BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
K. K. BIRLA GOA CAMPUS
FIRST SEMESTER 2024-2025

CS F213 (Object Oriented Programming) - Lab 5, Date: 04/09/24

Lab Problem Statement

Part 1: Car Classes

Modify abstract class `Car` with the following attributes and methods:

- Attributes:
  - `String model`
  - `int speed`
  - `int fuelConsumption` (fuel consumed per kilometer)
  - `int fuel` (initial fuel available)
  - `int distanceCovered` (distance covered in the test)
- Methods:
  - Abstract method `void accelerate()`
  - Abstract method `void brake()`
  - Method `void drive(int distance)` to drive the car for a specified distance, adjusting fuel and distance covered
    Fuel Consumed=Distance Driven×Fuel Consumption Rate
  - Getter and setter methods for all attributes

Modify the following subclasses of `Car`:

`Sedan`:

- **Speed and Fuel Consumption:**
  - Moderate speed (starting at 50 km/h)
  - Fuel consumption rate of `2` liters per kilometer
- **Constructor:**
  - Takes parameter `int fuel` (for initial fuel)
    and intialise attributes with values stated above.
  - **Overrides `accelerate()`:**
    - Increases speed by `10` km/h each time it is called
- **Overrides `brake()`:**
  - Decreases speed by `10` km/h each time it is called

`SUV`:

- **Speed and Fuel Consumption:**
  - Lower speed (starting at 40 km/h)
  - Higher fuel consumption rate of `3` liters per kilometer
- **Constructor:**
  - Takes parameter `int fuel` (for initial fuel)

  and intialise attributes with values stated above.
- **Overrides `accelerate()`:**
  - Increases speed by `5` km/h each time it is called
- **Overrides `brake()`:**
  - Decreases speed by `15` km/h each time it is called

`SportsCar`:

- **Speed and Fuel Consumption:**
  - High speed (starting at 70 km/h)
    - Low fuel efficiency with a fuel consumption rate of `4` liters per kilometer
- **Constructor:**
  - Takes parameter `int fuel` (for initial fuel)

  and intialise attributes with values stated above.
- **Overrides `accelerate()`:**
  - Increases speed by `20` km/h each time it is called
- **Overrides `brake()`:**
  - Decreases speed by `5` km/h each time it is called

Part 2: CommandProcessor Class

Modify `CommandProcessor` class to handle the input string:

- **Attributes:**
  - `String[] commands` (to store the split commands)
  - `Car car` (to store the car object)
- **Methods:**
  - `void processInput(String input)`:
    - Split the input string by commas and store in `commands[]`
    - Create the appropriate `Car` object based on the first element (e.g., "Sedan", "SUV", "SportsCar")
    - Set the initial fuel using the second element

        - Loop through the remaining elements and execute corresponding methods (`drive`, `accelerate`, `brake`)
    - `Car createCar(String type, int fuel)`: A helper method to create and initialize the correct `Car` object.
    - `void executeCommand(String command)`: A helper method to execute a command based on the input string.

Input:

The input is a single line of comma-separated values, structured as follows:

1. The first value is the car type: "Sedan", "SUV", or "SportsCar"
2. The second value is the initial fuel amount (an integer)
3. The remaining values are commands, which can be:
    - "drive X" where X is the distance to drive
    - "accelerate" to increase the car's speed
    - "brake" to decrease the car's speed

Output:
The output is a multi-line string containing the following information:

<CarType> - Model: <CarType>
Final Speed: <speed> km/h
Fuel Remaining: <fuel>
Distance Covered: <distance> km

Example:

Input 1:

SportsCar,5,drive 4,accelerate,drive 1,brake

Output 1:

SportsCar - Model: SportsCar
Final Speed: 70 km/h
Fuel Remaining: 0
Distance Covered: 1 km

Input 2:

SUV,100,accelerate,drive 20,brake,drive 15,accelerate,drive 30,brake,drive 5,accelerate,drive 25,brake

Output 2:

SUV - Model: SUV
Final Speed: 30 km/h
Fuel Remaining: 0
Distance Covered: 33 km

Instructions:

Follow the steps given below to complete the OOP lab problem.

Step 1: Read Lab Question

Read and understand the Lab problem given above.

Step 2: Edit the Solution java file

Edit the solution java file to solve the given problem. For Lab 5, the solution java file is named: `L5_Q1_Soln.java`. Note that `L5` refers to Lab 5 and `Q1` refers to Question 1. The exact name of the solution java file will be dependent on the Lab and Question numbers.

Step 3: See the input and the expected output

Use the following command to see the input and the expected output for test case `T1`. Note that `L5` refers to Lab 5, `Q1` refers to Question 1 and `T1` refers to test case 1.

```
:~$ ./RunTestCase L5 Q1 YourBITSId T1
```

Use your own (13 character) BITS Id in place of `YourBITSId` in the above command. Type your BITS Id in upper case (capital letters). Ensure that you enter your BITS Id in 202XA7PSXXXXG format.

Run the command from within the folder containing the executables and the java files.

Step 4: Modify the solution java file

Modify the solution Java file and repeat Step 3 until all the test cases are passed. The test cases are numbered `T1`, `T2`, etc.

Step 5: Passing evaluative test cases

There are evaluative test cases whose expected output is masked. The evaluative test cases are numbered `ET1`, `ET2`, etc. The lab marks will be based on the evaluative test cases. Use the following command to check whether your solution passes the evaluative test cases.

```
:~$ ./RunTestCase L5 Q1 YourBITSId ET1
```

Ensure that your solution passes all the evaluative test cases ET1, ET2, etc. The expected output of the evaluative test cases are *hidden*; therefore, only the hash value is shown.

Step 6: Create submission zip file

Use the following command to create the submission zip file.

```
:~$ ./CreateSubmission L5 YourBITSId
```

The first command line argument must correspond to the lab number and the second command line argument must be your 13 character BITS id.

The above command will create a zip file. The command will also list the evaluative test cases that your solution program has passed.

Step 7: Upload submission zip file

Upload the submission zip file created in Step 6. Do not change the name of the zip file or modify any file inside the zip file. You will not be awarded marks if the zip file is tampered with in any manner.