# Project: Air Ticket – Part 2

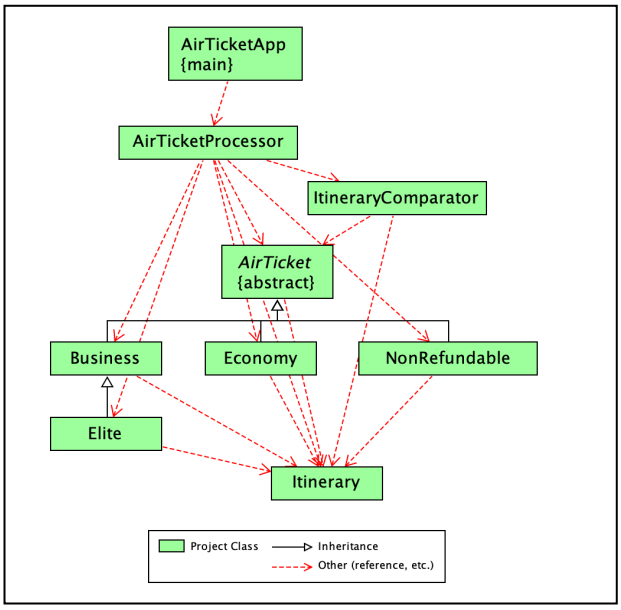


## Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. Note that there is also an optional Skeleton Code assignment which will indicate level of coverage your tests have achieved (there is no late penalty since the skeleton code assignment is ungraded for this project). The files you submit to skeleton code assignment may be incomplete in the sense that method bodies have at least a return statement if applicable or they may be essentially completed files. To avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code assignment. If you are unable to submit to Web-CAT directly or via jGRASP, you should e-mail your project Java files in a zip file to your TA before the deadline. Test files are not required for this project. If submitted, you will be able to see your code coverage, but this will not be counted as part of your grade.

Files to submit to Web-CAT (*test files are optional*):
Classes from Project 9
Itinerary.java, *ItineraryTest.java*
AirTicket.java
NonRefundable.java, *NonRefundableTest.java*
Economy.java, *EconomyTest.java*
Business.java, *BusinessTest.java*
Elite.java, *EliteTest.java*

New Classes in Project 10
ItineraryComparator, *ItineraryComparatorTest*
AirTicketProcessor, *AirTicketProcessorTest*
AirTicketApp, *AirTicketAppTest*




## Recommendations

You should create new folder for Part 2 and copy your relevant Part 1 source and optional test files to it. You should create a jGRASP project with these files in it, and then add the new source and optional test files as they are created.

## Specifications – Use arrays in this project; ArrayLists are not allowed!

**Overview**: Project 10 is the second of three projects that will involve the pricing and reporting of air tickets. In Project 9, you developed Java classes that represent an itinerary and categories of air tickets including non-refundable, economy, business, and elite, all of which include an itinerary. In Project 10, you will modify the AirTicket class and you will create three additional classes: (1) AirTicketProcessor that includes a method to read in a ticket data file and methods to generate several reports, (2) ItineraryComparator which implements the Comparator interface for AirTicket; and (3) AirTicketPart2App which includes the main method for the program. Note that the main method in AirTicketPart2App should create a AirTicketProcessor object and then call the readFile method on the AirTicketProcessor object, which will add AirTicket objects to the list as the data is read in from a file. You can use AirTicketPart2App in conjunction with interactions by running the program in a jGRASP canvas (or debugger with a breakpoint) and single stepping until the variables of interest are

created. You can then enter interactions in the usual way. In addition to the source files, you may create an *optional* JUnit test file for each class and write one or more test methods to ensure the classes and methods meet the specifications. You should create a jGRASP project upfront and then add the new source and optional test files as they are created. All of your files should be in a single folder.

- **Itinerary, NonRefundable, Economy, Business, and Elite**

  **Requirements and Design**: No changes from the specifications in Project 9

- **AirTicket.java**

  **Requirements and Design**: In addition to the specifications in Project 9, the AirTicket class should (1) implement the Comparable interface for type AirTicket and do the comparison based on the flight number field; (2) implement the getItinerary() method.
  - `compareTo`: Takes an AirTicket as a formal parameter and returns an int indicating the results of comparing tickets based on their respective flight numbers. This method is required for the AirTicket class to implement the Comparable interface for AirTicket.
  - `getItinerary`: Accepts no parameters and returns the value for the trip data field of type Itinerary.

