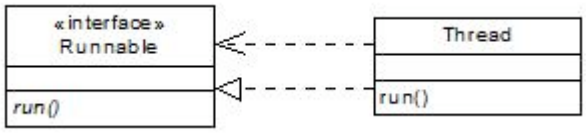
Example answers to PE Questions for GUIs, Networking and Multithreaded Programming

Medialogy, Semester 4, 2009
Aalborg University (Aalborg)

Multithreaded programming

1. Word processor: event handling thread handles keyboard input while worker threads manage spell-checking and periodic backup.
   Web crawler: multiple threads simultaneously get information from many different web pages and store the information in a database.
   Lots of other possibilities.
2. Depends on project.
3. A race condition can occur when two threads can access the same data simultaneously. The final state of the data depends on the specific order in which the instructions in the different threads are executed. The final state of the data is unpredictable because the specific order in which the instructions in the independent threads are executed is not controlled. The threads are therefore "racing" each other.
4. A thread is created using the Thread constructor:
   Thread t = new Thread();
   A thread is started using the start() method:
   t.start();
   A Java thread does not necessarily need to be stopped manually, but if it does, the interrupt method can be used:
   t.interrupt():
5. A new thread can be created by subtyping the Thread class, overriding its run() method and then constructing an instance of this new subclass of Thread. A thread can also be created by defining a new class that implements the Runnable interface. This involves implementing the Runnable interface's run() method. This new Runnable object can then be given as an argument to the constructor of a normal Thread.
   Implementing the Runnable interface is usually preferable to subtyping Thread as it means that the new class can inherit from a class other than Thread.
6. 

7. Interference is when interleaved operations from different threads on the same shared data could corrupt the data. The code segments that contain interfering actions are called critical sections or critical regions.

8. Code in critical regions should be *synchronized* on shared data. This means that while code in a critical region is being executed, the thread that is executing the code has a lock on the shared data which ensures that no other thread can access that shared data while the synchronized critical region of code is being executed.
9. Given an object, obj, that has a method, meth(), then meth() should be declared synchronized if it is important that no other thread should access obj (e.g., change its state) while meth() is executing. A synchronized method acquires a lock on the object referenced by **this**. That is, if meth() is synchronized, then while obj.meth() is executing, meth() has a lock on obj.
10. synchronized(obj) {
        obj.methodA();
        obj2.methodB(obj);
    }
11. Server-side synchronization is where the object to be locked takes responsibility for synchronizing code that uses (often by using a synchronized method). Client-side synchronization is where responsibility for synchronizing critical regions lies with the code that uses the object to be locked. Client-side synchronization is usually accomplished using synchronized statements. Server-side synchronization is generally more secure because each the programmer doesn't have to remember to sychronize client code every time it uses the object to be locked. However, server-side synchronization can lead to unnecessary blocking of threads which can slow down a program. Conversely, client-side synchronization allows the programmer to decide when and when not to synchronize code which can make the code more efficient.
12. Thread.currentThread()
13. Thread.sleep(delay). Throws an InterruptedException.
14. t.join()
15. A thread, *t*, can be stopped by calling *t*.interrupt(). This sets the interrupt status flag in *t* and the next time a method is run which throws an InterruptedException, an InterruptedException is thrown. This can be caught by a catch block which usually simply causes the *t* to return. If *t*'s run() method does not call any methods that throw InterruptedException, then the programmer must manually code a periodic check for whether the thread has been interrupted (using the Thread.interrupted() method).
16. The message *t*.interrupt() causes the interrupt status flag to be set to true for Thread *t*. If Thread.interrupted() called within *t* to check status of interrupt status flag, then the flag is cleared and no InterruptedException is thrown. A second thread, *b*, can find out if *t* has been interrupted by calling *t*.isInterrupted(), which does *not* clear the interrupt status flag. When an InterruptedException is thrown, the interrupt status flag is cleared.

17. *t*.isAlive()
18. A guarded block is a block of code in a thread *t* that is only run when some other thread has set up the conditions necessary for the guarded block in *t* to be run. When execution in *t* arrives at the beginning of a guarded block, *t* should stop executing and wait until a specified condition is satisfied before proceeding.
19. A guarded block can be implemented by using a while loop that continues to iterate until its condition becomes false. This means that the waiting thread is actually continuously using the processor while it is waiting because it is repeatedly checking the condition on the while loop. This uses resources unnecessarily and is called busy waiting.
20. A call to wait() is coded inside a try-catch block inside a while loop whose condition is the one that has to be false in order for the guarded block to be executed:

```
while(!condition) {
        try {
                wait();
        } catch (InterruptedException e) {
                //Handle interruption
        }
}
```

    This while loop has to be inside a synchronized method of some object, *obj*, which is called in the thread that has to be suspended. wait() suspends the current thread and releases the lock on this object. The notifyAll method is called by some second thread on the object *obj*, that is,

    obj.notifyAll();

    This causes the wait() method to reacquire the lock on this object and check the loop condition. wait() throws an InterruptedException if its thread is interrupted while it is waiting. Note that there is no busy waiting.

