CS180 Spring 2010 Exam 2 Solutions, 29 March, 2010

Prof. Chris Clifton

Turn Off Your Cell Phone. Use of any electronic device during the test is prohibited.

Time will be tight. If you spend more than the recommended time on any question, **go on to the next one**. If you can't answer it in the recommended time, you are either going in to too much detail or the question is material you don't know well. You can skip one or two parts and still demonstrate what I believe to be an A-level understanding of the material.

*Multiple choice solutions marked in **bold face**. Scoring descriptions given in the short answer section are just a rough guide.*

Multiple Choice

Provide answers on the Scantron card. (You may mark them on this sheet as well, but the mark on the Scantron card is the official version.)

1 Exceptions and Inheritance (5 minutes, 2 points)

Assume we have defined the following exceptions:

```
public class AException extends Exception { }
public class BException extends AException { }
public class CException extends BException { }
```

and we use them in the following program:

```
public static void main(String[] args)
{
  try {
    FuncA(true);
    System.out.print("1");
  }
  catch(AException ex) {
    System.out.print("2");
  }

  System.out.print("3");
}
```

```
static void FuncA(boolean x) throws AException
{
  try {
    if ( x ) throw new CException();
    System.out.print("4");
  } catch(BException ex) {
    System.out.print("5");
    throw new AException();
  } catch(AException ex) {
    System.out.print("6");
  }

  System.out.print("7");
}
```

What should the output of the program be?

A 413

B **523**

C 5673

D 54213

E 564713

Remember that after an exception is thrown, the first matching catch is executed. Once that block is done, processing continues outside the try/catch context - it never returns to the try block.

2 Constructing Windows (3 minutes, 2 points)

When constructing a window using a JFrame, you add objects to a JPanel. Which two of the following are true statements (you get one point for each true statement you mark, but lose a point for each false statement you mark):

A Each object added must be a different class.

B **Multiple components in a window can share the same instance implementing ActionListener.**

C **You must create an instance of a component before adding it to the JPanel.**

D Each object added must have its own event handler (ActionListener).

E There can only be one instance of an object for each class that implements the ActionListener interface.

3 Threads (2 minutes, 1 point)

Which of the following statements about programming with threads is true?

A After a call to start() a thread, the next statement must be a call to join() that thread or the thread will not execute.

B Threads can only be used on multiprocessor machines.

C **Different threads can call methods of the same object at the same time.**

D Once a program starts a thread, it cannot do anything else until that thread has finished running.

4 Synchronization (4 minutes, 2 points)

Given the following code:

```
public class Vertical {

  private int alt;

  public synchronized void up() {
    ++alt;
  }

  public void down() {
    --alt;
  }

  public synchronized void jump() {
    up();
    down();
  }
}
```

Which two of the following are correct (you get one point for each one you get right, but you lose a point for each incorrect one you mark):

A The code will fail to compile.

B Separate threads can execute the up() method concurrently on the same object.

C **Separate threads can execute the down() method concurrently on the same object.**

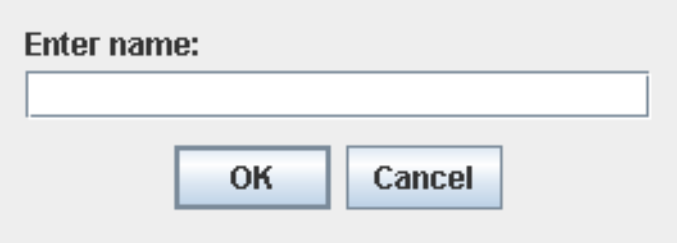
D **One thread can execute up() on an object concurrently with another executing down() on the same object.**

E Two threads can both execute up() at the same time on the same object, both execute down() at the same time on the same object, and one can execute up() while the other executes down() on the same object.

Registration System

Questions 5 through 9 concern a hypothetical "registration kiosk": A computer with multiple screens and keyboards. Each screen starts up with a "Register for CS18000" window:

When a student walks up and clicks "Register for CS18000", the program brings up an "Enter name" window:



When the student clicks "OK", the registration is complete. This means that an instance of class Student has been created, and the number of students registered (the class variable totalStudents in class Student) has been updated.

If a student walks up to the kiosk and clicks "Register for CS18000", but the class is full (totalStudents $\geq$ MAXSTUDENTS), the student instead sees a "Class full" message.

