COMPUTER SCIENCE 12B (SPRING TERM, 2017)
PROGRAMMING IN JAVA

PROGRAMMING ASSIGNMENT 3 – PART 1
(40% OF PA3)

Due: Sunday, February 19th at 11:55pm

Overview:

This assignment will give you practice with inheritance.

Part 1: Class Inheritance

Consider the task of representing types of tickets to school campus events. Each ticket has a unique number and a price. There are three types of tickets: regular tickets, walk-up tickets, advance tickets and student advance tickets. Your task is to write an inheritance class hierarchy that implements the following guidelines:

Please use the specified names for the classes and methods.

1. Implement a `RegularTicket` class that represents the normal general type ticket. The operations a regular ticket object should have are:

   a. **Construct** a ticket by number.

   b. **Return** a ticket's price. Regular tickets cost $40. Name this method **getPrice()**.

   c. **Print** a ticket object. This printed string should specify the ticket number and the price. Name this method **print()**. This method should print the specific String and then return it. The output String should be in this exact format: **Ticket number = "+number+" and price = "+price**

These operations are common to all types of tickets. In all types of tickets these methods should have the same name (besides the constructor of course).

2. Implement a class `WalkupTicket` to represent a walk-up event ticket (a ticket that is purchased the day of the event). These are also **constructed** by number and they have a price of $50. Walk up tickets have the same print out as the regular ticket plus they indicate that it is a walk up ticket.

3. Implement a class `AdvanceTicket` to represent tickets purchased in advance. An advance ticket is **constructed** with a ticket number and with how many days in advance the ticket was purchased. Advance tickets purchased 20 or more days before the event cost $15, and advanced tickets purchased between 20 and 10 days before the event cost $20. If a ticket is purchased 10 or fewer days before the event costs $40. Advance tickets have the same print out as the regular ticket plus they indicate the days in advanced they were purchased.

4. Implement a class `StudentAdvanceTicket` to represent tickets **purchased in advance but by students**. A student advance ticket is **constructed** with a ticket number and with how many days in advance the ticket was purchased. Student advance tickets are sold at **half the price of normal advance tickets**: one 20 or more days early they cost $7.5, 20 to 10 days early cost

$10, and fewer than 10 days cost $20. These tickets have the same print out as the advance tickets plus they indicate that ID is required.

Write a `TestTicket` client program to test the functionality of your classes. Your program should do the following:

1. Use a ticketing system that starts from number 1 to represent ticket numbers. Each time you create a new ticket this number should increment by 1 (that should be done in your test program not in your classes).
2. Create an array that can store up to 15 tickets.
3. Array elements in positions 0-2 store regular tickets.
4. Array elements in positions 3 and 4 store walk up tickets.
5. Array element in positions 5,6 and 7 store tickets purchased 30 days, 15 days and 3 days in advance respectively.
6. Array elements in positions 8,9 and 10 store student tickets purchased 30 days, 15 days and 3 days in advance respectively.
7. Array elements in positions 11-14 store regular tickets.
8. Print all the tickets in the array.
9. Calculate the total profit from selling all tickets in the array.

Stylistic Guidelines:

You need to apply most (if not all) the concepts we've been discussing so far regarding inheritance and interfaces/abstract classes (using the `super` keyword, figuring out which fields should go in each subsclass, which methods need to be implemented/overridden, avoid redundant code, proper usage of field and method encapsulation.)

We will be graded on your appropriate definition of hierarchies, the correct usage of method overriding, constructors, and proper interaction with the superclass.

You should keep in mind the ideas we have been stressing in class so far. You do not want to have redundant code. You should properly encapsulate your classes, use constants when appropriate to avoid hard-coded numbers or hand-coded strings in your code. The choice of what constants to use is up to you, but they should represent important values that are used in your classes.

Follow general stylistic guidelines, such as indentation and whitespace, meaningful identifier names, and localization of variables.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method. Also put brief comments inside longer methods explaining the more complex sections of code.

Submission:

Your Java source code should be submitted via Latte. For late policy check the syllabus.

Below is the list of files you should submit. : `RegularTicket.java`, `WalkupTicket.java`, `AdvanceTicket.java`, `StudentAdvanceTicket.java`, `TestTicket.java`, `readmepart1.txt`

The file readmepart1.txt should include your name, the date and the homework number and describe what the program of part1 is designed to do, and what it actually does.

Grading:

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.
- **One-on-one interactive grading:** By the end of the day that the assignment is due, please make an appointment with your TA for an interactive 10-15 minute grading session. You will receive an email notifying you of the name of the TA who has been assigned to you for this assignment with further instructions on setting up the appointment. (You will be meeting with a different TA for each assignment). One-on-one interactive grading will help you improve your programming skills and avoid repeating mistakes from one assignment to the next.