OS Final Lab 472

File HajjGroup.java

```
public class HajjGroup {
  private int availableSlots;

  public HajjGroup(int slots){
    availableSlots = slots;
  }

  public void register(){
    int updated = availableSlots - 1;
    try { Thread.sleep(50); } catch (Exception e){}
    System.out.println("Pilgrim registered.  Slots = " + updated);
    availableSlots = updated;
  }

  public void cancel(){
    int updated = availableSlots + 1;
    try { Thread.sleep(50); } catch (Exception e){}
    System.out.println("Registration cancelled. Slots = " + updated);
    availableSlots = updated;
  }

  public int getSlots(){
    return availableSlots;
  }
}
```

File RegistrationTask.java

```
public class RegistrationTask implements Runnable {
  private HajjGroup group;
  private boolean isRegistering;  // true = register, false = cancel

  public RegistrationTask(HajjGroup g, boolean reg){
    group = g;
    isRegistering = reg;
  }

  public void run(){
    // ---- Part (a): complete the run method ----
    // If isRegistering is true -> call group.register()
    // If isRegistering is false -> call group.cancel()
 }}
```

File TestHajj.java

```
public class TestHajj {
  public static void main(String[] args){
    HajjGroup group = new HajjGroup(50);

    for (int i = 1; i <= 10; i++){
      RegistrationTask reg = new RegistrationTask(group, true);
      RegistrationTask cnl = new RegistrationTask(group, false);

      // ---- Part (b): create TWO Thread objects from reg and cnl,
      //        then START both threads ----

    }
    try { Thread.sleep(3000); } catch (Exception e){}
    System.out.println("FINAL slots = " + group.getSlots());
  }
}
```

• Part A (2 Points): Open RegistrationTask.java and complete the run() method.

• Part B (2 Points): Open TestHajj.java. Inside the for loop, create two Thread objects (one for registering, one for canceling) and start both of them.

• Part C (2 Points): Run the program. You will notice that the final number of slots is sometimes wrong, but the program finishes and does not freeze. Tell me: Is this code suffering from a Race Condition or a Deadlock? Please write 1-2 lines to justify your answer.

☐ Race Condition

☐ Deadlock

• Part D (4 Points): Modify the Java code to solve the problem you identified in Part C. Please clearly write down which file and which line(s) you changed.

Answers

File HajjGroup.java

```
public class HajjGroup {
  private int availableSlots;

  public HajjGroup(int slots){
    availableSlots = slots;
  }

  public void register(){
    int updated = availableSlots - 1;
    try { Thread.sleep(50); } catch (Exception e){}
    System.out.println("Pilgrim registered.  Slots = " + updated);
    availableSlots = updated;
  }

  public void cancel(){
    int updated = availableSlots + 1;
    try { Thread.sleep(50); } catch (Exception e){}
    System.out.println("Registration cancelled. Slots = " + updated);
    availableSlots = updated;
  }

  public int getSlots(){
    return availableSlots;
  }
}
```

File RegistrationTask.java

```
public class RegistrationTask implements Runnable {
  private HajjGroup group;
  private boolean isRegistering;  // true = register, false = cancel

  public RegistrationTask(HajjGroup g, boolean reg){
    group = g;
    isRegistering = reg;
  }

  public void run(){
    // ---- Part (a): complete the run method ----
    // If isRegistering is true -> call group.register()
    // If isRegistering is false -> call group.cancel()
 }}
```

File TestHajj.java

```
public class TestHajj {
  public static void main(String[] args){
    HajjGroup group = new HajjGroup(50);

    for (int i = 1; i <= 10; i++){
      RegistrationTask reg = new RegistrationTask(group, true);
      RegistrationTask cnl = new RegistrationTask(group, false);

      // ---- Part (b): create TWO Thread objects from reg and cnl,
      //        then START both threads ----

    }
    try { Thread.sleep(3000); } catch (Exception e){}
    System.out.println("FINAL slots = " + group.getSlots());
  }
}
```

• Part A (2 Points): Open RegistrationTask.java and complete the run() method.

```
try{   if(isRegistering == true){
            group.register();
       }else{
            group.cancel();
       }} catch(Exception e){}    }
```

• Part B (2 Points): Open TestHajj.java. Inside the for loop, create two Thread objects (one for registering, one for canceling) and start both of them.

```
Thread reg1 = new Thread(reg);
 Thread can1 = new Thread (cnl);
reg1.start();
can1.start()
```

• Part C (2 Points): Run the program. You will notice that the final number of slots is sometimes wrong, but the program finishes and does not freeze. Tell me: Is this code suffering from a Race Condition or a Deadlock? Please write 1-2 lines to justify your answer.

☑ Race Condition

☐ Deadlock

This is a Race Condition, because multiple threads are accessing and modifying the shared slots variable at the same time without synchronization, causing incorrect final values.

• Part D (4 Points): Modify the Java code to solve the problem you identified in Part C. Please clearly write down which file and which line(s) you changed.

```
public synchronized void register() { ..... }
public synchronized void cancel() { ..... }
```