Graphical User Interfaces

21. Initial thread: usually just the main thread, running when the program starts.
    Event dispatch thread: code that interacts with Swing, particularly event-handling code executes on this thread.
    Worker threads: “Background” threads on which time-consuming tasks are executed.
22. A Java GUI should implement the Runnable interface and execute on the Event Dispatch Thread.
23. A GUI is typically started in Java by calling
    javax.swing.SwingUtilities.invokeLater(gui)
    where gui is a Runnable object (i.e., it implements the run() method).

24. JFrame
25. A Worker thread.
26. The event dispatch thread
27. The content pane contains all the Swing components except the JMenuBar.
28. pack()
29. jFrame.setVisible(true);
30. A JPanel.
31. BorderLayout: Five regions, four around the four sides, one in the middle. Each region can contain one component (which can be a JPanel that contains more than one component).
    FlowLayout: Components placed one after the other in a row, new row started if container not wide enough to hold all the components. Components centred if container wider that sum of widths of components.
    BoxLayout: Stacks components in a column or places them in a row.
    GroupLayout: Defines vertical and horizontal layouts separately. Components arranged into sequential or parallel groups arranged in a hierarchy. A group may contain components or other groups.
32. One (however this can be a JPanel that contains multiple components).
33. BorderLayout.
34. FlowLayout
35. The JButton emits an ActionEvent object whenever it is pressed. This ActionEvent will cause code to execute if there is an ActionEventListener object that has been registered as a listener with the button. This is done using code something like:

    button.addActionListener(listenerObject);

    Now, when the button is pressed, the ActionEvent emitted is received by the registered ActionEventListener object and causes this object to run its actionPerformed(ActionEvent event) method.

Network programming

36. The network interface layer is the hardware used to communicate bits from one physical location to another (e.g., ethernet, bluetooth).
37. 2^32 = 4294967296. We will run out of IP addresses within a few years. IPv6 is a proposed solution to this which provides 2^128 addresses.
38. TCP is a reliable protocol which is connection-oriented. UDP is an unreliable protocol that is datagram-oriented. UDP is faster, TCP is more reliable.

39. UDP is often used for video streaming where speed is paramount and a few lost packets is tolerable. Sometimes used for voice streaming and provides basis for DNS.
40. A socket is an endpoint in a TCP connection. A socket is also used to send and receive datagrams using UDP. Each socket is associated with a particular port on a particular machine. Each application that needs to send or receive data across a network will be listening for data on a specific port. There are 65536 ports, each identified using a 16-bit number. Ports 0-1023 are "well-known ports" and are assigned to server applications executed by privileged processes (e.g. UNIX root server). Such ports include 80 for http communication, 20 and 21 for FTP servers, etc. Ports 1024 – 49151 are registered ports and should be used for specific purposes only (e.g., 8080 for an http server run using ordinary privileges). Ports 49152 and above are dynamic or private ports and can be freely used.
41. java.net
42. Protocol: https
    Host name: www.chromamorph.com
    File name: /papers/icmc.html
    Port number: 8080
    Reference: introduction
    Resource name:
    www.chromamorph.com:8080/papers/icmc.html#introduction
43. URL url = new URL("http://www.smurf.com/splurge.html")
44. openStream() method provides a stream of bytes to read from. InputStreamReader converts the byte stream into a character stream. BufferedReader causes lines or chunks of characters to be read from the character stream provided by InputStreamReader.
45. In the server program, a new ServerSocket object is constructed and assigned to listen on a specified port. The accept method is called on the ServerSocket object which causes the program to block and wait for an incoming connection request which will be accepted. Meanwhile, the client program creates a Socket object and assigns this to the same port listened to by the server program's ServerSocket object on the server's machine. The server and client programs can then obtain input and output streams on their sockets and thus send and receive data across the connection.
46. A DatagramSocket object (socket) is created in the server program and assigned a port number. A DatagramPacket object (packet) is then constructed which can be used to store a packet of data received on the socket using the socket.receive(packet) method. The IP address and port number of the sending application's socket can be found from the received packet using the DatagramPacket's getAddress() and getPort() methods. A packet can be sent back to the client using the socket.send(packet). A DatagramSocket object is also created in the client program and this socket is used to send a DatagramPacket to the IP

address and port number of the server's DatagramSocket. The DatagramPacket can be addressed appropriately by using the DatagramPacket constructor that takes an IP address and port number as two of its arguments:

```
DatagramPacket packet = new
DatagramPacket(buffer,bufferLength,ipAddress,portNumber)
```