Code fragments covering the following questions are given at the end of the test (Appendix A). However, you should read the questions first. The questions concern basic design principles of graphical user interfaces and concurrent systems; you may be able to answer them without looking at the code at the end of the test. It is best if you read the question and answer choices first, then if you are unsure, look at the code.

5 Event handling (5 minutes, 1 point)

When we run the program, the "Register for CS18000' window pops up on each screen (we see the button shown above), but when users click on "Register for CS18000" nothing happens. Which of the following is a likely fix for the problem?

A Create an instance of Student for each kiosk screen.

B Call the RegisterAction.actionPerformed() method.

C **Add the RegisterAction listener to the "Register for CS18000" button.**

D Add a try/catch block with "join" statements at the end of the main method, as the program now ends as soon as the screens are set up.

E Add the "Register for CS18000" button to a JPanel.

6 Concurrency (3 minutes, 1 point)

Assume we have fixed the preceding problem. We now get all the screens set up properly, and if one person walks up and registers, it works. But if a second person clicks "Register for CS18000" while the first person is registering, nothing happens until the first student clicks "OK". This is happening because:

A **ActionListeners are not concurrent; the next action is not performed until the previous action has completed.**

B The class variable totalStudents is shared by all Student instances, so once one starts, no other can run.

C Even though we have multiple screens, there is only one processor, so it isn't possible for the program to handle inputs from two users.

D The JFrames have to be used in the same order they were created in the main method.

E The try-catch block in actionPerformed only allows one thread to execute at one time.

7 Threads (4 minutes, 2 points)

Realizing we need concurrency, we make class Student extend class Thread. This requires we implement a run method; since the other methods already do everything we need, we simply make it:

```
public void run() { return; }
```

We then rewrite the event handler for the "Register for CS18000" button (method actionPerformed()) to include:

```
Student s = new Student(screen);
s.start();
try { s.join(); }
catch ( InterruptedException ie ) { }
// Save s into appropriate variables
```

Running this gives the same behavior as before - nothing seems to have changed. In hindsight, we realize this couldn't have worked because:

A It won't compile, we haven't defined a start() and join() method for Student.

B **The only thing executed concurrently is the run() method; the hangup is in the constructor for Student.**

C When another user clicks a button, it causes an InterruptedException; we need to create a new Student in the catch block.

D The only place we can start a new thread is in the main method.

8 Managing Concurrency (5 minutes, 2 points)

We manage to fix all of the above problems - we now have multiple threads running so that several people can use the kiosk at once, and all get a quick response. Unfortunately, we seem to be ending up with too many students in class (i.e., we end up with totalStudents > MAXSTUDENTS.) This is because:

A After we throw the ClassFull exception, we continue to register the student. We need to put a return after the throw.

B Each thread has its own copy of totalStudents, so each can have MAXSTUDENTS.

C The value of totalStudents is initialized to 0 every time we create a new instance of Student, so totalStudents is always 0 when it is checked.

D **We have a race condition: The check if there is room happens before we update the number of students, and this can happen concurrently.**

Short Answer

Write your answers in the space provided. The space provided should be sufficient, but if you need more, use the back of the sheet.

9 Synchronization (12 minutes, 5 points)

If you are unsure about your answer to Question 8, you should make sure you have done everything else before you do this one. We fix the "too many students" problem of the preceding question with a simple (one word) change to class Student. However, in doing this fix we lose concurrency: only one person is able to register at a time.

Part 1 (1 point): What did we do to fix the problem? *Make Student a synchronized method.* Technically this isn't possible in Java 1.5, as Constructors can't be synchronized - there were other acceptable answers, as long as you got across the point that the entire method was synchronized.

Part 2 (4 points): Rewrite the Student method to fix this. (It is okay if you don't rewrite everything, as long as it is quite clear what your solution does.)

One solution:

```
synchronized(totalStudents) {
  if ( totalStudents >= MAXSTUDENTS ) throw new ClassFull();
  else totalStudents++; // Reserve a place for us before unsynchronizing.
}
  name = JOptionPane.showInputDialog(
//... as before.
  if ( name == null ) { // User pressed cancel button
    synchronized(totalStudents) {
      totalStudents--;  // Free up the space we reserved.
    }
    throw new StudentCancelled();
  }
```

Scoring: 1 point for synchronizing check on totalStudents, 1 for synchronize on a class variable (totalStudents or other), 1 point for moving check and update together, 1 point for making sure that totalStudents isn't

increased if the student cancels / class is full. Alternate credit for pointing out that this wouldn't solve the problem, since the handler still isn't concurrent: 2 for pointing out the problem, 1 for a reasonable start at a fix, 1 for noting need for synchronization, up to a total of five points.

