# Assignment 1

COMP 250 Winter 2023

| | |
|---|---|
| posted: | Friday, Jan. 27, 2023 |
| due: | Friday, Feb. 10, 2023 at 23:59 |

## Learning Objectives

This assignment is meant for you to practice what we have learned in class in the first four weeks of the semester. A lot of the design decision have been taken for you, but it is important for you to ask yourself why each choice has been made and whether there could be a better way of doing it. Some of our choices were influenced by the intention of having you practice most of what you have learned in class. Others, were dictated by our need to be able to fully test your assignments. Finally, we will not be grading your work on efficiency for the first assignments, but your implementation decisions can affect how efficient the code for this assignment is. Try to keep this in mind while coding. Make sure to ask yourselves why certain modifiers (`private`, `public`, `static`,...) were used and if you would have made the same decisions. We hope that the assignment will help you appreciate the importance of class design. This is of course just a taste, you will learn much more about it in COMP 303. As mentioned in class, we suggest you take the time to draw out a class diagram. This should help you develop a clear picture of the relationship between all these classes.

## General Instructions

- **Submission instructions**
  - Late assignments will be accepted up to 2 days late and will be penalized by 10 points per day. Note that submitting one minute late is the same as submitting 23 hours late. We will deduct 10 points for any student who has to resubmit after the due date (i.e. late) irrespective of the reason, be it wrong file submitted, wrong file format was submitted or any other reason. This policy will hold regardless of whether or not the student can provide proof that the assignment was indeed "done" on time.
  - Don't worry if you realize that you made a mistake after you submitted: you can submit multiple times but **only the latest submission will be evaluated**. We encourage you to submit a first version a few days before the deadline (computer crashes do happen and Ed Lessons may be overloaded during rush hours).
  - These are the files you must submit on Ed:
    - `Airport.java`
    - `Room.java`
    - `Hotel.java`

    - `Reservation.java`
    - `FlightReservation.java`
    - `HotelReservation.java`
    - `BnBReservation.java`
    - `Basket.java`
    - `Customer.java`

    **Do not submit any other files, especially .class files.** Any deviation from these requirements may lead to lost marks.

- Please make sure that all the files you submit are part of a package called `assignment1`.
- You will have to create all the above classes from scratch. The assignment shall be graded automatically. Requests to evaluate the assignment manually shall not be entertained, so please make sure that you follow the instruction closely or your code may fail to pass the automatic tests. Note that for this assignment, you are NOT allowed to import any other class (including for example `ArrayList` or `LinkedList`). **Any failure to comply with these rules will give you an automatic 0.**
- Whenever you submit your files to Ed, you will see the results of some exposed tests. These tests are a mini version of the tests we will be using to grade your work. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. Please note that these tests are only a subset of what we will be running on your submissions, we will test your code on a more challenging set of examples. Passing the exposed tests assures you that your submission will not receive a grade lower than 40/100. We highly encourage you to test your code thoroughly before submitting your final version.
- With this assignment, a class called `Minitester` is also posted. The tests included in this class are equivalent to those exposed on Ed. We encourage you modify and expand this class. ***You are welcome to share your tester code with other students on Ed.*** Try to identify tricky cases. Do **not** hand in your tester code.
- Your submission will automatically get a 0 if the code does not compile.
- Failure to comply with any of these rules will be penalized. If anything is unclear, it is up to you to clarify it by asking either directly a TA during office hours, or on the discussion board on Ed.
- IMPORTANT: Do NOT start testing your code only after you are done writing the entire assignment. It will be extremely hard to debug your program otherwise. If you need help debugging, feel free to reach out to the teaching staff. When doing so, make sure to mention what is the bug you are trying to fix, what have you tried to do to fix it, where have you isolated the error to be.

# Travel Agency

For this assignment you will write several classes to simulate an online travel agency. Note that for this assignment to be feasible in two weeks, a lot of simplifications have been made. For instance, we are completely ignoring time throughout the assignment. Make sure to follow the instructions below very closely. Note that in addition to the required methods, you are free to add as many other **private** methods as you want (no other additional method is allowed). You are **not** allowed to add any additional fields (whether they would be `private` or `public`) unless it is stated otherwise. Finally, **whenever you are required to compare two strings you can ignore the case of the characters**.

[**9 points**] Write a class `Airport`. An airport has the following `private` fields:

- An `int` indicating the x-coordinate of the airport on a world map with a scale to 1 km.
- An `int` indicating the y-coordinate of the airport on a world map with a scale to 1 km.
- An `int` indicating the airport fees (***in cents***) associated to this airport.

The class must also have the following `public` methods:

- A constructor that takes three `int` as input indicating the position of the airport on a map (x and y coordinate) and the fees of the airport (in cents) respectively. The constructor uses the inputs to initialize the corresponding fields.
- A `getFees` method to retrieve the fees of the airport.
- A `static` method `getDistance` which takes as input two `Airport`s and returns a integer indicating the distance in kilometer between the two. Note that the method should **round the distance up** (e.g. both 5.9 and 5.2 should become 6). More over, remember that given two points $(x_1, y_1)$ and $(x_2, y_2)$, the distance can be computed with the following formula:

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

[**12 points**] Write a class `Room`. An room has the following `private` fields:

- A `String` indicating the type of the room.
- An `int` indicating the price (in cents) of the room.
- A `boolean` indicating whether or not the room is available.

