Programming Language Concepts 2026

Assignment 2

7 Apr 2026, due 20 Apr 2026

Problem 1

The new CMI amusement park coming up at Siruseri will have a bumper car ride where people drive cars that bang and bounce off each other on a special driving floor. Each car holds one rider. At most five cars are allowed on the driving floor at a time.

Cars and riders are identified by unique integer ids. Empty cars and riders waiting for a ride form two separate queues. Initially, the cars are lined up in order $1, 2, ..., M$, where M is the total number of cars available.

The length of each rider's ride on the floor is fixed randomly when the rider enters the waiting queue. When there are fewer than five cars on the floor, the first rider in the rider queue gets into the first car in the car queue, goes onto the floor and rides for the specified length of the time.

After the ride is over, the car returns to the rear end of the car queue. The rider gets off. The empty car stays in queue and will be used again when it comes to the front of the queue.

The rider who has got off walks around the park for a random amount of time before rejoining the line for his next ride. Each rider takes between 2 and 4 rides before going home.

Write a Java class `DrivingFloor` that coordinates the rider and car queues and services all riding requests. Riders line up by calling the method

```
takeAride (int rider_id, int ride_time)
```

where `rider_id` is the integer id of the rider and `ride_time` is the length of time that this rider's ride will last.

The method `takeAride` should print out diagnostic messages about the status of the riders and cars at the following times:

1. When a rider joins the line.

   e.g.

   ```
   Rider 3 joins queue at Thu Apr 09 11:53:10 IST 2026
   Car queue: [5, 6]
   Rider queue: [4, 3]
   Cars on the floor: 4
   ```

2. When a rider gets into the car and starts a ride.

   e.g.

   ```
   Rider 4 starts ride in car 5 at Thu Apr 09 11:53:10 IST 2026
   Car queue: [6]
   Rider queue: [3]
   Cars on the floor: 5
   ```

3. When a rider finishes the ride.

   e.g.

   ```
   Rider 7 finishes ride at Thu Apr 09 11:53:11 IST 2026
   Car 2 joins queue at Thu Apr 09 11:53:11 IST 2026
   Car queue: [6, 2]
   Rider queue: [3, 2, 1]
   Cars on the floor: 4
   ```

To print out the time in the diagnostic message, use the class `java.util.Date` as follows.

```
import java.util.*;
...
Date date;
...
date = new Date();
System.out.println("..."+date);
```

Hint The class `DrivingFloor` maintains the state of the system: which cars and riders are in queue and who is on the driving floor. Implement

```
takeAride (int rider_id, int ride_time)
```

using a sequence of synchronized methods that ensure that all changes of state are made in a safe manner.

The ride itself is modelled using `Thread.sleep()`.

See the attached presentation `Concurrent-Programming-Example.pdf` for a worked out example of a similar problem.

Sample output

```
Rider 0 joins queue at Thu Apr 09 11:56:53 IST 2026
Car queue: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Rider queue: [0]
Cars on the floor: 0

Rider 12 joins queue at Thu Apr 09 11:56:53 IST 2026
Car queue: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Rider queue: [0, 12]
Cars on the floor: 0

Rider 0 starts ride in car 1 at Thu Apr 09 11:56:53 IST 2026
Car queue: [2, 3, 4, 5, 6, 7, 8, 9, 10]
Rider queue: [12]
Cars on the floor: 1

Rider 11 joins queue at Thu Apr 09 11:56:53 IST 2026
Car queue: [2, 3, 4, 5, 6, 7, 8, 9, 10]
Rider queue: [12, 11]
Cars on the floor: 1

Rider 12 starts ride in car 2 at Thu Apr 09 11:56:53 IST 2026
Car queue: [3, 4, 5, 6, 7, 8, 9, 10]
Rider queue: [11]
Cars on the floor: 2

Rider 10 joins queue at Thu Apr 09 11:56:53 IST 2026
Car queue: [3, 4, 5, 6, 7, 8, 9, 10]
Rider queue: [11, 10]
Cars on the floor: 2

Rider 11 starts ride in car 3 at Thu Apr 09 11:56:53 IST 2026
Car queue: [4, 5, 6, 7, 8, 9, 10]
Rider queue: [10]
Cars on the floor: 3

Rider 9 joins queue at Thu Apr 09 11:56:53 IST 2026
Car queue: [4, 5, 6, 7, 8, 9, 10]
Rider queue: [10, 9]
Cars on the floor: 3
```

```
Rider 10 starts ride in car 4 at Thu Apr 09 11:56:53 IST 2026
Car queue: [5, 6, 7, 8, 9, 10]
Rider queue: [9]
Cars on the floor: 4

...

Rider 10 finishes ride at Thu Apr 09 11:56:55 IST 2026
Car 3 joins queue at Thu Apr 09 11:56:55 IST 2026
Car queue: [6, 7, 8, 9, 10, 3]
Rider queue: [8, 7, 6, 5, 4, 3, 2, 1]
Cars on the floor: 4

...
```

Attached are two files:

1. `Rider.java`: defines a `Rider` class that implements `Runnable`
   - The instance variables define a distinct numeric id for each rider, the number of times this rider will take a ride and a reference to the Driving Floor to be used.
   - Each rider performs a loop consisting of taking a ride using the method takeAride(...) followed by a walk around the amusement park, modelled by sleeping.
2. `Assignment2.java`: Creates a random number of `Riders` and sets them off in parallel.

Add your class `DrivingFloor.java` and compile and run `Assignment2.java` to test your code.

Problem 2

Consider a distributed system with a set of N processes numbered $\{1, 2, ..., N\}$ connected in a ring — each process i sends messages to its neighbour $(i + 1)$ mod N through a one-way channel.

A distributed leader election protocol allows these N processes to elect a leader. Here is a probabilistic protocol due to Itai and Rodeh.

- The protocol is initiated in parallel by all processes.
- Each process begins by randomly choosing a number from $\{1, 2, ..., N\}$, and passing it on to its neighbour.
- If a process receives a number lower than one it has generated, the message is dropped.
- If a process receives a number higher than its own, the process drops out of the election and forwards the message.
- Finally, if a process receives a number that is the same as its own, the process forwards the message, noting that there is a number clash.

In the end, the leader is the process that chose the highest random number initially. If a number clash is recorded, all processes with the highest number choose a fresh number and start another round.

Write Rust implementations of the protocol for $N = 5$ and $N = 10$.

- Print informative diagnostic information when a process receives or sends a message.
- At the end of the protocol, each process should print a diagnostic message with the identity of the process that has been elected leader.
- **Note:** You may need to pass extra information with each message.
