Project

Evaluation Objectives

1. Programming skills using Swift
2. Object oriented programming in Swift
3. Problem solving abilities
4. Logical and Analytical skills
5. Modular code organization

Technical Requirements

1. The app must be created in XCode using MacOS -> Command Line app
2. The app name must be **Group#_Store** such as **Group1_Store**.

Besides implementing the required functionality, submissions are required to use the correct coding conventions used in class, professional organization of the code, alignment, clarity of names is all going to be part of the evaluation.

Academic Integrity

Learners are permitted to use the internet and course resources to search for syntax related to Swift.

Learners are NOT allowed to:

- Search/use partial or full solutions from the Internet
- Communicate with others, either inside or outside the class
- Share resources, including but not limited to links, computers, accounts, etc.

Submission Checklist

- A zip file containing your entire project. Here is an example of what the entire project means.

  Before zipping your file, ensure you run Build > Clean

- Name your zip file: **Group#_Store.zip**. For example: **Group1_Store.zip**
  ***.rar and *.7zip files are NOT accepted and will automatically be graded 0.**

Problem Description

You have been hired to build a console-based version of an online store. The online store sells two types of items, video games and movies. All store items have an id, title, and price.

The store has a search feature that enables its inventory to be searched by keyword

Items can be purchased from the store by customers using a gift card associated with their account. The gift card contains a starting balance of $10 and can be reloaded anytime.

Items can only be purchased if the customer has enough money in their gift card balance. When the customer makes a purchase:

1. the cost of the item is deducted from their balance and
2. the item is added to the list of items the customer owns.

The customer ***cannot*** purchase the same item twice.

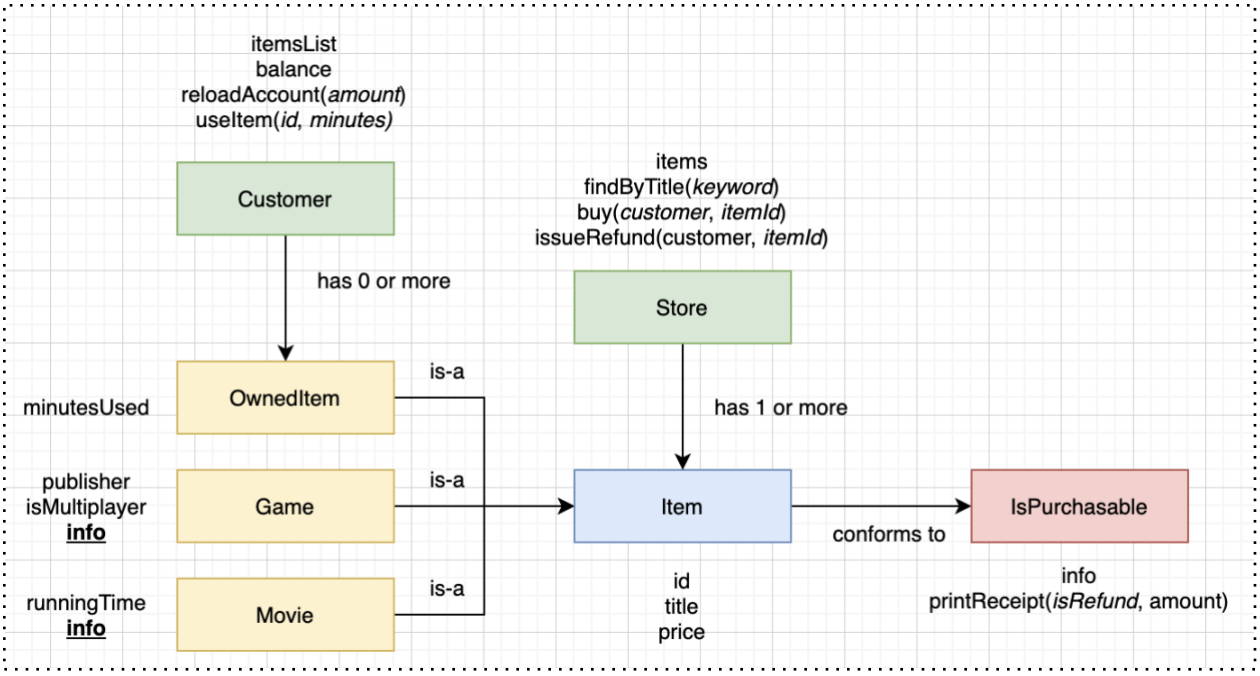
The store may also issue refunds to customers who are dissatisfied with their purchase. Refunds are issued if the customer has consumed less than 30 minutes of the product (example: playing less than 30 minutes of the video game, or watching less than 30 minutes of the movie). Otherwise, refunds are not permitted.

