Midterm

CMSC 433

Programming Language Technologies and Paradigms

Spring 2003

March 20, 2003

Instructions

This exam contains 12 pages, including this one. Make sure you have all the pages. Put your name and class account number (last 3 digits only) on each page before starting the exam.

Write your answers on the exam sheets. If you finish at least 15 minutes early, bring your exam to the front when you are finished; otherwise, wait until the end of the exam to turn it in. Please be as quiet as possible.

If you have a question, raise your hand. 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.

You may avail yourself of the *punt* rule. If you write down *punt* for a question, your will earn 1/5 of the points for that question (rounded down).

Question	Score	Max
1		20
2		20
3		30
4		30
Total		100

Question 1. Short Answer (20 points).

a. (5 points) Consider the following class that represents a book:

```
class Book {
    String author;
    String title;
    String publisher;


    boolean equals(Book b) {


        return (author.equals(b.author) &&


            title.equals(b.title) &&


            publisher.equals(b.publisher));


    }
}
```

The `equals` method for this class does not meet the standard Java specification. Edit the above code to fix the problem(s). You do not have to fix or supply any other functions.

Answer:

```
public boolean equals(Object o) {
    if (!o instanceof Book) return false;
    Book b = (Book) o;
    return ...  as above
}
```

b. (5 points) The following code is rejected by `gjc`:

```
class Evil<A> {
    static A x;
    static A get() { return x; }
    static void set(A y) { x = y; }
}
```

Complete the following example, which shows that if the above code were allowed by `gjc`, then a source program with no type casts may fail with a run-time type error:

```
Integer i = new Integer(42);
String s;

Evil<Integer>.set(i);
```

Answer: `s = Evil<String>.get();`

c. (5 points) Does having a `final abstract` method ever make sense? Explain.

Answer: No. A `final` method cannot be overridden, and an `abstract` method has no implementation. Thus a `final abstract` method could never be implemented and is useless.

d. (5 points) Consider the `cat` method, which opens the specified file and prints it to standard output:

```
class FileReader {
    // throws FileNotFoundException if the file s is not found
    public FileReader(String s) throws FileNotFoundException;

    // throws IOException if an I/O error occurs
    public int read() throws IOException;
    ...
}

public static void cat(String s) throws Exception {
    FileReader f = null;
    int c;

    try {
        f = new FileReader(s);
        while ((c = f.read()) != -1)
            System.out.write(c);
    }
    finally {
        f.close();
    }
}
```

What exception will `cat(s)` throw if the file `s` does not exist? You only need to specify the class of the exception, not its message contents.

Answer: A `NullPointerException` will be thrown. The exception is raised before `f` is assigned to, and then when `f.close()` is called `f` will still be null. This is an example of how an exception can be lost in Java.

Question 2. Glass Box Testing (20 points). Below is the sample implementation of the `contains` method from project 2. On the chart on the next page, draw a series of pairs of graphs G_1 and G_2 such that invoking G_1.`contains`(G_2) on all pairs will cover all statements in `contains`. You should assume that the `Iterator`s constructed on lines 1 and 7 walk through the vertices in alphabetical order (hint: you will need to use this assumption to guarantee coverage of one particular statement.). We've given you an example to get you started. You can leave lines in the chart blank if you don't need them to cover all statements.

```
class MyGraph implements Graph {

    HashMap nodes; // maps vertex labels to MyVertex objects

    /**
    * Determines if graph g can be embedded in this graph.
    * A graph g1 is contained in graph g2 iff for each
    * vertex v in g1, an equivalent vertex with the
    * same label exists in g2, and for all edges e in g1,
    * an edge between the corresponding vertices exists
    * in g2.
    *
    * @param g graph to check if it can be embedded
    * @return true iff g can be embedded in this graph
    */
    public boolean contains(Graph g) {
1       Iterator i = g.vertices().iterator();

2       while (i.hasNext()) {
3           Vertex from = (Vertex) i.next();
4           MyVertex myFrom = (MyVertex) nodes.get(from.getLabel());
5           if (myFrom == null)
6               return false;
7           Iterator j = g.outEdges(from);
8           while (j.hasNext()) {
9               Vertex to = (Vertex) j.next();
10              MyVertex myTo = (MyVertex) nodes.get(to.getLabel());
11              if (myTo == null)
12                  return false;
13              if (!myFrom.outEdges.contains(myTo))
14                  return false;
            }
        }
15      return true;
    }
    ...
}
```

