Bike Shop

Gena Gizzi, Eugene Koval,

Thai Nghiem, Kevin Trinh

Polymorphic Collections

- Inventory list
  - A collection of products filled with objects of different dynamic types that all extend either the Bike or Accessory interface, which both extend the Product interface
    - Bikes: BMX, MountainBike, TandemBike, Cruiser, Unicycle
    - Accessories: Light, Helmet, AirPump

```
Predicate<Product> isBike = product -> product instanceof Bike;
Set<Product> bikeSet = accList.keySet().stream().filter(isBike).collect(Collectors.toSet());
bikeStream = inventory.filterInventory(isBike);
```

Interfaces

- Buyable
  - Represents anything that may be purchased
  - Method signatures to calculate and return retail cost
- Rentable
  - Represents anything that may be rented out for an hourly fee
  - Method signatures to calculate deposit and hourly fee
- Display
  - Frames that implement this will be able to change view states
  - Method signature to set frames view to the view passed as parameter

Abstract Classes

- Offer an easy way to represent the generic or abstract form of all the classes that are derived from it
- Product - represents the most generic form of a product sold at the shop
  - Bike
    - Abstract class that extends Product
    - Used to represent the most generic form of bikes sold
    - BMX, MountainBike, Cruiser are classes that extend Bike
  - Accessory
    - Also extends Product
    - Used to represent the most generic form of accessories sold
    - AirPump, Light, Helmet are classes that extend Accessory

Abstract Classes

- Person - represents the most generic form of a person at the shop
  - Customer
    - Concrete class that extends Person
    - Represents a customer at the shop
- View - represents the most generic views that a user interface can display
  - BikeView
    - Extends view
    - Represents more specific view of bikes
    - LogInView, ProductsView, GreetingView are classes that extend BikeView

GUI

- Challenging aspects:
  - Filtering and displaying products
  - Changing views
  - Getting different layout views to look the desired way
  - Taking everyone's individual code and combining them successfully

Customized Error Handling

- Customer record validation
  - Customer must input valid information when creating an account or logging on
  - Exceptions are thrown and errors are displayed if customer tries to input invalid information
    - Illegal email exception
      - Email must have ‘@’ character and must fit regular expression pattern
    - Illegal password exception
      - Password must be greater than 6 characters

Customized Error Handling

- Cost format exception
  - Cost - abstract data type that represents money and has add and subtract methods
  - Exception is thrown when there is an error parsing the cost
- Invalid product exception
  - Exception is thrown if an invalid product is listed in the CSV file that is being imported

File Input/Output

- File input
  - CSV file imported using BufferedReader and FileReader
  - File data contains list of products, model, make, current inventory, prices
- File output
  - Customer information records are serialized and saved to an output file
  - Serialized files are deserialized when customer logs in and views account information

Design Patterns

- Model-View-Controller (MVC)
  - Software architecture that separates application into three parts: model, view, controller
  - Defines the roles that objects play and the way they communicate
    - Model - products
    - View - panels and frames
    - Controller - views of GUI
- Visitor
  - Command pattern that defines a new operation separate from the object structure on which it operates
    - Application separates views from the frame containing them

Team Contributions

- Gena Gizzi
  - Customer records
  - Validation and error handling
  - GUI
- Eugene Koval
  - File I/O serialization
  - Validation and error handling
  - Inventory
  - GUI
- Thai Nghiem
  - Customer cart
  - Sequence Diagram
  - Class Diagram
- Kevin Trinh
  - Products Filter
  - GUI