

433 Practice 2nd Midterm Answers

- Define a Java class Room. When a room is created, you specify the capacity (an integer). Implement two functions, enter() and leave(). Keep track of the number of threads that have entered a room. If the room is at capacity, a new thread trying to enter the room will block until somebody leaves and the thread can enter without pushing the room over capacity.
Don't worry about handling recursive locks.
 - Provide an additional function doInRoom(Runnable r) that enters the room, invokes the run method of r, and then leaves the room. The leave() method is invoked even if the run method of r throws an exception (although execution of doInRoom still terminates with the exception thrown by the run method).

Answer:

```
public class Room {
    int capacity;
    public Room (int c) {
        capacity = c;
    }
    public synchronized void enter() throws InterruptedException {
        while (capacity == 0) wait();
        capacity--;
    }
    public synchronized void leave() {
        if (capacity == 0) notify();
        capacity++;
    }
    public void doInRoom(Runnable r) {
        // Ignore InterruptedException
        while (true) {
            try { enter(); break; }
            catch (InterruptedException e) {}
        }
        try {
            r.run();
        }
        finally {
            leave();
        }
    }
}
```

2. Say a server wanted to be able to send a stream of messages to a client. You want to make sure that the client receives the messages in order, without skipping any. Allow for the machine the client is running on to crash, with the client migrating to another computer.

Describe the high level design of a system to handle these goals.

Answer: You need to assign sequence numbers to each message so that missing, duplicate and/or out-of-order messages can be detected.

There needs to be some sort of log-on authentication for a client so that if a client reconnects or moves, the server can be sure that it is authorized.

The server should hold onto old messages, so that if a client moves, the server can resend any messages that were not committed by the client before the crash/move.

The server API needs to provide methods both for reconnection and for requesting old messages.

3. Why does a remote object need to implement a remote interface? Alternatively, what is special about a remote interface?

Answer: The remote interfaces tell the RMIC what methods to build stubs for and which interfaces the stub should implement.

Also, if an object wants to reference a remote object, it cannot do so with a reference whose type is that of the remote object (because then the field cannot reference the stub). Instead, fields used to reference remote objects should have a type that is one of the remote interfaces implemented by the remote object.

4. Say you had implemented a HashMap class (ignore the one already provided in the JDK). Is it reasonable for you to declare that your class is Serializable? What would the default serialization do to a HashMap? What if some values appeared multiple times?

If you wrote a custom serialization, how might you do it differently to make serialization more efficient?

Answer: It is reasonable to declare it as Serializable. The standard semantics would be to just capture the structure of the entire HashMap, along with all of the entries. Any duplicate values would correctly be serialized as a single value, referenced multiple times.

For a custom serialization, it would be more efficient to simply encode the hash table size, the number of entries, and then the key-value pairs.

5. Give an example where the Singleton design pattern would be useful.

Answer: In any system where exactly one instance of a service is required, for example a print spooler that controls access to a set of printers. In the examples we've seen, the idea is that having more than one of the service would create a resource contention problem, with multiple services/objects vying for the same physical or virtual (e.g., a window manager) resources.

6. Using the Abstract Factory design pattern, show the declarations of Java classes that are widget factories for the Windows and X operating environments to instantiate GUI objects, with the following interface:

```
interface WidgetFactory {
    public Window createWindow();
    public Icon createIcon(String filename);
    public Menu createMenu(String name, String[] entries);
}
```

Also provide code for a client that uses the X widget factory to create an icon from a file named *icon.gif* and a menu named *Window Ops* with two entries, *Move Window* and *Resize Window*.

What restrictions/requirements are there on the types of the objects returned from the methods in the Windows and X widget factories?

Answer:

```
class XWidgetFactory implements WidgetFactory {
    public Window createWindow();
    public Icon createIcon(String filename);
    public Menu createMenu(String name, String[] entries);
}
```

```
class WindowsWidgetFactory implements WidgetFactory {
    public Window createWindow();
    public Icon createIcon(String filename);
    public Menu createMenu(String name, String[] entries);
}
```

```
/* client code in some method */
/* note the use of base types in all declarations, making the code
   completely generic, only depending on which WidgetFactory is
   instantiated */
WidgetFactory w = new XWidgetFactory();
Icon i = w.createIcon("icon.gif");
String[] s = { "Move Window", "Resize Window" };
Menu m = w.createMenu("Window Ops", s);
```

The return value objects have to be subclasses of the return values of the methods in interface `WidgetFactory` (e.g., if `XWidgetFactory.createIcon()` returns an `XIcon` object, then `XIcon` must be a subclass of `Icon`).