If a refund is issued, the original cost of the item is returned to the customer's gift card balance. The item is also removed from the customer's list of owned items.

The store will issue a receipt to the user when an item is either purchased or refunded. Every receipt displays information about the item, and the amount deducted/refunded.

Task:

Using Swift, implement the system described above. Your program must be developed using Swift's object-oriented paradigms and adhere to the following design:



After implementing the code for the entities shown above, write code to demonstrate the basic functionality of each entity, specifically:

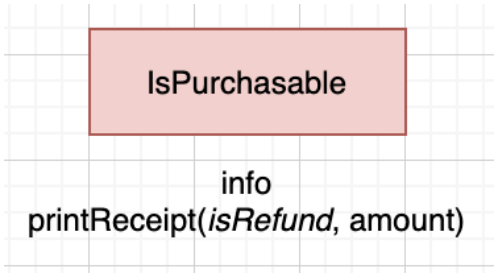
1. Creating a customer
2. Creating a variety of movies and games
3. Creating a store and adding the movies and games to the store
4. Searching for an item that exists
5. Searching for an item that does not exist
6. Trying to purchase an item that the customer cannot afford
7. Reloading the customer's gift card so they have sufficient funds
8. Trying to purchase the same item again (now, they should be able to afford it)
9. Purchasing an item that the user *does not* own
10. Purchasing an item customer *already* owns
11. Using one of the items for more than 30 minutes
12. Trying to refund an item that does NOT the refund policy requirements
13. Refunding an item that DOES meet the refund policy requirements

Class and Protocol Description

IsPurchasable protocol

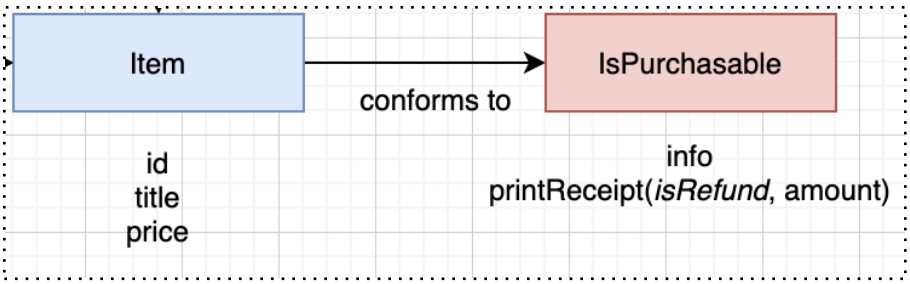
1. This protocol contains:
    - a computed property called `info`
    - a method called `printReceipt(boolean isRefund, double amount)`

2. The **Item** class conforms to this protocol



Item class

The Item class is the parent class for the OwnedItem, Game, and Movie class. It conforms to the behaviours specified in the IsPurchasable interface.



1. Contains the stored properties: id (int), title (string), price (double).

2. Conforms to the IsPurchasable protocol
    - The **`info`** computed property returns a string containing the item's title and price.
    - The **`printReceipt`** method outputs a receipt to the screen for either a purchase or refund.

        a. A sample receipt for a purchase is generated by calling the `printReceipt(..)` function with isRefund set to false. For example, `printReceipt(false, 54.99)` produces:

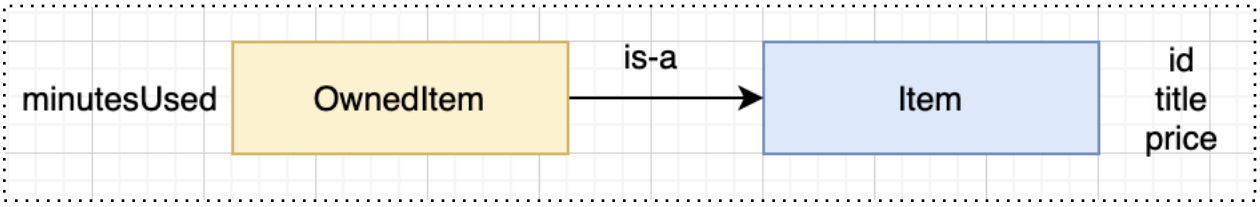
```
-------------------------
YOUR RECEIPT
-------------------------
Thank you for purchasing Heroes: Might and Magic
Purchase amount: $54.99
```

b. A sample receipt for a refund is generated by calling the printReceipt(..) function with isRefund set to true. For example, `printReceipt(true, 54.99)` produces:

```
-------------------------
YOUR RECEIPT
-------------------------
We are refunding the purchase of Heroes: Might and Magic
Refund amount: $54.99
```

