Java Advanced Course

Challenge 30. Sealed classes

A pirate game

In this challenge, you'll be adding functionality to the Pirate Game from the previous challenge.

You'll be adding hidden treasure, town features, and opponents.

A town feature will affect the health of your pirate.

For example, if your pirate runs into an alligator, you may want to subtract health points.

If he finds a freshwater spring, you'll probably increase his health.

As your pirate finds loot, each item should increase his score.

Each town will have different loot, and different features.

A pirate game

In this challenge, you will have to abstract some of the functionality of your Pirate player to a Combatant class.

This means you should make a copy of Pirate, calling it Combatant, and place it in the pirate package.

This class should be abstract, and contain most of Pirate's fields (like name, weapon, and game data), as well as methods related to these fields.

The Pirate class should extend Combatant.

You'll also create a couple of other Combatant classes, such as Islander and Soldier.

You should **seal** the Combatant class.

A pirate game

You need to create two enums, both with constructors.

The first, **Loot**, defines pirate treasure such as gold coins or pearl necklaces, each with its own worth, that when found, will increase your **pirate's score**.

The second enum, **Feature**, should describe some town features, that will either add to the **pirate's health**, or subtract from it.

Features should have a health value (positive or negative), so that when a pirate runs into this feature, his health is adjusted accordingly.

Some examples of features might be an Alligator with a large negative health value, or a Pineapple with a small positive adjustment.

A pirate game

Use a record to create a Town, with the fields: name, island, level, loot, features, and opponents. For the last three use any Collection you wish.

Create a compact constructor, to initialize randomized lists of loot and features. These lists should contain only a portion of the ones available on each enum.

You should also add an opponent or two, in this constructor.

Next, you'll add a custom constructor that takes a name, and additional items to be tracked in the game data map.

At least one of the opponents should use their weapon, when your pirate uses his.

A pirate game

Randomize part of the use weapon method, so that opponents only occasionally hit each other, and deduct points from each others health accordingly.

Change PirateGame's loadData method, to load up a List of Town, instead of Strings.

Change your game's menu items to include a Find Loot option, and an Experience Town Feature option.

Change your code so that a pirate only visits a new town, after they've found all the loot in the current town.