Homework 2: Cat Cafe

Photo by Van Thanh on Unsplash

Pawffee is a up and coming Cat Cafe which is about to open in the upstate New York area. They serve food items, organic single source coffee imported from farm direct co-ops around the world, and have partnered with two different local animal shelters. Patrons can order food and drinks and interact with the cats which roam freely inside the cafe. They can also arrange to be adopted by a cat, which involves some adoption fees and paperwork which is routed through the appropriate shelter, or if they are unable to adopt they can still sign up to sponsor a feline. Sponsorships involve recurring donations which are attached to a specific cat and corresponding shelter and continue until the cat is adopted by someone. To sponsor a cat, a patron only need "favorite" the cat in the list of cats and their credit card is automatically charged.

The Pawffee cat cafe has approached you, an ambitious Kotlin consultant, to build a financial reporting system to help manage the cafe finances.

Watch the **Programming in Kotlin: Functions & Custom Types** course and use what you learn there and from the previous course to complete this challenge.

The system should define the following entities:

- People: Patrons, Employees
  - first name
  - last name
  - ID
  - phone number
  - email address

Properties unique to employees:

- hire date
- social security number
- salary

Methods unique to employees:

- clock in
- clock out

- Menu Items - drinks, food items
  - price
  - product ID

Photo by Hannah Wei on Unsplash

- Receipts
  - menu items (product ID, quantity)
  - customer ID
  - receipt total (this should be a computed property)

- Cats
  - name
  - breed
  - sex
  - shelter ID
  - cat ID
  - sponsorships

Methods for cats:

- adopt cat (takes a customer ID as a parameter)
- sponsor cat (takes a customer ID as a parameter)

- Shelter
  - shelter ID
  - shelter address
  - shelter phone number

- Sponsorship
  - patron ID
  - cat ID

Use classes, inheritance, and polymorphism to define the above entities and their relationships. For example, you might have a Person class. Both Patrons and Employees could then subclass the Person class. Properties and methods that are common for all people, should be defined in the Person class, while those that are unique to employees or patrons would be defined in the respective subclass. Feel free to modify the above structure as you see fit as you develop the application.

Note that polymorphism is key to these relationships. For example, one of the properties of a cat is a *collection* of sponsorships. A sponsorship itself is its own type of object. So, a cat has both simple properties like an ID (Integer) and a collection of Sponsorship.

The Pawffee cafe used to use some antiquated cash registers that were donated by the owner's grandmother who used to work at NASA. The cash registers worked with a modified **System/360 Series**IBM Mainframe that wrote out each receipt to a JSON text file. The system is no longer going to be used since you are making them a new one, but the owner provided you with a sample of the JSON data as an example of the data. In your application you can create some dummy data in order to be able to test the functionality described below. You can store this data in memory however you choose, for example using some hard coded arrays, or creating a utility class for populating the data.

You system must then allow the cafe to produce a report at the end of each business day with the following information:

- determine the total number of transactions (receipts) for that business day
- determine the total number of customers (both employees who bought stuff and regular patrons) for the day
- determine the total number of non-employee patrons for the day (hint: you can make use of atype check)
- for each receipt:
  - if the purchaser is an employee, a 15% employee discount is applied, but only for food and drink items
  - if a cat is on a receipt (adoption), no discount is applied

- determine the total number of adoptions, per shelter
- produce a list of the unadopted, unsponsored cats staying at the cafe currently
- produce a list of the sponsored, unadopted cats staying at the cafe
- produce a list of the top ten selling menu items for that day (exclude cats from this) , along with the gross sales of each item
- produce a list of the most popular cats, in order of rank (popularity is judged based on number of sponsorships).

Whenever a cat is adopted, it should be removed from the cat inventory.

Some suggestions:

- there are only two partnered cat shelters now, but that could change in the future
- every cat needs to belong to a single shelter, until / unless they are adopted
- You might make a property on the shelter class that is a collection of cats. When a cat is adopted, remove it from the collection. You will need to somehow *iterate* through the collection, find the cat by its unique ID, and then remove it from the collection.
- make use of `map` and `filter`

Some notes about the sample data:

- square brackets `[]` in JSON indicates an array
- more about JSON JSON Introduction

Documentation

Pawffee wants you to deliver a future proof, scalable product. Therefore they want the code you deliver to be self documenting, so that future developers have an easier time working with the code base. Follow clean code best practices, and annotate and comment your code. See Documenting Kotlin Code - Kotlin Programming Language for more information on commenting your code and using Markdown in your comments.

For the purposes of this exercise, all the data can be retained in memory - you don't need to worry about writing to a database, to disk, or fetching or posting data to an API. Don't worry, those things will be coming later. ;) Let's just hope nobody trips on the power cord to the computer hosting your application. 😻

Photo by Bundo Kim on Unsplash