University of Glasgow | School of Computing Science

XXXday, XXth XXX 2014
2:30 pm- 4.30 pm
(Duration: 2 hours)

DEGREE OF MSc in Information Technology

ADVANCED PROGRAMMING (M)

(Answer all 4 questions.)

This examination paper is worth a total of 75 marks

You must not leave the examination room within the first hour or the last half-hour of the examination.

INSTRUCTIONS TO INVIGILATORS

Please collect all exam question papers and return to School together with exam answer scripts

1.

(a) Explain what is meant by *type checking*. What is the difference between *static* and *dynamic* type checking? Give an example of how each is used in Java.

[5 marks]

(b) Explain the meaning of the terms *method overriding* and *dynamic dispatch*.

[4 marks]

(c) Compare and contrast *abstract classes* and *interfaces* in Java.

[6 marks]

The questions are book work, main points noted.

(a) Type checking means checking that operations are only applied to data of an appropriate kind: for example, not adding two values together unless they are numbers (or strings) [1 mark]. Static type checking means that the compiler or development environment classifies data according to its type, perhaps assisted by type declarations in the program, and checks that operations are being used correctly, before the program is run [1 mark]. Dynamic type checking means that the runtime system checks that every operation is being applied to appropriate values, before doing the corresponding computation [1 mark]. Example of static type checking in Java: every object variable has a declared class, and when a method is called on a variable, the compiler checks that the method is defined in the class declared for the variable [1 mark]. Example of dynamic type checking in Java: if the program contains a cast then at runtime, the type of the object being cast is compared with the type that it is being cast to [1 mark].

(b) If class A contains method m, and class B extends class A, then B can give a new definition of m. This is overriding [2 marks]. Dynamic dispatch: if a variable x has declared type A, then at runtime the object stored in x might be an instance of A or an instance of B. In the call x.m(), the definition of m that is used is determined by the runtime type of the object stored in x [2 marks].

(c) Abstract classes and interfaces are two mechanisms for specifying what a class must define, without actually giving definitions [1 mark]. If A is an abstract class, then it may contain abstract methods, which have no definition; a class that extends A must give definitions for those methods [1 mark]. Similarly, an interface may contain method declarations without definitions, and a class that implements the interface must define those methods [1 mark]. There can be no instance of an abstract class or an interface [1 marks]. Both abstract classes and interfaces support polymorphism in the sense that they enable a range of concrete classes to be used in contexts where an abstract class or interface has been specified [1 mark]. They have different uses, for example because a class can implement several interfaces but can only extend one abstract class [1 mark].

2. **(a)** Explain the terms *stack* and *heap*, and how they are used, in relation to memory management in Java. Your answer should include an explanation of what happens in a program with multiple threads.

[6]

(b) Give an overview, using diagrams and explanations as necessary, of how memory is allocated and de-allocated during the execution of the following program, and of how the threads involved are sharing state.

```
public class Q2 implements Runnable{
   public int counter = 0;

   public void run(){
      for( int i=0; i<1000000; i++ ){
         counter = counter + 1;
      }
   }

   public static int add(int x) {
      return x+1;
   }

   public static void main( String[] args ){
      Q2 q2 = new Q2();
      Thread t = new Thread( q2 );
      t.start();
      t.join();
      System.out.println(add(q2.counter));
   }
}
```

[8]

(a) [Bookwork] The heap is a single repository for the whole program consisting of a set of memory chunks, each one holding the data for a single object, and using internal memory pointers to represent references between objects. The stack is set of primitive data values and object references held in a last-in, first-out structure. The stack grows and shrinks according to the predictable execution of methods and blocks, as parameters, local variables, and expressions are created and evaluated. The lifetimes of such values matches with the block structure of the program. Each method call creates a new stack frame which is used to store the parameter values and local variables for the method. All objects are allocated in the heap, and the stack contains references to them. Threads share the single heap but each have a separate stack.

(b) [Application] There should be a single heap and two stacks. The heap should contain an object of class Q2 (this is q2 created in the main method) and an object of class Thread. The object q2 should contain an instance variable "counter". The two stacks are for the two threads that are created by the code: one thread is running the main method and the other thread is created by the main method. The two stacks should each have a pointer at the base of the stack into the Q2 object in the heap. This set up gains 5 marks, since understanding this was the main learning outcome of this section.

