Coaster: Concurrent Programming in Java

Concurrency 223
Department of Computing
Imperial College

Overview

A roller coaster at a fairground has two cars that carry passengers around the ride. One car can carry two passengers, the other three. The cars proceed independently around the track. Passengers wait for the coaster cars at a platform that can accommodate a maximum of nine passengers. When a coaster car arrives at the platform, it waits until there are enough passengers to fill it to capacity (i.e. two or three passengers). The car then sets off around the track, deposits its passengers and returns to the platform. Only one of the two cars can be waiting at the platform at any one time. The FSP model of this system is given below:

```
const Max  = 9
const MCar = 4

// models passenger arrival at the platform
PASSENGERS = (newPassenger -> PASSENGERS).

// limits passengers on platfrom to Max & allocates passengers to cars
CONTROLLER
    = CONTROL[0][0],
CONTROL[count:0..Max][carSize:0..MCar]
    = (when (count < Max)
            newPassenger   -> CONTROL[count+1][carSize]
      |requestPassenger[n:1..MCar] -> CONTROL[count][n]
      |when (carSize > 0 && count >= carSize)
            getPassenger[carSize] -> CONTROL[count-carSize][0]
      ).

// the coaster car requests N passengers and departs when the
// controller responds with getPassenger
COASTERCAR(N=MCar)
    = (arrive -> requestPassenger[N] -> getPassenger[i:1..MCar] ->
        if (i > N) then ERROR else (depart -> COASTERCAR))
      +{{requestPassenger,getPassenger}[1..MCar]}.

// controls access to the platform
PLATFORMACCESS = ({arrive,depart} -> PLATFORMACCESS).

// system with two coaster cars with capacity two and three
||ROLLERCOASTER
        = (PASSENGERS
           || car[1..2]::(CONTROLLER || PLATFORMACCESS )
           || car[1]:COASTERCAR(2)
           || car[2]:COASTERCAR(3)
           )
           /{newPassenger/car[1..2].newPassenger}.
```

What to do

Download the following files from CATE:

- `simple_coaster.lts`, a copy of the FSP model above.
- `coaster.zip`, a zip containing a partial implementation of the model as a concurrent Java program.

Unzip `coaster.zip` by using the command `unzip coaster.zip`. A subdirectory called `src` will appear containing the java files. Compile the files. Run the partial implementation by typing `java RollerCoaster`. You can run the lab version of the solution by typing the command `labcoaster`.

Part I

(a) The current model has a bug which means that more passengers can get into a coaster car than its set capacity. Find this bug by performing a safety check on the model and modify the model such that no safety violations occur. Copy the error trace produced by LTSA on the original model into a file `error_trace.txt` and save your modified model in a file `correct_coaster.lts`.

(b) You must complete the implementations of `Controller.java` and `PlatformAccess.java`. Note that the `requestPassenger` and `getPassenger` model actions are combined into a single method `getPassenger` in the `Controller` class. You will modify `Controller.java` again in Part II, so once you have finished this section copy your `Controller.java` to `ControllerPartI.java`. You will not need to modify `PlatformAccess.java` in Part II so there is no need to save it under a new name.

Part II

The roller coaster system has the disadvantage that passengers may wait a long time if there are not enough of them to fill a coaster car. To overcome this problem, a button is installed on the platform that when pressed by an operator releases the coaster car even if it is not full. The button should only have an effect if there is a car waiting at the platform, i.e. a button press should not be remembered after the car has left. In addition, a car should not leave the platform empty.

(a) Update the model to include the behaviour of this button by adding a process:

```
BUTTON = (goNow -> BUTTON).
```

and by modifying `CONTROLLER` and `ROLLERCOASTER`. Check that your modified model has the required behaviour. Save your modified model in a file `with_go_coaster.lts`.

(b) Update the implementation of the `Controller` class to reflect the additional behaviour. Check that it runs correctly.

Submission

- `error_trace.txt` produced in Part I (a).
- `correct_coaster.lts` produced in Part I (a).
- `ControllerPartI.java` produced in Part I (b).
- `PlatformAccess.java` produced in Part I (b).
- `with_go_coaster.lts` produced in Part II (a).
- `Controller.java` produced in Part II (b).

Assessment

The marks in this exercise are allocated as follows:

Part I (a)	15%
Part I (b)	35%
Part II (a)	25%
Part II (b)	25%

Optional Extension

For interest - extend the Java program with a third coaster car (do not submit it, but if you would like feedback you can discuss it with one of the lab demonstrators).