Owned Item Class

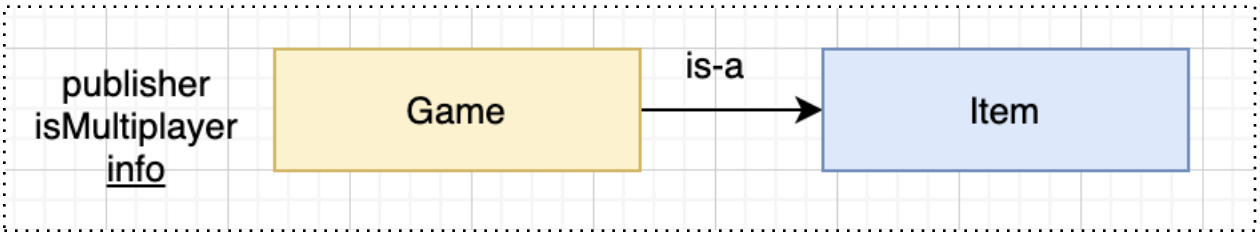
When the customer buys an item, a copy of the item is associated with the customer. This "copy" is represented by the *OwnedItem* class.



Contains the stored property **`minutesUsed`** (int). This property represents how many minutes the customer has spent watching or playing the item.

Game class

The game class represents a video game that is sold in the store.



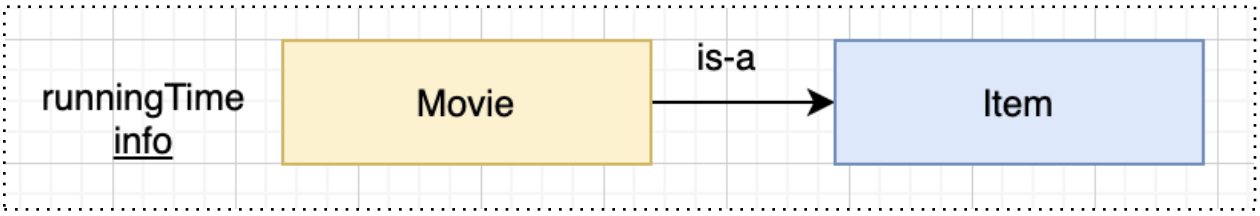
1. Contains the properties:
    a. `publisher` (String, stored property)
    b. `isMultiplayer` (boolean, stored property)
    c. `info` (String, overrides the parent implementation).

2. The `info` property overrides the parent's implementation of the property. Returns the game's title, price, and publisher, and multiplayer features of the game to the screen, like this:

```
Kingdom Rush Origins, $19.79
Publisher: Ironhide Game Studio
Is Multiplayer: false
```

Movie class

The movie class represents a movie that is sold in the store.



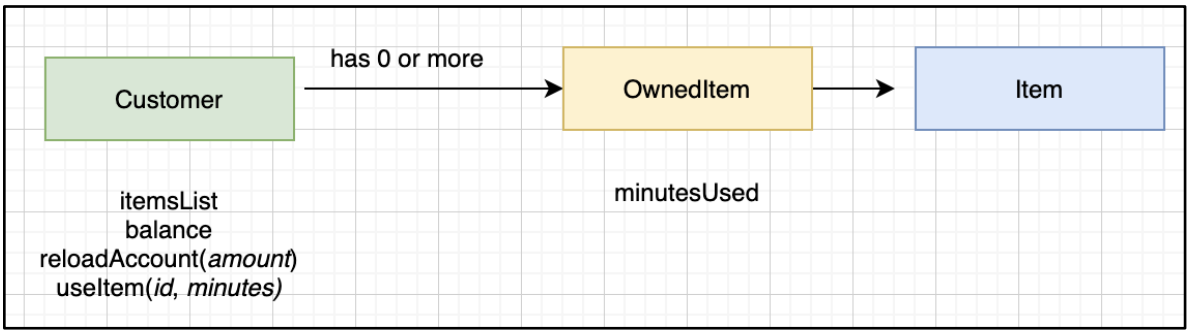
1. Contains the properties:
    a. `runningTime` (Int, stored property)
    b. `info` (String, overrides the parent implementation).

