## SmartCard

Passengers on a certain subway system insert a SmartCard when entering a station and when exiting at their destination station. The SmartCard, after checking some things, subtracts the cost for traveling between stations. The stations are assigned to zones and the cost depends on the travel from zone to zone. Thus, each station knows its own name and zone.

Write the Station class. Include

* Two private fields
* A default constructor
* A two-arg constructor. The arguments are the name of the station and its zone.
* Two accessor methods, getZone() and getName()
* A standard toString method, which returns the private fields, nicely labeled, like this: <station\_name>, zone <x>

Write the complete Station class below. There is no shell.

Of course, test your class. Station\_Driver has been given to you.

Now write the SmartCard class. (There is a shell.) The SmartCard stores an amount of money. When a person enters a Station and puts his/her SmartCard into the turnstyle, the SmartCard records the Station and the fact that the passenger has entered. When people exit a Station and put their card into the turnstyle, their SmartCard calculates and subtracts the cost of the trip.

A principle of object oriented programming is that the state of the object should always match the state of reality. For example, if the passenger exits a train, then that passenger cannot be on board the train. If the passenger exits a train, then the station at which the passenger boarded is set to null. It’s the programmer’s job to take care of the state of the object.

* How many fields do you declare in SmartCard?
* Make a one-arg constructor to satisfy

SmartCard buddy = new SmartCard(20.00); .

Set the other fields to the standard default values: 0 for numbers, null for the Station object, and false for the boolean.

* Include these for accessor (getter) methods:

getBalance()– returns the balance on the SmartCard as a double.

getFormattedBalance()– returns the balance on the SmartCard as a String   
 formatted as dollars and cents (Ex: $10.00). You must import java.text.DecimalFormat,   
 which has a convenient format method.

getIsBoarded()– this method returns a boolean value as to whether the passenger is   
 on board or not

getBoardedAt() – returns the Station where the passenger entered. Note: the   
 method returns the entire Station object, not simply the name of the Station.

Now instantiate buddy with $20.00 and examine the data in the fields. If your data models reality, then your output should be:

getBalance() 20.0  
getFormattedBalance() $20.00

getIsBoarded() false  
getBoardedAt() null

Now we need to make the SmartCard functional by writing *instance methods*. What methods would be helpful in a SmartCard? Implement board, cost, exit, and addMoney.

public void board(Station s)

If the passenger tries to board without having previously exited, it prints (exactly, with single spaces) "Error: already boarded?!" and returns without changing anything. If the passenger has less than $0.50 (minimum fare), it prints "Insufficient funds to board. Please add more money." and returns without changing anything. If it passes those checks, the method saves the station where the passenger boarded and updates the boarding status.

To test this method, have buddy board at Downtown and see if the SmartCard models reality.

getIsBoarded() true  
 getBoardedAt() Downtown, zone 1

Then make buddy board twice without exiting. What should be printed?

If someone’s SmartCard starts off with $0.25, what should be printed?

public double cost(Station s)

Calculates and returns the cost to exit at this station, according to zones, as follows:

* Travel within the same zone is charged 0.50
* For each additional zone that passengers pass through, they are charged an additional 0.75
* To get from any zone to another zone, passengers must pass through all zones in between. For example,
  + to travel from zone 1 to zone 1 costs 0.50
  + to travel from zone 1 to zone 2 costs 1.25
  + to travel from zone 1 to zone 4 costs 2.75
  + to travel from zone 4 to zone 1 costs 2.75

The cost method is hard to test by itself, because it needs to know at which station the passenger boarded. That information is stored in a field of the SmartCard. We will test the cost method by doing boarding and exiting several times.

public void exit(Station s)

If the passenger tries to exit without having previously boarded, it prints (exactly, with single spaces) "Error: Did not board?!" and returns. If the cost of the travel exceeds the balance on the SmartCard, it prints "Insufficient funds to exit. Please add more money." and returns. If it passes those two checks, the method updates the balance and the boarding information, prints the names of the two cities, the cost, and the balance on the SmartCard . For example:

From Center City to Downtown costs $0.50. SmartCard has $9.50

Test boarding and exiting across as many zones as you need.

If a passenger boards and exits at the same station, it costs $0.50.

If someone tries to exit, but the SmartCard does not have enough money, what is printed?

If someone tries to exit twice in a row, what is printed?

addMoney(double d)- Finally, let’s allow the passenger to add some money to the SmartCard. The method prints, e.g.,

"$5.00 added. Your new balance is $25.00"

Test it.

Assume that the system will always work and that no one will lose a SmartCard between boarding and exiting. You are given SmartCard\_Driver.java. You will turn in SmartCard.