

Assignment2

[Submit Assignment](#)

Due Monday by 12pm **Points** 15 **Submitting** a file upload
Available after Jan 5 at 7:30pm

Java second assignment

You will be implementing a travel in Europe project. Make the code as good quality as you can. Follow the Java conventions when it comes to styling/formatting your code (correct indentation, standard capitalization in names, etc.) and other best practices (proper encapsulation, closing all opened files, etc.).

Task 1: DateAndTime, Destination, DestinationUtils, Flight (3 Points)

Create the `assignment2.travel.DateAndTime` class with the following fields that are not accessible from other classes:

``year``, ``month``, ``day``, ``hour``, and ``minute`` all of type ``int``. All of these fields have getters.

- The class has two constructors:

1. A constructor that takes all the fields, and sets their values accordingly.

2. A constructor that takes no parameters. It calls the previously defined constructor with the following values:

- ``year``: 2021
- ``month``: 12
- ``day``: 22
- ``hour``: 04
- ``minute``: 30

- Hint: Calling an other constructor can be done with the ``this(parameters)``.

- Create an implementation the standard method that creates the textual representation of a ``DateAndTime`` object returns a text like this: ``2021.12.22 at 4:30``

- Create ``getTime()`` which returns a text like this: ``4:30``

Create an enum `assignment2.travel.Destination` which includes five destinations: ``BERLIN``, ``ROME``, ``AMSTERDAM``, ``PARIS``, and ``HELSINKI``.

Create a class `assignment2.travel.DestinationUtils`.

The methods in ``DestinationUtils`` are the following:

- Its static method ``getDestination()`` takes a ``String`` argument which represents a travel duration.

The string is formatted as ``hour:minute`` (you **don't** have to check if it's formatted properly);

the method returns the proper `assignment2.travel.Destination` value based on the following durations:

```

~~~
BERLIN      01:34
ROME        01:45
AMSTERDAM   02:05
PARIS       02:20
HELSINKI    02:43
~~~

```

- For example, the return value of ``getDestination("01:45")`` is ``Destination.ROME``.

- If the duration is not listed above, the method returns ``null``.

- Note that it is very bad to return ``null``, and you shouldn't do it in real code.

- Its static method `getDestinationDuration()` takes a `Destination` instance and returns the proper duration as string from the table above.
- Its static method `getRoundedHours()` takes a `Destination` instance and returns the number of hours in the duration associated to the argument, plus one more if it contains at least `30` minutes.
 - Hint: partition the duration text along the `:` character returned by `getDestinationDuration()`, then convert the two parts to integers.

Create an `assignment2.travel.flying.Flight` class with the following protected fields: `name` of type `String`, `destinationCity` of type `Destination`, `numberOfTravellers` of type `int`, and `flightDateAndTime` of type `DateAndTime`.

- Create getters for all of the fields, the `destinationCity` getter should return the name of the constant, while the `flightDateAndTime` getter should return its textual representation.

- `Flight` has two constructors:

1. A constructor that takes all the fields, and sets their values accordingly.
 - Before doing so, check whether `numberOfTravellers` is at least `15` and at most `100`. If it is outside of this range, throw an `IllegalArgumentException`.

2. A constructor that takes no parameters. It calls the other constructor with the following values: `AirBus`, `ROME`, `83`, and a `DateAndTime` instance initialised using the empty constructor

- Create `getFlightDuration()` which returns `String` value, represent the flight duration. (you need to use one of the enum's method here)
- The standard textual representation of `Flight` objects has to look like this: `Flying AirBus with 83 passengers to ROME on 2021.12.22 at 11:42`

Task 2: Flyable, Plane (3 Points)

Create the interface `assignment2.travel.flying.Flyable` with the method `estimatedArrivalTime` that takes a `Destination` and `departHour`. and the method `getPrice` that takes `discountRate` of type `double` and returns a `double` value.

Create the class `assignment2.travel.flying.Plane`, a child class of `Flight` that also implements `Flyable`. It has the following fields: `name` of type `String`, `id` of type `int`, and `ticketPrice` of type `int`. All fields have getters, but not setters.

- Its single `private` constructor takes values for its fields and sets their values.
 - It throws an `IllegalArgumentException` if `name` is `null` or `ticketPrice` is less than 10.
- Write a static method `make` which takes `data` of type `String` as argument, and returns `Plane` type. `data` is in the form `name,id,ticketPrice`.
 - You do not need to check the format, you may assume that the input data is OK.
 - The method returns a `Plane` object initialized with the three components of `data`.
 - Hint: take `data` apart along the occurrences of the `,` text, then convert the resulting text sections as necessary.
- Two instances of the class are considered equal in content if the values of the three fields are matching, make sure that you override the appropriate method.
 - Also override `hashCode()` based on the values of the three fields.
- Make the textual representation of an instance very simple: `NameOfThePlane,12,83` if the `id` value is `12` and the ticket price is `83`.
- Implement the `estimatedArrivalTime()`. It returns the (integer) hour that is the estimated flight hours later.
 - Example: if the plane departs at `15` hours, and the destination is `01:45` away, the return value is `17`.
 - You may suppose that the departure and arrival happen on the same day.

- The `getPrice()` takes a `discountRate`, a `double`. It returns the `double` price which is the discounted ticket price.

- Example: if the `ticketPrice` is `100`, and the `discountRate` is `0.2`, the return value is `80`.

Task 3: FlightWithManyPlanes (3 Points)

Create `assignment2.travel.flying.FlightWithManyPlanes`, a child class of `Flight`. It has a field called `planes`, a list of `Plane`'s.

- It has one constructor takes all the fields of the (non-zero-arg) `Flight` constructor, and an arbitrary number of `Plane`'s (possibly zero, possibly a thousand of them) in an array.

- Invoke the constructor of the base class with the arguments.

- Fill the list with the planes with the elements of the list.

- Create `save()` which takes a `filename`. Open the file and write the following content into it.

- The first four lines contain the textual representations of `name`, the enum value name of `destination` (hint: enums have a `name()` method), `numberOfTravellers`, and `flightDateAndTime`.

- The remaining lines each contain the textual representation of a plane.

- Create `load()` which takes a `filename`. Open the file and read the following content from it. You may assume that the file exists, and it contains properly formatted data.

- Read the first three lines and decode them into the appropriate fields.

- As decoding a `DateAndTime` would be slightly more complex, you may skip the fourth line. That is, read it and ignore it.

- Empty `planes`. Then read all remaining lines, and add them as new planes.

- Let both `load()` and `save()` throw `IOException` exception to appease the compiler.

- Create method `getCheapestRide` which takes an argument of type `double`, `discountRateIncrease`.

- The method throws an `IllegalStateException` if the number of Planes is zero.

- Otherwise, it finds the plane with the lowest price.

- The first plane is not discounted at all.

- With each further plane, the discount rate increases by `discountRateIncrease`.

- So, the second plane is discounted at `discountRateIncrease`, the third one is at twice `discountRateIncrease` etc.

- The method returns the reference of the cheapest plane.

Task 4: Testing (6 Points)

- Testing on Task1:

- Both constructors of `DateAndTime`. Check for the year, month, day, hour, and minute in each test.

- `DestinationUtils`. one test case for each method.

- Testing on Task2:

- test cases for `equals()` and `hashCode()` methods.

- two test cases for `estimatedArrivalTime()` method.

- Testing on Task3:

- two test cases for `load()` method.

- two test cases for `save()` method.