a Galleon. The probability of each outcome should be proportional to the number of coins of each kind in the `CoinPurse`. For example, if a `CoinPurse` contains one Galleon, two Sickles and one Knut, `drawRanodmCoin()` should return 0 with a 25% probability, 1 with a 50% probability and 2 with a 25% probability. `drawRandCoin()` should not remove the coin from the purse. Imagine that a random coin is drawn, observed and then returned. I suggest using the class `java.util.Random` for this method. You may also use `Math.random`. If there are no coins in the `CoinPurse`, an appropriate error message should display. *(1.5 pt)*
- `drawRandSequence(int n)` draws a sequence of n coins at random from the `CoinPurse`. Returns an array of length n, where the i'th entry of the array is 0 if the i'th coin drawn is a Knut, 1 if it is a Sickle, and 2 if it is a Galleon. Coins should be drawn *with* replacement. After a coin is drawn, it is returned to the `CoinPurse` before the next coin is drawn. `drawRandSequence(int n)` should not remove any coins from the purse. If there are no coins in the `CoinPurse`, an appropriate error message should display. *(1 pt)*
- The static method `compareSequences(int[] coinSeq1, int[] coinSeq)` compares two coin sequences, element by element. The two sequences must be of equal length to be compared, else an error message should be output. A comparison occurs for each pair of elements. Each comparison is won by the coin sequence with the higher number at that index (or is a tie). `compareSequences(int[] coinSeq1, int[] coinSeq2)` returns 1 if `coinSeq1` won more rounds, -1 if `coinSeq2`won more rounds or 0 in case of a tie. *(1 pt)*.

  Consider the two coin sequences: `coinSeq1 = [0, 1, 2, 2, 0]` and `coinSeq2 = [1, 1, 1, 2, 2]`. Below is an element by element comparison of these sequences. If `coinSeq1` won the round, 1 is shown below the horizontal line. If `coinSeq2` won the round, -1 is shown. If the round was a tie, 0 is shown.

0	1	2	2	0
1	1	1	2	2
-1	0	1	0	-1

In the end, `coinSeq2` won two rounds and `coinSeq1` won one round. Therefore, `compareSequences(coinSeq, coinSeq2) = -1`.

Bonus question (.5 pt)

`drawRandSequenceNoRepl(int n)` draws a sequence of n coins at random from the `CoinPurse`. Returns an array of length n, where the i'th entry of the array is 0 if the i'th coin drawn is a Knut, 1 if it is a Sickle, and 2 if it is a Galleon. Coins should be drawn *without* replacement. This means the probability of drawing each coin type may need to be updated after each draw. Imagine that a sequence of n unique coins are drawn, observed and then all n returned. If there are not n coins in the `CoinPurse`, an appropriate error message should display.

3 Submission instructions

Submit a single file named `CoinPurse.java` through Canvas. No report is required. Your grade will be determined as follows:

- You will get 0 if your code does not compile.
- The code must implement the following UML diagram precisely (see below).
- We will try to feed erroneous and inconsistent inputs to all methods. All arguments should be checked.
- 1 point of your score will be given for style, comments and readability.

CoinPurse
private int numGalleons private int numSickles private int numKnuts private static final int CAPACITY
public CoinPurse() public CoinPurse(int g, int s, int k) public void depositGalleons(int n) public void depositSickles(int n) public void depositKnuts(int n) public void withdrawGalleons(int n) public void withdrawSickles(int n) public void withdrawKnuts(int n) public int numCoins() public int totalValue() public String toString() public boolean exactChange(int n) public int[] withdraw(int n) public int drawRandCoin() public int[] drawRandSequence(int n) public static int compareSequences(int[] coinSeq1, int[] coinSeq2) BONUS public int[] drawRandSequenceNoRepl(int n)