Amusement Park Programming Project

Project Outcomes

1. Use the Java selection constructs (`if` and `if else`).
2. Use the Java iteration constructs (`while`, `do`, `for`).
3. Use Boolean variables and expressions to control iterations.
4. Use arrays or `ArrayList` for storing objects.
5. Proper design techniques.

Project Requirements

Your job is to implement a simple amusement park information system that keeps track of admission tickets and merchandise in the gift shop. The information system consists of three classes including a class to model tickets, a class to model gift shop merchandise, the amusement park, and the amusement park tester. The gift shop supports access to specific merchandise in the park's gift shop and to purchase the merchandise or to order new merchandise for the gift shop. The UML diagram for each class (except the tester class) is given below.

1) Develop a simple class that models admission tickets. Each admission is described by several instance fields:
    a. A ticket number as a long integer to identify the unique ticket,
    b. A ticket category represented as a String to store the category of the ticket (i.e. adult, child, senior),
    c. A ticket holder represented as a String to store the name of the person who purchased the ticket,
    d. A date represented as a Date to store the admission date for the ticket,
    e. A price represented as a double to store the price of the ticket,
    f. A purchase status represented as a boolean to indicate if the ticket has been purchased (or is reserved).

Ticket
-number : long -category : String -holder : String -date : Date -price : double

```
+Ticket (String, String, Date, double, boolean)
+setPrice(double)
+changePurchaseStatus(boolean)
+getNumber() : long
+getCategory() : String
+getHolder() : String
+getDate() : String
+getPrice() : double
+toString() : String
```

In addition to these fields, the class has the following constructors and methods:

a. A parameterized constructor that initializes the attributes of a ticket.
b. **`setPrice(double price)`** to change the price of a textbook.
c. **`changePurchaseStatus(boolean newStatus)`** to change the purchase status of the ticket.
d. Accessor methods for all instance fields.
e. **`toString()`** to return a neatly formatted string that contains all the information stored in the instance fields.

2) Develop a simple class that models merchandise available in the gift shop such as t-shirts, sweatshirts, and stuffed animals. The class has several instance fields:
   a. An ID as a long integer to identify the specific merchandise item,
   b. A category as a String to store the specific type of merchandise,
   c. A description as a String to store the description of the merchandise,
   d. A price represented as a double to store the price of the merchandise,
   e. An instock as a boolean to indicate if the merchandise is instock or on-order.

   Valid values for category include "T-Shirt", "Sweatshirt", and "Stuffed Animal", as well as any additional category you choose to support. If invalid values are entered, an error message must be printed and the category instance field must be set to "UNKNOWN".
   In addition to these attributes, the class has the following constructors and methods:
   f. A parameterized constructor that initializes the attributes of a merchandise item.
   g. **`setPrice(double price)`** to change the price of the merchandise.
   h. **`setInstock(boolean newStatus)`** to change the status of the merchandise item.
   i. Accessor methods for all instance fields.
   j. **`toString()`** to return a neatly formatted string that contains all the information stored in the instance fields.

Merchandise
-id : long -category : String -description : String -price : double -inStock : boolean
+Merchandise(String, String, String, double, boolean) +setPrice(double) +setInstock(boolean) +getId() : String +getCategory() : String +getDescription() : String +getPrice() : double +getInstock() : boolean +toString() : String

3) Develop class **`AmusementPark`** that keeps track of tickets and gift shop inventory. The `AmusementPark` uses two `ArrayList`s to store `Ticket` and `Merchandise` objects. The `AmusementPark` provides several methods to add merchandise to the gift shop and to access merchandise. The following UML diagram describes the class, the constructor, and the methods:

