CPSC 121
Project 01: Parking Finder

Project Overview

In this project, you will develop an application that randomly generates four parking spots and determines which parking spot is the closest to your current location. **You can only use material covered in these instructions and/or in this course up through Week 07.**

Objectives

1. Write a Java application (`ParkingFinder`).
2. Use an existing, custom class (`edu.cwi.parking.ParkingSpot`).
3. Use classes from the Java standard library.
4. Work with numeric data and its formatting.
5. Work with standard input and output.

Specification

Existing class that you will use: `edu.cwi.parking.ParkingSpot` (in `Parking.jar`)
Class that you will create: `ParkingFinder.java`

Getting Started

1. Create a new folder called `Project_01` for this project, create a folder called `lib` within that folder, and copy/paste `Parking.jar` into your project.
2. Create a new Java application called `ParkingFinder` within the project folder.
3. Read the `ParkingSpot` documentation below (and at *https://lmsevigny.github.io/CPSC-121/ParkingSpot/edu/cwi/parking/ParkingSpot.html*) on this new class to familiarize yourself with what it contains and how it works before writing any code of your own.
4. Start implementing your `ParkingFinder` class according to the specifications. TEST OFTEN!
5. You may need some methods from the `Math` class, such as `Math.abs`, `Math.ceil`, and `Math.min`.

ParkingFinder (the driver class)

In this class, you will create a random starting coordinate (your car's position) and four ParkingSpot objects with random location coordinates. After your car's position is determined and the spots are created, you will calculate how much it would cost to park in each spot for a given amount of time and use conditional statements to determine which spot is closest to you.

Your ParkingFinder code should not duplicate any data or functionality that belongs to the ParkingSpot objects. Make sure you are accessing parking spot data (like x and y position, cost per interval, etc.) using the ParkingSpot methods. DO NOT use a loop or array to generate your ParkingSpots. We will be discussing loops and arrays later, but do not use them now and please do not over complicate things; you will be docked points if you do.

Program Logic

1. User Input
    a) Note - console input should be handled with a single `Scanner` object. You should **never** make more than one Scanner from `System.in` in any program. There is only one keyboard, there should only be one `Scanner` for it.
    b) Your program will prompt the user for the following values and read the user's input from the keyboard and assign the values to appropriate variables you will need to declare.
        i. A long value to seed your Random number generator. The seed value read from the keyboard should be used to initialize your Random number generator. Look at the different constructors available in the Random class to recall how seed values are used. You can use the `nextLong()` method of the Scanner class to read the seed for the random number generator.
        ii. An int value for the amount of parking time needed in minutes. You can use the `nextInt()` method of the Scanner class to read the parking time.
    c) After prompting the user, reading, and storing these values, move to step 2.

2. Generating the car's random location
    a. Declare and initialize two random integers (carX and carY) in the range 0-99 (inclusive) to designate the driver's starting position. Use your seeded Random object to generate the random coordinates. These must be generated before your random parking spots in order for your output to match the sample output.
    b. Print the position of the vehicle. Your output should be (with numbers included):
    `The position of your vehicle is:  X: ## Y: ##`
3. Generating random parking spots
    a. Now, create (declare, instantiate, and assign) four parking spot objects at random x, y coordinates.
        i. Use the following process to create each new spot.
            - Use your existing seeded Random object to generate random x, y coordinates. The x and y coordinates must be in the range 0-99 inclusive.
            - Use the ParkingSpot constructor to instantiate a new spot with a street name (make this anything you like) and the random x, y coordinates.
        ii. Let the first two parking spots have the default cost per interval and set the cost per interval of the last two spots to $0.30 per 10 minute interval. You can set the cost per interval for a particular spot using the `setCostPerInterval()` method of the ParkingSpot class.
        iii. Before moving to the next step, print your generated parking spots to make sure everything looks correct. To print a spot, you can just use the parking spot object in a `System.out.println()` with String concatenation (if needed). Look at the ParkingSpot documentation for an example of how to print a parking spot.

        Note – You should not "permanently" store any of the data for your parking spots in variables in the main method. Each ParkingSpot instance you create will store its own data, so you can use the methods of the ParkingSpot class to get

that data when you need it. For example, once you set the x location of your spot, you may retrieve it using `spot1.getLocationX();`

4. Calculate the distance and parking charge. You must do the following for each parking spot.
   a. Calculate the distance from the driver's car.
      i. The distance is calculated using Manhattan geometry.
      ii. In Manhattan geometry (also known as taxicab geometry), the distance between two points (x1, y1) and (x2, y2) is the sum of the absolute value of the differences between the x and y coordinates.
         - distance = $|x_1 - x_2| + |y_1 - y_2|$
   b. Calculate the cost to park in the spot for the time requested.
      i. The driver is charged per 10 minute period (INTERVAL), so any time between 1 and 10 minutes is counted as a full 10 minutes. In other words, if the charge is $0.20 per 10 minutes, it would cost $0.40 to park for 11 minutes. You may need to check out the `Math.ceil()` method.
   c. Print the details of the spot along with the distance and cost to park. Format the output as shown in example below. *Note – use the* `ParkingSpot` *object along with String concatenation to print the parking spot details; though the* `toString()` *method describes how the object will appear as a String, there is generally no need to call this method directly... just use the object reference directly.*

```
Spot 1: [Street = 1st Street, location = 33, location = 20, available = true, costPerInterval =
$0.25]
        Distance:  18
        Total Cost:  $0.75
```

      i. Use the `NumberFormat` currency formatter to print the formatted cost.

