**Off We Go! Airlines**

**Due: Tuesday, May 2, 11:55 p.m.**

**(but strongly recommend completing this as practice for quiz)**

**As with all programs, this program is to be the sole effort of the individual handing in the program. Activities such as code copying, collaboration, looking at another person’s code, getting assistance from individuals who are not a Ta or the instructor, etc. are considered academic dishonesty. You are to only get assistance from a Ta or from the instructor.**

This is a bit longer than an actual quiz question because we need to include sufficient content to be able run and test the program, but the logic is ***very*** similar to what you can expect on a quiz. Also, on a quiz, since the code does not need to be completely functional for testing, you may be asked to write selected methods for a class, rather than all of them or code segments rather than complete classes or methods.

**Program notes:** Similar to the example of the check printing class covered in lecture, this problem uses the ability of a variable of a super class type to hold addresses of subclasses in the context of parameter passing. Though the description is long, the code is not.

**Style note:** While on a quiz you frequently to do not need to adhere to good programming style, on a program you do. Remember every class needs a heading comment containing your name, the program title and a brief description of what the class represents. Every method needs its own heading briefly indicating what it accomplishes. **Also, as an important note**: calling methods is an expensive operation. A single method should not be retrieving the same value repeatedly via multiple method calls if the value is not changing between those method calls. If you find your code calling a method repeatedly to retrieve the same value, then create a local variable, call the method once and assign the value returned into the variable. Then, use the variable (instead of repeatedly making calls) in the remainder of the method.

For all methods, except the main, use good programming style and avoid the use of literal values in your code. This may well mean introducing additional instance variables not specifically noted in the description.

**The Program**

**ALL CLASS AND METHODS MUST BE NAMED EXACTLY AS SPECIFIED. METHODS MUST TAKE EXACTLY THE NUMBER AND TYPEs OF PARAMETERS SPECIFED IN THE ORDER SPECIFIED.**

Off We Go! Airlines needs to manage the passenger and checked baggage information for its flights. Use the following class structure and code design to write a Java program to solve their problem.

**Passenger Class.** Write a class definition for a class **Passenger**, that represents a passenger on a plane. Each instance of the passenger class has a String variable **name**, that holds the passenger’s name, a Sting variable **passStatus**, that holds the status of the passenger and an integer **numCheckedBags** that holds the number of bags the passenger has checked onto the plane. Two constant instance variables should be created: a double **checkedBagFee** that holds the regular price of a checked bag, which is currently $50 per bag, and an integer **maxBags** that holds the maximum number of bags a passenger can check, which is currently 4. Both variables are to be ***shared by all instances*** of the Passenger class, and their values ***should not be able to be changed*** in the code. All variables are private.

When an instance of the Passenger class is created, it will be provided with the values for **name** and **passStatus**, *in that order*.

The passenger object will contain the following additional public methods:

**addBags**- takes one integer parameter indicating the number of bags the passenger is checking in for a flight and returns a String. This method must insure that the number of checked bags does not exceed the maximum number of bags allowed. If the request is not over the limit, variables should be updated as appropriate. If it is over the limit, no change to variables should occur. The method ***returns a String*** either indicating that the transaction was successful and giving the current number of checked bags, or a String indicating that the transaction was not successful because baggage limit was exceeded by the request.

**getPassInfo** - takes no parameters and returns a String that contains, formatted with labels, the name of the passenger and the passenger’s status.

**getnumCheckedBag** - takes no parameters and returns an integer, which is the number of checked bags.

**getcheckedBagFee-** takes no parameters and returns an integer, which is the regular fee for a checked bag.

Finally, each class that extends the **Passenger** class MUST define a method **calcBagCost**. This method takes no parameters and returns a String containing the number of checked bags and the associated baggage charge.

**Subclasses of Passenger**

You are to define two subclasses of the **Passenger** class:

* **premierStatus** -when an instance of this class is created it will be given a single parameter, which is the name of the passenger. (Note to think about: the superclass constructor must be provided with the passenger status as well, which will be “premier” in this case.) As described above, the class must also define a method **calcBagCost**, that takes no parameters and returns a String. It will calculate the price of the premier passenger’s checked baggage as follows: the first 2 checked bags are free and the passenger is charged the regular checked baggage fee for every bag over 2 bags that they check. The method returns a String containing, in a clearly labelled format, the number of bags the passenger checked and the total cost of the checked bags.
* **standardStatus** –when an instance of this class is created it will be given a single parameter, which is the name of the passenger. (Again, note to think about: the superclass structure must be provided with the passenger status as well, which will be “standard” in this case.) As described above, the class must also define a method **calcBagCost**, that takes no parameters and returns a String. It will calculate the price of the standard passenger’s checked baggage as follows: every checked bag is charged the regular checked baggage fee. The method returns a String containing, in a clearly labelled format, the number of bags the passenger checked and the total cost of the checked bags.

**CheckInClass**

Write the code for a separate class **checkInClass** (this class does not extend and is not in any way a subclass of Passenger class). The **checkInClass** provides one method **printCheckedBagReceipt**. This method will accept one parameter, which will be a reference variable that will be capable accepting an object for any subclass of Passenger. It will call methods as appropriate to output the name and status of the passenger as well as their current number of bags and baggage charge.

**Class for Main**

Write the code for a class, **FlightInfo**, which will contain the main. To keep things simple, have your main test your program by inserting the appropriate code to do the following (you may use literal values to make the method calls):

* Create a premier passenger, with the name “Sue Smith”.
* Call **addBags** to check 4 bags for Sue and display the results of that attempt to check bags.
* Call **printCheckedBagReceipt** to output the passenger and baggage charge information for the passenger Sue Smith.
* Create a standard passenger, with the name “Sally Summers”.
* Call **addBags** to check 3 bags for Sally and display the results of that attempt to check bags.
* Call **printCheckedBagReceipt** to output the passenger and baggage charge information for the passenger Sally Summers.
* Create a premiere passenger, with the name “Tom Traveler”.
* Call **addBags** to check 5 bags for Tom and display the results of that attempt to check bags.
* Call **printCheckedBagReceipt** to output the passenger and baggage charge information for the passenger Tom Traveler.

**Drop all 5 classes in moodle when done.**