PRA2003 Programming Skills

Week 5: Threads and Concurrency

Cameron Browne & Andreea Grigoriu

4 December 2018

This week you will learn about threads in Java through some simple examples. Then in the final assignment you will make some final improvements to your Emerald Mine game, including adding a concurrent simulator and two new monsters.

Task 5.1. Sleepy Threads

The `Sleepy.java` program (available in the Java Files section on EleUM) takes an array of non-negative integers, then for each integer creates an anonymous Thread object that immediately starts then goes to sleep for that number of seconds.

Try to work what this code is actually doing. What is the ultimate result of this program? Insert some non-negative numbers into the `nums` array in main() then run the program to see if you are right.

What is an easy way to speed this program up? And what is the danger in doing so?

Task 5.2. Concurrency issues

Compile and run the `Counter.java` program. This example shows another way of creating worker threads, by simply creating `Thread` objects and overriding their `run()` method (as opposed to *implementing* it for the `Runnable` interface).

In this code, `threadA` is increasing the value of a shared variable by one. At the same time `threadB` is decreasing the value of the same shared variable by one. Both threads are executing these operations 1,000 times. So ideally, the final result should be 0 since `threadA` increases the value 1,000 times and `threadB` decreases the value 1,000 times. Right? Run this program several times. What do you observe?

Explain what is causing this odd behaviour. *Hint:* it is due to a race condition that interrupts the normal flow of execution in each thread. This kind of problem can happen if more than one thread attempts to access a shared resource concurrently.

Correct this problem by using `synchronized`. You can either use `synchronized` for the methods themselves or for the block of code that access the common resources. Is either of these solutions better than the other in this case?

Task 5.3. Lottery Threads

Imagine that you would like a holiday but are not sure which destination to pick out of Fiji, Jamaica and Bora-Bora. Write a new Java program called `LotteryThreads` using the following code as a basis.

```
class LotteryThread extends Thread
{
    public LotteryThread(final String str)
    {
        super(str);
    }

    @Override
    public void run()
    {
        // Put your code here.
        // If you use sleep, you need to catch
        // the associated Exception.
    }
}

class LotteryThreads
{
    public static void main (final String[] args)
    {
        // Create the threads and start them here
    }
}
```

Your run() method should work as follows. Execute a loop 10 times, and for each iteration print the iteration number (1, 2, 3, …, 10) and the name of the process using `this.getName()`. Then, put the thread to sleep for a random number of seconds (choose a not small but also not very big number). Once the loop ends, the process should print its name and that it's finished.

In the main() method, create three new threads based on the `LotteryThread` constructor (with the name of each potential destinations) and start them. Observe the results. The thread with the destination that finished last is the winner! Have a nice vacation! If you are not happy with the result, just run the program again.

Assignment 6 ** Due 23:59 on Monday 10 December 2018 **

In order to make the game more interesting, add the following functionalities / modifications (skip these steps if you have already implemented them):
* Gravity is applied all the way, once possible (i.e. if a rock has no dirt below, then it drops till the last free tile).
* Similarly, if a player goes below a falling rock, then the rock crushes the player.
* Change the code so that going out of bounds is no longer possible, for the player and for the monsters. If a move is taken that would push the player or a monster off the board, then the move fails.

Concurrent Simulator

Using the `Simulator.java` as a starting point, create a class that simulates the world while *waiting* for the player to move. The simulator is run as a separate *thread*. There are several steps to make the simulator possible to be integrated:

1. Change `World.applyGravity()` to make it public (so it can be invoked from the simulator).

2. Change `KeyEventTrapper.keyPressed()` to print the number of remaining emeralds (to the console), and invokes `System.out.exit(-1)` to terminate the JVM when the player wins or loses, also informing the player of their win/loss on the console. Skip this step if you have already implemented it.

3. Change `World.applyMove()`: Remove the code that moves the alien and the application of gravity (the simulator will invoke the appropriate methods instead). Then, make a more general method `boolean moveMonsters()` that moves all the monsters in the world. To do this, loop over all the objects in the world, and for each one that is moveable and *not* a player, get its move and change the World accordingly. Monsters should never move into other monsters. Also, monsters should not be allowed to move through Dirt: *This is a special case and differs from previous assignments.* `moveMonsters()` returns true if one of the monsters moves to the space of the player, false otherwise. ***Note:*** Until other monsters are added, this method will move the single alien.

4. **important:** Ensure that all the code that modifies the world from different threads is enclosed within keyword synchronized(this) block, as seen below! This locks the World object to prevent two threads from modifying the world at the same time.

```
public boolean applyGravity(int exceptRow, int exceptCol)
{
  boolean fellOnPlayer = false; // flag to indicate if an
object with // mass fell on the player
  synchronized(this) {
    // ... same code as before ...
}   return fellOnPlayer;

}
public boolean moveMonsters() {
  boolean deathByMonster = false;
  synchronized(this) {
    // add code here
}

  return deathByMonster;
}
public int applyMove(char move) {
  synchronized(this) {
    // ... same code as before ...
}

return 0; }
```

5. Change the signature of `readrawWorld()` in `EmeraldMinePanel` to:

```
public synchronized void redrawWorld(World world) {…}
```

Add a method, `public void redraw()` to GUI that simply invokes `EmeraldMinePanel.redrawWorld(world)`. This method is used by the simulator.

6. Change the main method in `Main.java` so that it creates a World, creates and initialises the GUI, then creates the simulator and starts it as a new thread:

```
Thread sim = new Simulator(world);
sim.start(); // starts a new concurrent thread
```

7. Test the concurrent simulator using the example world with a single alien before moving on to the two new monsters. Add a second alien to the example world and test the code to make sure it works with two aliens.

Bugs and Spaceships

Using the code in `Bug.java` as a starting point, create two new monster classes: `Bug` and `Spaceship`. The character for a bug is 'b' and the character for a spaceship is 's'. You can put them in your `WorldObject` class if you want a more neat code.

Unlike the alien, these monsters have a *state*: their current direction. Each monster always tries to move in its current direction. Therefore the `getMove()` methods simply return the corresponding character depending on the monster's current direction.

Add a method `public void changeDirection()` to the `WorldObject` base class with an empty default implementation.

Modify `World.moveMonsters()` in the following way: whenever a monster tries to move but is unsuccessful (due to being blocked by an invulnerable object or another monster), then it changes direction. Bugs change their direction 90 degrees clockwise order, so that up becomes right, right becomes down, down becomes left, and left becomes up. Similarly, spaceships change direction in counter-clockwise order. Add code to `Bug.changeDirection()` and `Spaceship.changeDirection()` to implement these change in directions. Aliens do not have any direction, so the change in direction does not affect them.

New images for each direction are available via the portal. Add the necessary code for these new monsters to be displayed properly.

Finally, add code to the constructor to create the bug and spaceship whenever the corresponding character is seen in the input file. Load the world `example2.txt` and use it to test the new monsters.

Honour code, coding style and deliverable

• Please refer to Infosheet → Paragraph 7 → Honor policy.

• You are welcome to expand your program to do extra things but they are not mandatory. Do not deviate from the specific requirements of the assignment!

• If you use some code found online (or anywhere else, e.g. stackoverflow), clearly state it (with comment about where you found it) in your .java files. That means mention the author and the www source you found it. Be sure that you read the terms of the licence for the code that you are reusing. Most code that is found openly in the internet has specific licences that you should not violate.

• Please refer to the coding style guidelines uploaded to courses portal (under Course materials) on how to write readable programs.

Deadline for submission: 23:59 on Monday 10 Dec. 2018