5. Find the closest spot
   a. Determine which parking spot is the closest out of the four spots.
   b. Print the distance to the closest spot and the details of the closest spot.

```
Distance to closest spot:  73
Closest spot:  [Street = 1st Street, location = 33, location = 20, available = true,
costPerInterval = $0.25]
```

Sample Output:

```
Enter your random seed:
1234
Enter the necessary parking time (minutes):
30
The position of your vehicle is:  X: 28 Y: 33

Spot 1: [street = 1st Street, locationX = 33, locationY = 20, available = true, costPerInterval = 0.25]
        Distance to spot1 is: 18
        Total Cost for Spot 1 is: $0.75

Spot 2: [street = 2nd Street, locationX = 10, locationY = 93, available = true, costPerInterval = 0.25]
        Distance to spot2 is: 78
        Total Cost for Spot 2 is: $0.75

Spot 3: [street = 3rd Street, locationX = 29, locationY = 49, available = true, costPerInterval = 0.3]
        Distance to spot3 is: 17
        Total Cost for Spot 3 is: $0.90

Spot 4: [street = 4th Street, locationX = 97, locationY = 37, available = true, costPerInterval = 0.3]
        Distance to spot4 is: 73
        Total Cost for Spot 4 is: $0.90

The distance to the closest spot is: 17
The closest spot is: [street = 3rd Street, locationX = 29, locationY = 49, available = true, costPerInterval = 0.3]


Enter your random seed:
1234
Enter the necessary parking time (minutes):
45
The position of your vehicle is:  X: 28 Y: 33

Spot 1: [street = 1st Street, locationX = 33, locationY = 20, available = true, costPerInterval = 0.25]
        Distance to spot1 is: 18
        Total Cost for Spot 1 is: $1.25

Spot 2: [street = 2nd Street, locationX = 10, locationY = 93, available = true, costPerInterval = 0.25]
        Distance to spot2 is: 78
        Total Cost for Spot 2 is: $1.25

Spot 3: [street = 3rd Street, locationX = 29, locationY = 49, available = true, costPerInterval = 0.3]
        Distance to spot3 is: 17
        Total Cost for Spot 3 is: $1.50

Spot 4: [street = 4th Street, locationX = 97, locationY = 37, available = true, costPerInterval = 0.3]
        Distance to spot4 is: 73
        Total Cost for Spot 4 is: $1.50

The distance to the closest spot is: 17
The closest spot is: [street = 3rd Street, locationX = 29, locationY = 49, available = true, costPerInterval = 0.3]
Enter your random seed:
2232
Enter the necessary parking time (minutes):
21
The position of your vehicle is:  X: 76 Y: 8

Spot 1: [street = 1st Street, locationX = 88, locationY = 70, available = true, costPerInterval = 0.25]
        Distance to spot1 is: 74
        Total Cost for Spot 1 is: $0.75

Spot 2: [street = 2nd Street, locationX = 19, locationY = 55, available = true, costPerInterval = 0.25]
        Distance to spot2 is: 104
        Total Cost for Spot 2 is: $0.75

Spot 3: [street = 3rd Street, locationX = 92, locationY = 82, available = true, costPerInterval = 0.3]
        Distance to spot3 is: 90
        Total Cost for Spot 3 is: $0.90

Spot 4: [street = 4th Street, locationX = 88, locationY = 98, available = true, costPerInterval = 0.3]
        Distance to spot4 is: 102
        Total Cost for Spot 4 is: $0.90

The distance to the closest spot is: 74
The closest spot is: [street = 1st Street, locationX = 88, locationY = 70, available = true, costPerInterval = 0.25]
```

`ParkingSpot` (provided in `Parking.jar`)

You will just use this class in your `ParkingFinder` driver class as if it were part of the Java API (remember you will need an import since the package name is: `edu.cwi.parking` *).*

The `ParkingSpot` class has the following methods available for you to use:

1. `public ParkingSpot(String street, int locationX, int locationY)` – Constructor
2. `public boolean isAvailable()` – not used in this project
3. `public boolean setAvailable(boolean available)` – not used in this project
4. `public void setCostPerInterval(double cost)`
5. `public double getCostPerInterval()`
6. `public String getStreet()`
7. `public int getLocationX()`
8. `public int getLocationY()`
9. `public String toString()` – used indirectly when printed

The `ParkingSpot` class has the following constants available for you to use:

1. `public final double DEFAULT_COST`
2. `public final int INTERVAL`

To find out what each method does and what each constant represents, look at the documentation of the `ParkingSpot` class itself (https://lmsevigny.github.io/CPSC-121/ParkingSpot/edu/cwi/parking/ParkingSpot.html).

Overview

You are given a class named ParkingSpot that keeps track of the details about a parking spot. Each parking spot instance will keep track of the following data:

- Street name of the spot.
  - The street name must be specified when the spot is created. Street names should be given based on the order they are read in (e.g. "1st Street", "2nd Street", "3rd Street", "4th Street").
- Coordinates in a two-dimensional city grid.
  - The (x, y) coordinates must be specified when the spot is created.
- Whether or not the spot is available.
  - When a new parking spot is created, it is marked as available.
- The charge per 10 minute interval.
  - The default charge is $0.25 per 10 minute interval. As with all parking meters, payment must be in whole intervals.

Submitting the Lab

Compress the submission file (see below) into a .zip file named: **Project01_*LastName_FirstName*.zip**
When you are done and ready to submit, submit the following files in the .zip file:

- `ParkingFinder.java`