PPA Assignment 7

This coursework is designed to test the content from Weeks 1 - 5 and 7 and 8.

Do you think there is a problem with any of the content below? Let us know immediately at programming@kcl.ac.uk.

Read through this brief carefully before starting your attempted solution and make sure you follow it as closely as possible, paying particular attention to the names required for elements of your code. Also ensure that you comment your code, as part of the assignment documentation.

The total marks available for this assignment are .

For this week's assessment, consider the following scenario, and then complete the tasks that follow it:

On a series of islands in the Carribean exist a set of hidden treasure chests, buried there by different pirates over many years. Now, a new pirate wishes to sail to each of these islands and find any treasure that is buried there.

1. Model this scenario based on the following requirements:

    1. First model a `GoldCoin` with the following features. Each gold coin has a `coinNumber`. This number is stamped onto each coin by the Eastern Trading Company (who mint all gold coins, and wish to keep track of them), when it is first made. Each coin number is unique, and is determined using a process that is kept secret by the Eastern Trading Company. To model the secrecy of this process, it should not be possible to pass a `coinNumber` (or indeed any other information) to the `GoldCoin` class. Instead setting the coin number, and keeping it unique, should be handled internally by the `GoldCoin` class. However, it should still be be possible to read the number stamped onto a coin.

    2. Next, model a `TreasureChest`. A `TreasureChest` consists of a set of `GoldCoin`s. A `TreasureChest` can hold 12 `GoldCoin`s; a value that is fixed for all chests. When a chest is first made, it is filled with the maximum number of gold coins it can contain. It should be possible to `takeOneGoldCoin` from the chest, while there are still coins in the chest, which is then removed from the set of coins in the chest.

    3. An `Island` should be modelled as follows:

        1. Each `Island` has a `name`, and consists of a number of `locations` (for example a beach, a cave, the top of a mountain etc.), each of which may or may not contain a buried `TreasureChest`. When an `Island` is first created, a `TreasureChest` should be buried at one of these locations, chosen at random by the program.

        2. It should be possible to `dig` at one of the locations on that island, as indicated by a supplied integer. If, after digging, a location contains a treasure chest, this should be removed from the

island and returned as a result of the dig. If a location does not contain a buried treasure chest, then, naturally, no chest should be returned. Appropriate steps should be taken to ensure that the integer supplied corresponds to one of the available locations on the island, and if it does not, then, again, no chest should be returned.

3. Finally, it should be possible to access the `name` of an island, and the number of bury `locations` on that island.

4. A `Pirate` should be modelled as follows:

1. A `Pirate` has a name, and a `purse`, which can hold a number of `GoldCoin`s. A `Pirate` also has a `map`, which depicts a number of `Island`s.

2. A pirate has the ability to `search` for an island, which should involve the following process:

1. The search starts with a `name`, which may correspond to an `Island` on the pirate's `map` that contains treasure. This name is passed to the `Pirate`.

2. The `Pirate` then `search`es for an `Island` on their `map` with this `name`. If an island with this name is not found on the pirate's map, then a negative result is returned, otherwise the result is the correct `Island`.

3. A pirate has the ability to `getTreasure` buried on an `Island`, which should involve the following process:

1. A pirate attempts to `getTreasure` on an `Island` by `dig`ging in each location to see if it contains a `TreasureChest`. If no chest is found, a negative result is returned.

2. If a tresure chest is found in one of the locations, the pirate `takes``OneGoldCoin` from that `TreasureChest`, until all of the `GoldCoin`s are taken, and places these coins into their own `purse`. Once this is done, the treasure is obtained, and the result is positive.

4. A pirate can be asked for the `totalCoins` in their `purse`.

5. A `Pirate` can `speak`, which adds a randomly selected pirate-themed suffix, chosen by the program, to a supplied `phrase`, which is then printed. The list of pirate suffixes are as follows:

- `phrase` followed by ", arrr!"
- `phrase` followed by ", shiver me timbers!"
- `phrase` followed by ", avast!"
- `phrase` followed by ", ahoy, matey!"
- `phrase` followed by ", yo, ho ho!"

2. Create a class `TreasureHunt`, which can be compiled and run from the command line. Use this class to do the following (in order), using the classes and methods you have created for Question 1. For full marks, all print statements should be in `Pirate speak`.

1. Create a map with three islands. There should be 18 possible bury locations on each island. Add these islands to the map in the following order:

1. Port Royal
2. Tortuga
3. Dominica

2. Create a pirate with an appropriate name. For example, *Captain Chapman* or *Chapbeard*. This pirate should also have the map created above.

3. Allow a user to suggest the names of an island to the pirate, who will then `search` for this island on their map.

4. If the island cannot be found on the pirate's map, print that the suggested island could not be found.

5. If the island is found, prompt the pirate to `getTreasure` from this island, if any is buried.

6. If a chest is found, print which island the chest was found upon, and the new total coins in the pirate's purse.

7. If a chest is not found, print that the suggested island did not contain any treasure.

8. A user should be able to continuously suggest the names of island to the pirate, until they enter the word *stop*.

Once completed, submit your assignment using the link marked `Assignment 7: Nexus Submission Link' on KEATS.

You must complete the plagiarism and collusion training before submitting this assignment.

You must also submit complete documentation of your solution. You will find a sample piece of documentation in the Support section on KEATS marked `Sample Assignment Documentation'. Submit your documentation using the link marked `Assignment 7: Documentation Submission' on KEATS.

Students who do not submit documentation along with their code, or vice-versa, will receive a mark of zero.

Any submitted code or documentation that is found to be unduly similar to the code or documentation submitted by any other student(s), will result in a penalty for those involved.

Provisional mark for your code will be released on KEATS within one week of submission. Final assignment grades will be submitted to the exam board at the end of the semester, and will take into consideration the quality of your documentation and the quality of the comments written into your code directly.

For all other queries, see the Support section on KEATS, specifically the document marked `Introduction'.