- **AirTicketProcessor.java**

  **Requirements:** The AirTicketProcessor class provides methods for reading in the data file and generating reports.

  **Design:** The AirTicketProcessor class has fields, a constructor, and methods as outlined below.

  (1) **Fields:** An array of AirTicket objects, and an array of String elements to hold "invalid" records read from the data file. These two variables should be private so that they are not directly accessible from outside of the class. The "invalid" records array will be used in Project 11. Note that there are no fields for the number elements in each array. In this project, the size of the array should be the same as the number of elements in the array.

  (2) **Constructor:** The constructor has no parameters and initializes each of the array fields to an array of the appropriate type with length 0.

  (3) **Methods:** Usually a class provides methods to access and modify each of its instance variables (i.e., getters and setters) along with any other required methods. The methods for AirTicketProcessor are described below.

  - `readAirTicketFile` has no return value, accepts the data file name as a String, and throws FileNotFoundException. This method creates a Scanner object to read in the file one line at a time. When a line is read, a separate Scanner object on the line should be

created to read the values in that line. The data in each line is separated by a comma so the delimiter should be set to comma by invoking the `useDelimiter(",")` method on the Scanner object for the line. For each line read in, the appropriate AirTicket object is created and added to the AirTicketProcessor array field, or if not a valid category code, the line should be ignored. The data file has comma-delimited text records as follows: Each line in the file begins with a category for the ticket (N, E, B, and F are valid categories for ticket class indicating **N**onRefundable, **E**economy, **B**usiness, and **F**irst respectively; note that **F** (First) will indicate our Elite class. The second field in the record is the flight number, followed by the data for the itinerary (fromCity, toCity, departure Date/Time, Arrival Date/Time, and miles), which is followed by base price and fare adjustment factor. The last items correspond to the data needed for the particular category of ticket. The file *air_ticket_data.csv* is available for download from the course web site. Below are example data records:

```
B,DL 1865,ATL,LGA,2015/05/01 1400,2015/05/01 1640,800,450,2.0,50.0,50.00
E,DL 1867,ATL,LGA,2015/05/01 1500,2015/05/01 1740,800,450,1.0
F,DL 1863,ATL,LGA,2015/05/01 0900,2015/05/01 1140,800,450,2.5,50.0,50.00,100.00
N,DL 1861,ATL,LGA,2015/05/01 0800,2015/05/01 1040,800,450,0.45,0.90
```

- `addAirTicket` has no return value, accepts an AirTicket object, increases the capacity of the AirTicket array by one, and adds the AirTicket in the last position of the AirTicket array. *See Hints at end*.
- `addInvalidRecord` has no return value, accepts a String, increases the capacity of the invalidRecords array by one, and adds the String in the last position of the invalidRecords array. This method will be used in Project 11, but it still needs to be tested in this project. *See Hints at end*.
- `generateReport` accepts no parameters and returns a String representing the Air Ticket Report. The report is assembled by processing the AirTicket array using the original order from the file. The printed Air Ticket Report as shown below in the example output. This report does not begin with a newline character (\n).
- `generateReportByFlightNum` accepts no parameters and returns a String representing the Air Ticket Report (by Flight Number). The report is assembled by first sorting the AirTicket array using the natural ordering and then processing the AirTicket array. The printed Air Ticket Report (by Flight Number) as shown below in the example output. This report does not begin with a newline character (\n).
- `generateReportByItinerary` accepts no parameters and returns a String representing the Air Ticket Report (by Itinerary). The report is assembled by first sorting the AirTicket array by Itinerary and then processing the AirTicket array. The printed Air Ticket Report (by Itinerary) as shown below in the example output. This report does not begin with a newline character (\n).

**Code and Test:** See examples of file reading and sorting (using Arrays.sort) in the lecture notes. The natural sorting order for AirTicket objects is determined by the compareTo method from the Comparable interface. If the variable for the array of AirTickets is tickets, it can be sorted with the following statement.

```
Arrays.sort(tickets);
```

The sorting order based on itinerary is determined by the ItineraryComparator class which

implements the Comparator interface (described below). It can be sorted with the following statement.

```
Arrays.sort(tickets, new ItineraryComparator());
```

- **ItineraryComparator.java**

  **Requirements and Design:** The ItineraryComparator class implements the Comparator interface for AirTicket objects. Thus, it implements the `compare(AirTicket t1, AirTicket t2)` method that defines the ordering from lowest to highest based on the toString() value of each ticket's Itinerary. Note that the compare method is the only method in the ItineraryComparator class. An instance of this class will be used as one of the parameters when the Collections.sort method is used to sort by "itinerary". For an example of a class implementing Comparator, see examples in the class notes.