The class must also have the following `public` methods:

- A constructor that takes as input the type of the room and uses it to initialize the fields. Note that there are only 3 type of rooms supported by the program: *double*, *queen*, and *king*. If the input does not match one of these room type, then the constructor should throw an `IllegalArgumentException` explaining that no room of such type can be created. The price of the room is based on its type as follows: \$90 for a double, \$110 for a queen, \$150 for a king. Remember that the price should be stored in cents. The constructor should set the availability for a new room to be true.

- A constructor that takes a `Room` as input and creates a copy of the input room (i.e. it initialize the fields using the values from the corresponding fields of the input room). This type of constructors are called *copy constructors*.
- A `getType` and a `getPrice` method which return the type and the price of the room respectively.
- A `changeAvailability` method which takes no input and sets the value stored in the availability field to be the opposite of the one currently there.
- A `static` method `findAvailableRoom` which takes as input an array of `Room`s as well as a `String` indicating a room type. The method should return the *first* available room in the array of the indicated type. If no such room exists (either because all rooms of said type are occupied, or because no room of such type is in the array), the method returns `null`. **Note that, no changes to any of the rooms in the input array should be made by this method!**
- A `static` method `makeRoomAvailable` which takes as input an array of `Room`s as well as a `String` indicating a room type. The method should make the *first* unavailable room in the array of the indicated type available again. If successful, the method should return `true`, otherwise the method should return `false`.

**[15 points]** Write a class `Hotel`. An hotel has the following `private` fields:

- An `String` indicating the name of the hotel.
- An array of `Room`s indicating the rooms in the hotel.

The class must also have the following `public` methods:

- A constructor that takes a `String` and an array of `Room`s respectively. The constructor uses the inputs to initialize the corresponding fields. Note that the array used to initialize the field representing the rooms should be a ***deep copy*** of the input array.
- A `reserveRoom` method which takes as input a `String` indicating the type of room to reserve. The method changes the availability of the first available room of the specified type in the hotel. If successful, the method returns the price of the room. Otherwise, an `IllegalArgumentException` should be thrown.
- A method `cancelRoom` which takes as input a `String` indicating the type of room to cancel. The method makes a room of that type available again. It returns `true` if it operation was possible, `false` otherwise.

**[5 points]** Write an `abstract` class `Reservation` which has the following `private` field:

- A `String` name

The class must also have the following `public` methods:

- A constructor that takes a `String` as input indicating the name of the client on the reservation and uses it to initialize the corresponding field.
- A `final reservationName` method to retrieve the name on this reservation.

- An `abstract` method `getCost` which takes no input and returns an `int`. This method should be abstract (thus, not implemented) because how to determine the cost depends on the type of reservation.
- An `abstract` method `equals` which takes an `Object` as an input and returns a `boolean`. This method should be abstract as well, since depending on the type of reservation different conditions should be met in order for two reservations to be considered equal.

**[25 points]** All of the followings must be subclasses of `Reservation`:

- Write a class `FlightReservation` derived from the `Reservation` class. This class has the following `private` fields:
  - An `Airport` indicating the place of departure.
  - An `Airport` indicating the place of arrival.

  The class has also the following `public` methods:
  - A constructor that takes as input a `String` with the name on the reservation, and two `Airport`s indicating the place of departure and arrival respectively. The constructor uses the inputs to create a `Reservation` and initialize the corresponding fields. Throw an `IllegalArgumentException` if the two input airports are the same. Ignore the fact that you did not overrode the `equals` method in the `Airport` class.
  - A `getCost` method that takes no input and returns the cost of the reservation (an `int`) in cents. The cost is computed adding together the fuel cost, the airports' fees, and \$53.75 (which include costs related to the plane plus taxes). To compute the fuel cost, you can assume that [1]:
    - Planes pay \$1.24 per gallon of fuel.
    - Planes can fly 167.52 kilometer per gallon of fuel.

    The cost should be **rounded up** to the nearest cent. For example, if after the computation above you obtain a flight cost of 103568.21187 cents, the method should return 103569 cents instead. You may assume that the cost of all reservations fits within an int and therefore doesn't cause overflow.
  - An `equals` method which takes as input an `Object` and return `true` if input matches `this` in type, name, and airports. Otherwise the method returns `false`. You can ignore the fact that you did not overrode the `equals` method in the `Airport` class.
- Write a class `HotelReservation` derived from the `Reservation` class. This class has the following `private` fields:
  - An `Hotel` indicating where to make the reservation.
  - A `String` indicating the type of room to reserve.
  - An `int` indicating the number of nights to be spent in the hotel.

[1] See: https://youtu.be/6Oe8T3AvydU

  - An `int` indicating the price (in cents) for one night in a room of the specified type in the specified hotel.

