Guidelines

Put your name and class account number on each page before starting the exam. Write your answers directly on the exam sheets, using the back of the page as necessary. Bring your exam to the front when you are finished. Please be as quiet as possible.

If you have a question, raise your hand and I will come to you. However, to minimize interruptions, you may only do this once for the entire exam. Therefore, wait until you are sure you don’t have any more questions before raising your hand. Errors on the exam will be posted on the board as they are discovered. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn’t need to do this at all, so be careful when making assumptions.
1. (8 points) In class we discussed two principles of design patterns: (1) Program to an interface, not to an implementation, and (2) favor composition over inheritance. Explain each of these principles and their advantages in your own words. Use no more than 50 words per principle.

2. (8 points total) The design patterns described in class often involve both abstract classes and concrete classes.

   (a) (2 points) Technically speaking, how does an abstract class differ from a concrete class? In particular, what can you do with a concrete class that you can’t do with an abstract one?

   (b) (4 points) Name the two mechanisms in Java that you can use to implement the concept of abstract classes. Name one benefit of using one mechanism over the other.

   (c) (2 points) Name two examples of conceptual abstract classes (using either of Java’s two mechanisms), taken either from the projects or from the Java class libraries.
3. (17 points) In Project 2, you wrote a web server whose behavior was implemented by subclasses of the class `HttpHandler`; for your reference, we have provided `HttpHandler`'s code below. This class has a method called `handleRequest` that, along with other methods, implements the basic algorithm for handling a web request. At the moment, handling a request consists of two parts: matching the request, and if the match occurs, writing a response in HTML.

Describe how you would modify the project code (which classes and how) so that handling a request occurs in three phases: matching the request, generating output in a generic format, and then formatting that output to be HTML. Use pseudocode, diagrams, text, etc. as necessary to explain your approach.

```java
public abstract class HttpHandler implements Comparable {

    protected HandlerOrder orderConstraint;

    public int compareTo(Object that) {
        if (that instanceof HttpHandler) {
            HttpHandler handler = (HttpHandler)that;
            return (this.orderConstraint.compareTo(handler.orderConstraint));
        }
        throw new ClassCastException();
    }

    public boolean handleRequest(HttpRequest request, StringBuffer out) throws HandlerException {
        if (matchRequest(request)) {
            writeResponse(out);
            return true;
        }
        return false;
    }

    public abstract boolean isExclusive();

    protected abstract boolean matchRequest(HttpRequest request);

    protected abstract void writeResponse(StringBuffer out) throws HandlerException;
}
```
4. (17 points) Referring once again to Project 2, recall that all handlers return their output in HTML. Suppose that you want to be able to write web servers that not only output HTML, but output other formats as well, e.g. plain text, audio, etc. Consider the effect of this change on the web server architecture: in the original server, only the function of the handlers (e.g. GetRecordsHandler, LogEventHandler, etc.) varies; in the proposed situation, both the function of the handlers, and the way handler output is formatted (e.g. as HTML, plain text, postscript, etc.) can vary.

Assuming that the way its output is formatted is determined at the time a handler is created (and not by the BasicServer), how would you implement this change so that new handlers and new formatting algorithms can be added with a minimum amount of code? What design pattern(s) could you use? Use class diagrams, pseudocode, text, etc. to illustrate your answer. Justify your design choices.
5. (25 points) Write a Junit test class that tests implementations of the Log interface from Project One.¹ Your class should have three different test cases, with one no-argument method for each. Include comments with each test case indicating what it intends to test. Your test class should include all the necessary code needed to be compiled correctly. You do not need to add a main method.

The Log interface:

```java
public interface Log {
    public void add(LogRecord record);
    public LogRecord[] getAll(long windowMS);
    public void setFilter(String pattern);
}
```

Your test case class:

```java
import junit.framework.*;
public class LogTest extends TestCase {

    public static Test suite() {
        return new TestSuite(LogTest.class);
    }

    public void testAdd() {
        // Test case for add method
    }

    public void testgetAll() {
        // Test case for getAll method
    }

    public void testSetFilter() {
        // Test case for setFilter method
    }
}
```

¹The semantics of the getAll function changed in Project Two with regard to an input of 0; you can use either semantics in your test cases.
6. (25 points) Write an implementation of the Log interface (shown in Problem 5) that acts as a security proxy for another Log, using the Proxy pattern. The constructor for the proxy will indicate which of the underlying Log’s methods will be disabled by the proxy. For example, the BasicServer of Project Two might allow a LogEventHandler to add records to its log, but not to look at that log’s contents or set its filter.