- **AirTicketApp.java**

  **Requirements and Design:** The AirTicketApp class contains the main method as described below.

  - `main` gets the file name from the command line (i.e., args[0]), creates an instance of AirTicketProcessor object, and then calls the methods in the AirTicketProcessor class to read in the data file and print the three reports as shown in the example output starting on the next page. Note that main will need a throws clause for FileNotFoundException since it will be calling the readFile method in AirTicketProcessor, which throws FileNotFoundException. An example data file can be downloaded from the assignment page in Canvas.

- **Code and Test:** If you have an optional test file for the AirTicketApp class, you should have at least two test methods for the main method. One test method should invoke AirTicketApp.main(args) where args is an empty String array, and the other test method should invoke AirTicketApp.main(args) where args[0] is the String representing the data file name. Depending on how you implemented the main method, these two methods should cover the code in main. As for the assertion in the test method, since the Economy class has a public constant for economy award miles factor, you could assert that `Economy._____` equals `1.5` in each test method. The blank should be whatever you names the constant for economy award miles factor.

  In the first test method, you can invoke main with no command line argument as follows:

```
// if you are checking for args.length == 0
// the following should exercise the code
String[] args2 = {};
AirTicketApp.main(args2);
```

In the second test method, you can invoke main as follows with the file name as the first (and only) command line argument:

```
String[] args = {"air_ticket_data.csv"};
AirTicketApp.main(args);
```

If Web-CAT complains the default constructor for `MarketingCampaignPart2` has not been covered, you may want to include the following line of code in one of your test methods to exercise the constructor.

```
// to exercise the default constructor
AirTicketApp app = new AirTicketApp ();
```

## Example Output

Output when no file is provided as a command line argument (i.e., `args.length == 0`):

```
----jGRASP exec: java AirTicketApp
File name expected as command line argument.
Program ending.

 ----jGRASP: operation complete.
```