As the loop in Q2.run executes, the stack of that thread is used to do the necessary calculations with the variables i and counter. When the main method calls the add method, a new stack frame is created in its stack; this stack frame contains the variable x. In the add method, the stack is used to calculate x+1. 3 marks for this explanation.

3 **(a)** Java programs that use the Swing graphical user interface (GUI) library make use of the *Event Dispatch Thread*. Explain what this thread does. Why do Swing programs often create other threads? Give a typical example.

[4]

(b) Explain what is meant by the statement "Swing is not thread safe". How should Java Swing applications that use multiple threads be structured, because of the fact that Swing is not thread safe?

[4]

(c) Describe one mechanism provided by Swing in order to help achieve the structure that you have explained in part (b).

[3]

(d) Examine the following Swing-based application, which creates a window containing a button and a label. The following actions are performed when the button is pressed:

- a message is placed on the label;
- a long-running task is carried out;
- a second message is placed on the label.

Explain how it meets, or fails to meet, the structure you have set out in (b) above.

If it does not meet the structure you have described, rewrite the code so that it does. You may refer to the line numbers below, rather than re-writing complete lines of code in your new solution.

[8]

(e) Explain the purpose of lines 8 to 15.

[3]

```
1     public class Q3 extends JFrame{
2        JLabel myLabel;

3        Q3() {
4           GridLayout g1 = new GridLayout( 2, 1 );
5           this.getContentPane().setLayout( g1 );

6           JButton myButton = new JButton( "action" );
7           myButton.addActionListener(
8              new ActionListener(){
9                 public void actionPerformed( ActionEvent e ){
10                   myLabel.setText( "started" );
12                   // Assume some code here to perform a long-running task
13                   myLabel.setText( "finished" );
14                }
15             });
16          this.getContentPane().add( myButton );

17          myLabel = new JLabel();
18          this.getContentPane().add( myLabel );
19       }

20       public static void main( String[] args ) {
21          Q3 q3 = new Q3();
22          q3.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
23          q3.setBounds( 0, 0, 250, 100 );
24          q3.setVisible( true );
25       }
26    }
```

(a) [Bookwork] The Event Dispatch Thread runs the code that monitors GUI actions [1 mark]: mouse clicks in particular regions of the GUI, and key presses. In response to each action, it calls application code that has been defined in the corresponding event handler [1 mark]. The typical reason for introducing other threads is because long-running computations on the Event Dispatch Thread will make the GUI unresponsive [1 mark]. An example would be loading images in a web browser [1 mark].

(b) If a class is thread safe then its methods can be called from multiple threads without problems due to conflicting access of shared state [1 mark]. If a class is not thread safe then calling its methods from multiple threads may have unpredictable effects [1 mark]. Swing is not thread safe, so methods on Swing classes should not be called from multiple threads; they should only be called on the Event Dispatch Thread [1 mark]. In particular, other threads should not update the components of the GUI [1 mark].

(c) A thread can use the invokeLater method to execute code on the Event Dispatch Thread. The code is wrapped in a Runnable object and passed as a parameter to invokeLater. The Runnable is placed on a queue of tasks that is accessible to the Event Dispatch Thread – which will in due course, execute the task. [3 marks]

OR:

The SwingWorker class allows the programmer to define code to be executed in another thread, and also define a done() method which is called on the Event Dispatch Thread thread when the subsidiary thread finishes. This can be used to return results from long-running computations in other threads. [3 marks]

(d) It does not meet the structure as outlined. The GUI is set up in the thread that is running the main method, not the EDT. The updates to the text area are carried out on the EDT as they should be, since the actionPerformed method of the ActionListener attached to the button is called on the EDT. The problem here is that the long running task is also being executed on the EDT, causing the GUI to become unresponsive.