2. The `info` property overrides the parent's implementation of the property. Returns the name price and running time of the movie. Example:

```
The Spongebob Square Pants Movie, $6.69
Running Time: 98 min
```

Customer Class

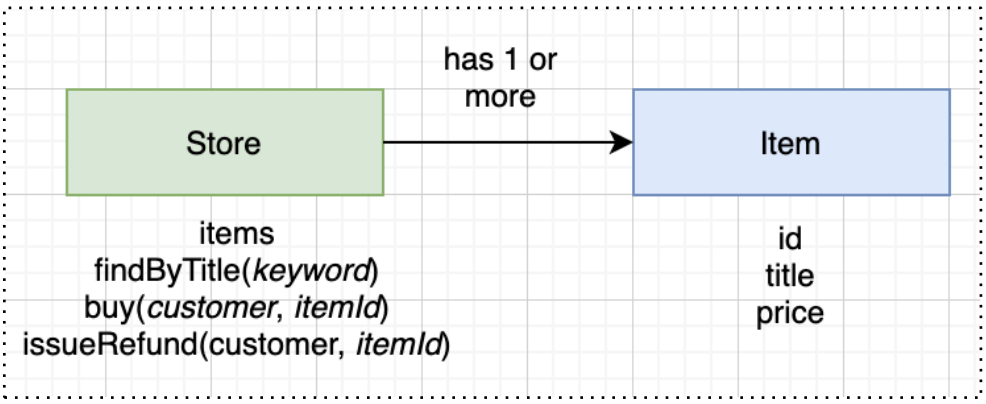
This class represents a person who is using the store.



1. Contains the properties:
   a. `itemsList` (array of OwnedItems, stored property): Contains a list of items purchased by the customer. By default, it is initialized to an empty array.
   b. `balance` (double, stored property): The current amount of money remaining on the customer's gift card. By default, initialized with a value of $10.

2. Contains the methods:
   a. `reloadAccount`(double amount): Updates the customer's balance by the specified amount
   b. `useItem`(int id, int numMinutes): Searches the customer's list of owned items for an item matching the specified id. If found, increase the number of minutes used by *numMinutes*. If no item with the specified id is found, then do nothing.

Store Class



1. Contains a stored property called `items` (`array` of `Item` objects). The `items` property represents a list of items sold in the store.

2. Contains the following methods.

`buyItem(Customer c, int itemId)`

- An item can be purchased if the customer has enough money on their gift card. If yes, then
  - The price of the game is deducted from the customer *balance*
  - The function creates a new OwnedItem object. The properties of OwnedItem should match the game that the user wants to buy.
  - This OwnedItem object is added to the Customer's array of owned items
  - Output "Purchase success!" to the console
  - Display a receipt for the purchase using the Item's **printReceipt**() function.

- An item cannot be purchased if:
  - The customer ALREADY owns a copy of the item
  - The customer does NOT have enough money to buy the item

  If the purchase fails, show an error message and the reason why the purchase failed.

`issueRefund(Customer c, int itemId)`

- An item can be refunded if the customer has used **less than 30 minutes** of the specified item. Your app should:
  - Return the price of the item to the user's *balance*
  - Display a receipt for the refund using the item's **printReceipt**() function.
  - Remove the item from the customer's list of owned items.

- If the item cannot be refunded:
  - Display an error message indicating the reasons why the refund failed.

`findByTitle(String keyword)`

- Loops through all the items in the store and outputs the details about any matching items.
- If no games are found, output an error message: "Sorry, no matching games found"
- If the matching item is a Movie, output the text [MOVIE], followed by information about the movie.
- If the matching item is a Game, output the text [GAME], followed by information about the Game.

*Expected output for a **Game**:*

```
[GAME] Kingdom Rush Origins   $19.79
Publisher: Ironhide Game Studio
Has Multiplayer: false
```

*Expected output for a **Movie**:*

```
[MOVIE] Deliverance: The Making of Kingdom Come $6.69
Length: 98 min
```

- **Your solution must demonstrate usage of object-oriented programming principles, such as reusing pre-existing properties, methods, and/or polymorphism**

--- END OF ASSESSMENT ---