Output when `air_ticket_data.csv` is successfully read:

```
----jGRASP exec: java AirTicketApp air_ticket_data.csv
----------------------------
Air Ticket Report
----------------------------

Flight: DL 1865
ATL-LGA (2015/05/01 1400 - 2015/05/01 1640) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.0
Total Fare: $1,000.00 (class Business)
   Includes Food/Beverage: $50.00 Entertainment: $50.00

Flight: DL 1867
ATL-LGA (2015/05/01 1500 - 2015/05/01 1740) 800 miles (1200 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 1.0
Total Fare: $450.00 (class Economy)
   Includes Award Miles Factor: 1.5

Flight: DL 1863
ATL-LGA (2015/05/01 0900 - 2015/05/01 1140) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.5
Total Fare: $1,325.00 (class Elite)
   Includes Food/Beverage: $50.00 Entertainment: $50.00
   Includes: Comm Services: $100.00

Flight: DL 1861
ATL-LGA (2015/05/01 0800 - 2015/05/01 1040) 800 miles (800 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 0.45
Total Fare: $182.25 (class NonRefundable)
   Includes DiscountFactor: 0.9
```

```
Flight: DL 1866
LGA-ATL (2015/05/01 1400 - 2015/05/01 1640) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.0
Total Fare: $1,000.00 (class Business)
   Includes Food/Beverage: $50.00 Entertainment: $50.00

Flight: DL 1868
LGA-ATL (2015/05/01 1500 - 2015/05/01 1740) 800 miles (1200 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 1.0
Total Fare: $450.00 (class Economy)
   Includes Award Miles Factor: 1.5

Flight: DL 1864
LGA-ATL (2015/05/01 0900 - 2015/05/01 1140) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.5
Total Fare: $1,325.00 (class Elite)
   Includes Food/Beverage: $50.00 Entertainment: $50.00
   Includes: Comm Services: $100.00

Flight: DL 1862
LGA-ATL (2015/05/01 0800 - 2015/05/01 1040) 800 miles (800 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 0.45
Total Fare: $182.25 (class NonRefundable)
   Includes DiscountFactor: 0.9

--------------------------------------
Air Ticket Report (by Flight Number)
--------------------------------------

Flight: DL 1861
ATL-LGA (2015/05/01 0800 - 2015/05/01 1040) 800 miles (800 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 0.45
Total Fare: $182.25 (class NonRefundable)
   Includes DiscountFactor: 0.9

Flight: DL 1862
LGA-ATL (2015/05/01 0800 - 2015/05/01 1040) 800 miles (800 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 0.45
Total Fare: $182.25 (class NonRefundable)
   Includes DiscountFactor: 0.9

Flight: DL 1863
ATL-LGA (2015/05/01 0900 - 2015/05/01 1140) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.5
Total Fare: $1,325.00 (class Elite)
   Includes Food/Beverage: $50.00 Entertainment: $50.00
   Includes: Comm Services: $100.00

Flight: DL 1864
LGA-ATL (2015/05/01 0900 - 2015/05/01 1140) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.5
Total Fare: $1,325.00 (class Elite)
   Includes Food/Beverage: $50.00 Entertainment: $50.00
   Includes: Comm Services: $100.00

Flight: DL 1865
ATL-LGA (2015/05/01 1400 - 2015/05/01 1640) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.0
Total Fare: $1,000.00 (class Business)
   Includes Food/Beverage: $50.00 Entertainment: $50.00

Flight: DL 1866
LGA-ATL (2015/05/01 1400 - 2015/05/01 1640) 800 miles (1600 award miles)
```

```
Base Fare: $450.00 Fare Adjustment Factor: 2.0
Total Fare: $1,000.00 (class Business)
   Includes Food/Beverage: $50.00 Entertainment: $50.00

Flight: DL 1867
ATL-LGA (2015/05/01 1500 - 2015/05/01 1740) 800 miles (1200 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 1.0
Total Fare: $450.00 (class Economy)
   Includes Award Miles Factor: 1.5

Flight: DL 1868
LGA-ATL (2015/05/01 1500 - 2015/05/01 1740) 800 miles (1200 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 1.0
Total Fare: $450.00 (class Economy)
   Includes Award Miles Factor: 1.5

--------------------------------------
Air Ticket Report (by Itinerary)
--------------------------------------

Flight: DL 1861
ATL-LGA (2015/05/01 0800 - 2015/05/01 1040) 800 miles (800 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 0.45
Total Fare: $182.25 (class NonRefundable)
   Includes DiscountFactor: 0.9

Flight: DL 1863
ATL-LGA (2015/05/01 0900 - 2015/05/01 1140) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.5
Total Fare: $1,325.00 (class Elite)
   Includes Food/Beverage: $50.00 Entertainment: $50.00
   Includes: Comm Services: $100.00

Flight: DL 1865
ATL-LGA (2015/05/01 1400 - 2015/05/01 1640) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.0
Total Fare: $1,000.00 (class Business)
   Includes Food/Beverage: $50.00 Entertainment: $50.00

Flight: DL 1867
ATL-LGA (2015/05/01 1500 - 2015/05/01 1740) 800 miles (1200 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 1.0
Total Fare: $450.00 (class Economy)
   Includes Award Miles Factor: 1.5

Flight: DL 1862
LGA-ATL (2015/05/01 0800 - 2015/05/01 1040) 800 miles (800 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 0.45
Total Fare: $182.25 (class NonRefundable)
   Includes DiscountFactor: 0.9

Flight: DL 1864
LGA-ATL (2015/05/01 0900 - 2015/05/01 1140) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.5
Total Fare: $1,325.00 (class Elite)
   Includes Food/Beverage: $50.00 Entertainment: $50.00
   Includes: Comm Services: $100.00

Flight: DL 1866
LGA-ATL (2015/05/01 1400 - 2015/05/01 1640) 800 miles (1600 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 2.0
Total Fare: $1,000.00 (class Business)
   Includes Food/Beverage: $50.00 Entertainment: $50.00
```

```
Flight: DL 1868
LGA-ATL (2015/05/01 1500 - 2015/05/01 1740) 800 miles (1200 award miles)
Base Fare: $450.00 Fare Adjustment Factor: 1.0
Total Fare: $450.00 (class Economy)
   Includes Award Miles Factor: 1.5


 ----jGRASP: operation complete.
```

## Hints

1. Adding an element to a full array in your **addAirTicket** and **addInvalidRecord** methods – Consider the example below where MyType[] myArray is an instance field and addElement is an instance method that adds newElement to myArray, which is full. Since the length of an array cannot be changed after it has been created, myArray must be replaced with one that has a length of myArray.length + 1 and then elements from the original array must be copied to the new array. This copy operation could be done using a loop. However, Java.util.Arrays provides a copyOf method, which creates the new array and performs the copy in a single statement as shown in the first statement in the method below. The second statement adds newElement as the last element in the array.

   ```
   public void addElement(MyType newElement) {
      myArray = Arrays.copyOf(myArray, myArray.length + 1);
      myArray[myArray.length - 1] = newElement;
   }
   ```

2. The advantage to keeping the array full is that it allows the use of for-each loops with the array.

   ```
   for (MyType mt : myArray)
   {
      // do something with each mt
   }
   ```

3. In the readFile method, if you use a switch statement to determine the category, you should use type char for the switch expression rather than String; that is, each of the case labels should be of type char (e.g., `case 'E':` rather than type String (e.g., `case "E":`). When the switch type is String, the code coverage tool used by Web-CAT fails to detect that the default case is covered. If category is the reference to the String that contains the category code, then the following statement returns the category code as type char.

   ```
   category.charAt(0)
   ```