10 Exception processing (4 minutes, 2 points)

The following code will fail to compile. You can assume that all methods and variables in this fragment are properly defined; the error is entirely in the code block below.

```
int i;
try {
  i = divide(a, b); // May throw a DivideByZeroException
} catch (DivideByZeroException dbz) {
  System.err.println("Divide by zero attempted.");
}
System.out.println(i);
```

Explain briefly why it fails (1 point) and a way to fix it (1 point).

If divide throws an exception, then the System.out.println(i) would have i uninitialized. The compiler will not allow this to happen. The solution is to move the System.out.println(i) into the try block.

Initializing i is not a good solution. While it fixes the compile error, it means that dividing a number by 0 gives a valid output - a logical error in the program.

11 Concurrent Programming (5 minutes, 3 points)

If the following code segment is an example of proper concurrent programming, explain why (what will happen when it is run). Otherwise, explain any mistakes you find and describe how to properly fix them. Hint: You don't need to know details of all the classes to figure this out. You should assume MyThreadedClass extends Thread.

```
import java.util.*;
public class ConcurrentCheck {
  private static ArrayList<Thread> threads = new ArrayList<Thread>(5);

  public static void main( String args[] ) {
    for( int i = 0; i < threads.size(); ++i )
      threads.add(new MyThreadedClass());
    for( int i = 0; i < threads.size(); ++i ) {
      threads.get(i).start();
      try {
        threads.get(i).join();
      } catch (InterruptedException e) { /* Ignore */ }
    }
    System.out.println("Done");
  }
}
```

The error here is that each thread is joined immediately after it has been started. Thus, the calling method (main) must wait until the first thread finishes before it can start the second thread. This is sequential operation, rather than concurrent. The proper way to do this is to start all threads first, and then have a second loop to join to all of them.

Scoring: 1 point for noting that it is sequential (2 for a really good discussion), 1 for getting start right, 1 for getting join right, to a max of 3.

A Code for Questions 5 through 9

Please tear these two pages off for easy reference; you don't need to turn them in (and removing them will make our grading job easier.)

```
import javax.swing.*;
class Student {
  private static final int MAXSTUDENTS = 175;
  public static int totalStudents = 0; // Total students who have registered.

  private String name; // Name of currently registering student.
  public Student(JFrame screenToUse) throws ClassFull, StudentCancelled {
    if ( totalStudents >= MAXSTUDENTS ) throw new ClassFull();

    name = JOptionPane.showInputDialog(
            screenToUse, // Show on the given kiosk screen
            "Enter name:", // Message in dialog
            "Registering for CS180", // Window titlebar
            JOptionPane.PLAIN_MESSAGE);

    if ( name == null ) // User pressed cancel button
      throw new StudentCanceled();

    totalStudents = totalStudents + 1;
  }
}

import javax.swing.*;
import java.awt.event.*;
class RegKiosk {
  private static final int SCREENS = 4;
  // Variables declared to save registered students

  private class RegisterAction implements ActionListener {
    JFrame screen;
    public RegisterAction ( JFrame thisScreen ) {
      screen = thisScreen;
    }
    public void actionPerformed( ActionEvent e ) {
      try {
        Student s = new Student(screen);
        // Save s into appropriate variables
      } catch ( StudentCanceled sc ) { /* No need to do anything */ }
        catch ( ClassFull cf ) {
          JOptionPane.showMessageDialog(
              screen, // Show on appropriate frame
              "CS18000 is full", // Message
              "Class full", // Title bar
              JOptionPane.ERROR_MESSAGE);
      }
    }
  }
```

```
  RegKiosk(int screen) {
    GraphicsConfiguration gc = // Create reference to screen i (trust that it works).
    JFrame window = new JFrame("CS18000 Registration Kiosk screen", gc);
    JPanel panel = new JPanel();
    window.add(panel);
    JButton button = new JButton("Register for CS18000");
    panel.add(button);
    window.setVisible(true);
  }
  // Various other methods defined and implemented.

  public static void main( String args[] ) {
    RegKiosk RegScreens[]
    for ( int i = 0; i < SCREENS; i++ )
      RegScreens[i] = new RegKiosk(i)

    }
  }
}
```