

Concurrent Programming: An Example

Madhavan Mukund, S P Suresh

Programming Language Concepts
January–April 2026

An exercise in concurrent programming

- A narrow North-South bridge can accommodate traffic only in one direction at a time.
- When a car arrives at the bridge
 - Cars on the bridge going in the same direction \Rightarrow can cross
 - No other car on the bridge \Rightarrow can cross (implicitly sets direction)
 - Cars on the bridge going in the opposite direction \Rightarrow wait for the bridge to be empty
- Cars waiting to cross from one side may enter bridge in any order after direction switches in their favour.
- When bridge becomes empty and cars are waiting, yet another car can enter in the opposite direction and makes them all wait some more.

An example ...

- Design a class `Bridge` to implement consistent one-way access for cars on the highway
 - Should permit multiple cars to be on the bridge at one time (all going in the same direction!)
- `Bridge` has a public method `public void cross(int id, boolean d, int s)`
 - `id` is identity of car
 - `d` indicates direction
 - `true` is `North`
 - `false` is `South`
 - `s` indicates time taken to cross (milliseconds)

An example ...

```
public void cross(int id, boolean d, int s)
```

- Method `cross` prints out diagnostics
 - A car is stuck waiting for the direction to change
`Car 10 going South stuck at Fri Feb 25 12:42:13 IST 2022`
 - The direction changes
`Car 10 switches bridge direction to South at Fri Feb 25 12:42:13 IST 2022`
 - A car enters the bridge
`Car 10 going South enters bridge at Fri Feb 25 12:42:13 IST 2022`
 - A car leaves the bridge
`Car 10 leaves at Fri Feb 25 12:42:14 IST 2022`

- The “data” that is shared is the `Bridge`
- State of the bridge is represented by two quantities
 - Number of cars on bridge — `int bcount`
 - Current direction of bridge — `boolean direction`
- The method `public void cross(int id, boolean d, int s)` changes the state of the bridge
 - Concurrent execution of `cross` can cause problems ...
- ... but making `cross` a synchronized method is too restrictive
 - Only one car on the bridge at a time
 - Problem description explicitly disallows such a solution

- Break up `cross` into a sequence of actions
 - `enter` — get on the bridge
 - `travel` — drive across the bridge
 - `leave` — get off the bridge
 - `enter` and `leave` can print out the diagnostics required
- Which of these affect the state of the bridge?
 - `enter` : increment number of cars, perhaps change direction
 - `leave` : decrement number of cars
- Make `enter` and `leave` synchronized
- `travel` is just a means to let time elapse — use `sleep`

Code for `cross`

```
public void cross(int id, boolean d, int s){  
  
    // Get onto the bridge (if you can!)  
    enter(id,d);  
  
    // Takes time to cross the bridge  
    try{  
        Thread.sleep(s);  
    }  
    catch(InterruptedException e){}  
  
    // Get off the bridge  
    leave(id);  
}
```

Entering the bridge

- If the direction of this car matches the direction of the bridge, it can enter
- If the direction does not match but the number of cars is zero, it can reset the direction and enter
- Otherwise, `wait()` for the state of the bridge to change
- In each case, print a diagnostic message

Code for enter

```
private synchronized void enter(int id, boolean d){
    Date date;

    // While there are cars going in the wrong direction
    while (d != direction && bcount > 0){

        date = new Date();
        System.out.println("Car "+id+" going "+direction_name(d)+" stuck at "+date);

        // Wait for our turn
        try{
            wait();
        }
        catch (InterruptedException e){}
    }

    ...
}
```

Code for enter

```
private synchronized void enter(int id, boolean d){
    ...
    while (d != direction && bcount > 0){ ... wait() ...}
    ...

    if (d != direction){ // Switch direction, if needed
        direction = d;
        date = new Date();
        System.out.println("Car "+id+" switches bridge direction
            to "+direction_name(direction)+" at "+date);
    }

    bcount++; // Register our presence on the bridge

    date = new Date();
    System.out.println("Car "+id+" going "+direction_name(d)+" enters bridge at "+date);
}
```

Code for leave

Leaving the bridge is much simpler

- Decrement the car count
- `notify()` waiting cars ... provided car count is zero

```
private synchronized void leave(int id){
    Date date = new Date();
    System.out.println("Car "+id+" leaves at "+date);

    // "Check out"
    bcount--;

    // If everyone on the bridge has checked out, notify the
    // cars waiting on the opposite side
    if (bcount == 0){
        notifyAll();
    }
}
```

Summary

- Concurrent programming can be tricky
- Need to synchronize access to shared resources
- ... while allowing concurrency
- This bridge crossing example is a prototype for a number of real world requirements