Students won't need to write all the code below. Lines 1-9 are a direct repeat from the code in the question, as are 14-17 and 19-22. There's only about 13-14 lines of new code to write, some of which only have curly braces on them!

```
public class Q3correct extends JFrame{
    JLabel myLabel;

    Q3correct(){
      GridLayout gl =new GridLayout(2,1);
      this.getContentPane().setLayout(gl);

      JButton myButton = new JButton("action");
      myButton.addActionListener( new ActionListener(){
          public void actionPerformed( ActionEvent e ){
            myLabel.setText( "started");
            // This section must be run in a separate thread
            new Thread(
                new Runnable(){
                  public void run(){
                      // Code omitted to perform some long-running task
                      // Now need to ensure the final GUI update is on the GUI thread
                      SwingUtilities.invokeLater( new Runnable() {
                        public void run(){
                         myLabel.setText( "finished");
                        }
                      });
                  }
                }).start();
          }
      });
      this.getContentPane().add(myButton);

      myLabel = new JLabel();
      this.getContentPane().add( myLabel );

    }

    private static void createAndShowGUI() {
      exQ3correct q3 = new exQ3correct();
      q3.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
      q3.setBounds(0,0,250, 100);
      q3.setVisible(true);
    }

    public static void main(String[] args) {
      // The set up code for the GUI should be run on the GUI thread
      SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }

}
```

(e) Lines 8-15 define an anonymous inner (or local) class. A new class is constructed, which implements the ActionListener interface by defining the actionPerformed method as shown. This class does not have a name. An object of this class is created, and is passed as the parameter to the addActionListener method. [3 marks]

4. Assume that the Java class **`ExamDatabase`** represents a database of exam papers. The class provides methods **`boolean paperSet(String course)`** and **`boolean paperChecked(String course)`**. These methods can be used to find out whether the exam paper for a course has been set and checked.

Your task in this question is to design and implement a client/server system that accesses the exam paper database over the internet, using socket connections.

(a) Describe a suitable protocol for interaction between client and server. The protocol must allow the client to find out, for any course, whether the exam paper has been set and/or checked.

[4]

The answer should specify the exact information transmitted and its format. For example: the client sends a line of text (terminated by a newline character) consisting of SET or CHECKED, followed by a line of text containing the name of the course. The server responds with a line of text (again terminated by a newline character) containing TRUE or FALSE. It would also be possible to transmit (serialized) objects instead of strings. The specification should be precise enough to enable a client and server to be implemented independently.

(b) Write the code for a simple server, as a class with a **`main`** method, implementing your protocol. You do not need to include error handling, and you do not need to use threads. Assume that the server will run forever.

[6]

The boilerplate code is bookwork, but the server must create and use an ExamDatabase object and must implement the protocol described in part (a).

```
public class ExamServer {

  public static void main(String[] args) {
    ExamDatabase database = new ExamDatabase();
    try {
      ServerSocket server = new ServerSocket(8765,2);
      while (true) {
        Socket connection = server.accept();
        BufferedReader input =
          new BufferedReader(new InputStreamReader(connection.getInputStream()));
        PrintStream output = new PrintStream(connection.getOutputStream());
        String option = input.readLine();
        String course = input.readLine();
        if (option.compareTo("SET")==0)
          output.println(database.paperSet(course));
        else if (option.compareTo("CHECKED")==0)
          output.println(database.paperChecked(course));
        connection.close();
      }
    }
    catch (Exception exception)
      exception.printStackTrace();
  }
}
```

(c) Write the code for a simple client of your server, as a class with a **`main`** method. Your client should find out whether the exam paper for AP(M) has been set, and display the result on standard output.

[6]

Again the boilerplate is bookwork.

```
public class ExamClient {

  public static void main(String[] args) {
    try {
      Socket connection = new Socket("localhost",8765);
      BufferedReader input =
        new BufferedReader(new InputStreamReader(connection.getInputStream()));
      PrintStream output = new PrintStream(connection.getOutputStream());
      output.println("SET");
      output.println("AP(M)");
      String result = input.readLine();
      System.out.println(result);
    }
    catch (Exception exception)
      exception.printStackTrace();
  }
}
```

(d) Explain what is meant by a *multi-threaded* server, and what advantage a multi-threaded server has over a single-threaded server like the one in part **(b)**. Explain how you would change your solution to **(b)** into a multi-threaded server. (You do not need to write the code, but if you want to, you can illustrate your answer with code fragments).

[8]

A multi-threaded server uses its main thread only for accepting connections [1], and starts a new thread in order to process each accepted connection [1]. The advantage is that several client requests can be handled simultaneously [1]. We need to define a class ExamThread extending Thread [1], which can be given a Socket (e.g. as a parameter to its constructor) [1]. The run method of ExamThread contains the code to communicate on the Socket [1], i.e. to receive a message from the client, send the response, and close the connection [1]. The main thread (i.e. the code in ExamServer) constructs an ExamThread for each accepted connection, and calls start on it [1].

Other correct approaches are acceptable, for example using ExecutorService.