AmusementPark
-tickets : ArrayList<Ticket> -merchandise : ArrayList<Merchandise> -name : String
+AmusementPark(String) +getName() : String +getTicketDates() : ArrayList<Date> +getTickets(Date date) : int +getTicket(long id) : Ticket +getMerchandise() : ArrayList<Merchandise> +getMerchandise(String category) : ArrayList<Merchandise> +getMerchandise(long id) : Merchandise +addTicket(Ticket) +addMerchandise(Merchandise) +buyMerchandise(String id) +buyTicket(String id)

a. The class has three instance fields:
    a. name, the name of the bookstore
    b. tickets, an `ArrayList<Ticket>` storing `Ticket` objects

    c. merchandise, an `ArrayList<Merchandise>` storing `Merchandise` objects
  b. **`getName()`** returns the name of the bookstore.
  c. **`getTicketDates()`** returns an `ArrayList<Date>` of all the dates for which tickets are still available. If there are no tickets available, an empty list is returned.
  d. **`getTickets (Date date)`** returns an integer indicating the number of tickets available for the specified date.
  e. **`getTicket(long id)`** returns the `Ticket` that matches the specified id. If there is no `Ticket` matching the given id, `null` is returned.
  f. **`getMerchandise()`** returns an `ArrayList<Merchandise>` of all the inventory (in-stock and ordered). This method must create a separate copy of the `ArrayList` before it returns the list. If there are no merchandise items in the `AmusementPark`, an empty list is returned.
  g. **`getMerchandise(String category)`** returns a list of `Merchandise` objects whose category matches the specified category. For example, if called with "T-shirt" the method returns all `Merchandise` objects with the category "T-shirt" as a new list. This method must create a new copy of an `ArrayList` that stores all the matched `Merchandise` objects. If no items in the `AmusementPark` match the given name, an empty list is returned.
  h. **`getMerchandise(long id)`** returns the merchandise item that matches the specified id. If there is no merchandise item matching the given id, `null` is returned.
  i. **`addTicket(Ticket)`** adds a new `Ticket` to the inventory of the `AmusementPark`.
  j. **`addMerchandise(Merchandise)`** adds a new `Merchandise` to the inventory of the `AmusementPark`.
  k. **`buyMerchandise(String id)`** removes a `Merchandise` object from the list of merchandise of the `AmusementPark`. If the id does not match any `Merchandise` object in the list, an exception is thrown.
  l. **`buyTicket(String id)`** removes a `Ticket` object from the list of ticket items of the `AmusementPark`. If the id does not match any `Ticket` object in the list, an exception is thrown.

4) Design a tester class called **`AmusementParkTester`**. The tester class has a `main()` method and tests the functionality of the class `AmusementPark` as follows:
  a. Create `AmusementPark` and name it "Walden Amusement Park".
  b. Create a minimum of three `Ticket` objects and add them to the bookstore.

c. Create `Apparel` objects, at least two of each category, and add them to the `AmusementPark`.
d. Set up a loop to:
    i. Display a short menu that allows a user to perform different actions in the gift shop such as looking up tickets or merchandise or purchasing items. Use all of the accessor methods in the `AmusementPark` to access specific items. Use the given methods to make purchases.
    ii. Prompt the user for a specific action.
    iii. Depending on the specific action prompt the user for additional input such as the id of a ticket or merchandise category, etc. You might want to use static methods in **`main()`** to handle each menu item separately.
    iv. Perform the action and display results such as the list of merchandise that the user has requested. Use the **`toString()`** method to display `AmusementPark` items on the screen.
    v. Prompt the user for continued access to the `AmusementPark` or to end the program.

Your program should handle input errors gracefully. For example, if a particular ticket is searched and not found, the program should display a message such as "Selected ticket not found." Feel free to experiment with the tester program in order to develop a more useful program.

Implementation Notes:

1) All accessor methods in `AmusementPark` must create a new `ArrayList` to copy objects into the new list. This requires loops to access objects from the corresponding instance fields and adding them to the new `ArrayList`.
2) Proper error handling is essential for this project.
3) Javadoc must be used to document `AmusementPark`, `Ticket`, and `Merchandise`.

Submission Requirements:

1. Your project submission should have four files for this assignment:
    a. Ticket.java - The `Ticket` class,
    b. Merchandise.java - The `Merchandise` class,
    c. AmusementPark.java - The `AmusementPark` class,
    d. AmusementParkTester.java - A driver program for testing your `AmusementPark` class.
2. Remember to compile and run your program one last time before you submit it