  The class has also the following `public` methods:

  - A constructor that takes as input a `String` with the name on the reservation, a `Hotel`, a `String` with the room type, and an `int` indicating the number of nights. The constructor uses the inputs to create a `Reservation` and initialize the corresponding fields. The constructor should also make sure to reserve a room of the correct type in the specified hotel. If such reservation is not possible, then an `IllegalArgumentException` should be raised.
  - A `getNumOfNights` method to retrieve the number of night on the reservation.
  - A `getCost` method that takes no input and returns the cost of the reservation (an `int`) in cents. The cost represents the price to pay for the specified type of room given the number of nights indicated in the reservation.
  - An `equals` method which takes as input an `Object` and return `true` if input matches `this` in type, name, hotel, room type, number of nights, and total cost. Otherwise the method returns `false`. Once again, you can ignore the fact that you did not override the `equals` method in the `Hotel` class.

- Write a class `BnBReservation` derived from the `HotelReservation` class. This class has no fields, but it has the following `public` methods:
  - A constructor that takes as input a `String` with the name on the reservation, a `Hotel`, a `String` with the room type, and an `int` indicating the number of nights. The constructor uses the inputs to create an `HotelReservation`.
  - A `getCost` method that takes no input and returns the cost of the reservation (an `int`) in cents. Since this reservation includes breakfast, to the cost of reserving the room in the hotel you should add $10 per night.

**[18 points]** Write a class `Basket` representing a list of reservations. Note that the instructions on how to implement this class are not always very specific. This is intentional, since your assignment will not be tested on the missing details of the implementation. Note though, that your choices will make a difference in terms of how efficient your code will be. We will not be deducting point for inefficient code in Assignment 1. Note once again that, you are NOT allowed to import any other class (including `ArrayList` or `LinkedList`). The class has (at least) the following `private` field (you can add additional `private` fields to this class if you wish):

- An array of `Reservation`s.

The class must also have the following `public` methods:

- A constructor that takes no inputs and initialize the field with an array with no reservations.

- A `getProducts` method which takes no inputs and returns ***a shallow copy*** of the array of `Reservation`s of the basket. This array should contain all the reservations in the basket in the same order in which they were added.
- An `add` method which takes as input a `Reservation`. The method adds the reservation **at the end** of the list of reservation of the basket and returns how many reservations are now there.
- A `remove` method which takes as input a `Reservation` and returns a `boolean`. The method removes the **first** occurrence of the specified element from the array of reservation of the basket. If no such reservation exists, then the method returns `false`, otherwise, after removing it, the method returns `true`. Note that this method removes **a reservation from the list if and only if such reservation is *equal* to the input received**. For example, two flight reservations from Montreal to Vancouver are not considered equal if they were created under two different names. After the reservation has been removed from the array, the subsequent elements should be shifted up by one position, **leaving no unutilized slots in the middle array**.
- A `clear` method which takes no inputs, returns no values, and empties the array of reservations of the basket.
- A `getNumOfReservations` method that takes no inputs and returns the number of reservations in the basket.
- A `getTotalCost` method that takes no inputs and returns the cost (in cents) of all the reservations in the basket.

**[16 points]** Write a class `Customer` which has the following `private` fields:

- A `String` name
- An `int` representing the balance (in cents) of the customer
- A `Basket` containing the reservations the customer would like to make.

The class must also have the following `public` methods:

- A constructor that takes as input a `String` indicating the name of the customer, and an `int` representing their initial balance. The constructor uses its inputs and creates an empty `Basket` to initialize the corresponding fields.
- A `getName` and a `getBalance` method which return the name and balance (in cents) of the customer respectively.
- A `getBasket` method which returns the reference to the basket of the customer (no copy of the basket is needed).
- An `addFunds` method which takes an `int` as input representing the amount of cents to be added to the balance of the customer. If the input received is negative, the method should throw an `IllegalArgumentException` with an appropriate message. Otherwise, the method will simply update the balance and return the new balance in cents.

- An `addToBasket` method which takes a `Reservation` as input and adds it to the basket of the customer if the name on the reservation matches the name of the customer. If the method is successful it should return the number of reservations in the basket of this customer. Otherwise, the method should throw an `IllegalArgumentException`.
- An `addToBasket` method which takes a `Hotel`, a `String` representing a room type, an `int` representing the number of nights, and a `boolean` representing whether or not the customer wants breakfast to be included. The method adds the corresponding reservation to the basket of the customer and returns the number of reservations that are now in the basket of this customer.
- An `addToBasket` method which takes two `Airport`s as input. The method adds the corresponding reservation to the basket of the customer and returns the number of reservations that are now in their basket. If the the flight reservation could not be created successfully, the method should still return the number of reservations in the basket after its execution.
- A `removeFromBasket` method which takes a `Reservation` as input and removes it from the basket of the customer. The method returns a boolean indicating whether or not the operation was successful.
- A `checkOut` method which takes no input. If the customer's balance is not enough to cover the total cost of their basket, then the method throws an `IllegalStateException`. Otherwise, the customer is charged the total cost of the basket, **the basket is cleared**, and balance left is returned.