G_1	G_2	Statements covered by G_1.`contains`(G_2) (fill in)
$x \rightarrow y$	x	1,2,3,4,5,7,8,15
x	z	1,2,3,4,5,6
x	$x \rightarrow y$	1,2,3,4,5,7,8,9,10,11,12
x y	$x \rightarrow y$	1,2,3,4,5,7,8,9,10,11,13,14

Question 4. Generic Java (30 points). In project three, you wrote a CachedGoogleMap class that caches and reuses the results of Google queries. In this question, you will write a generic decorator for wrapping a function with a cache. Consider the following interface for classes that have a function from `A` to `B`:

```
public interface Function<A, B> {
    B func(A x);
}
```

a. (21 points) Write a class `CachedFunction` that, constructed with an argument `f` that is an instance of a class that implements the `Function` interface, produces a new object that meets the same interface and caches and reuses the results of `f.func()`. As in project 3, your program may not include any type casts. Hint: You may wish to use the `Map` interface, shown below. Don't forget to fill in the `implements` portion of `CachedFunction`.

```
public interface Map<A, B> {
    boolean containsKey(A key);
    boolean containsValue(B value);
    B get(A key); // returns null if key not in map
    B put(A key, B value); // returns previous value or null if none
    ...
}
public class HashMap<A, B> implements Map<A,B> { ...  }
```

(Write your answer on the next page)

```
public class CachedFunction<A, B> implements Function<A, B>                 // fill in

HashMap<A, B> cache = new HashMap<A, B>();
Function<A, B> f;




    CachedFunction(Function<A, B> f) {
        // fill in
        this.f = f;














    }

    B func(A x) {
        // fill in

        B val;

        if ((val = cache.get(x))) return val;
        val = f(x);
        cache.put(x, val);
        return val;







    }
}
```

b. (9 points) Consider the following hypothetical usage of the classes from part a:

```
Function<Integer, String> f = ...;
CachedFunction<Integer, String> g = new CachedFunction<Integer, String>(f);
Integer i = new Integer(1);
String f1, f2, g1, g2;

g1 = g.func(i);
g2 = g.func(i);
f1 = f.func(i);
f2 = f.func(i);
```

Assuming that x`.equals(`x`)` is true for any x, which of the following are guaranteed to be true? For the ones, if any, that are guaranteed to be true, write "true." For the ones, if any, that aren't guaranteed to be true, give an example of a circumstance under which they are not true.

- `g1.equals(g2)`

  True

- `f1.equals(f2)`

  Sometimes false. `f` may have side effects and return a different value even if called with the same argument.

- `g1.equals(f1)`

  Sometimes false, as above.

Question 4. Design Patterns (30 points). In this question, you must re-implement the interface between Proxy and ProxyCounter from project 1 using the Observer design pattern. We will also generalize the ProxyCounter class to be a WebLog that can tally information from multiple sources.

Here are the observer interfaces you will use for this question:

```
public interface StringObserver {
    public void update(Subject s, String msg);
}

public interface Subject {
    public void attach(StringObserver o);
    public void detach(StringObserver o);
    public void notify();
}
```

On the last two pages, implement the constructor, `attach`, `detach`, and `notify` methods of `Proxy` (which is a `Subject`) and the constructor, `observeMe`, and `stopObserving` methods of `WebLog` (which is a `StringObserver`). The constructor for `Proxy` should ask the `WebLog` to observe the proxy. You do not need to implement the `update` method of `WebLog`. The method comments in the code skeleton we've given you should help jog your memory about observers.

Hints: Assume that the other parts of the MiniServlet interface are already implemented for `Proxy` and `WebLog`. Your implementation should allow for the `Proxy` to be observed by multiple, distinct `Observers`, and for the `WebLog` to observe multiple, distinct `Subjects`. The same observer may not be attached more than once to the same subject, and the same subject may not be observed more than once by the same observer.

For purposes of this question, we will use the Singleton pattern for MiniServlets, so that we don't have to keep on creating them (the resulting design isn't threadsafe, but that is a topic for after Spring break).

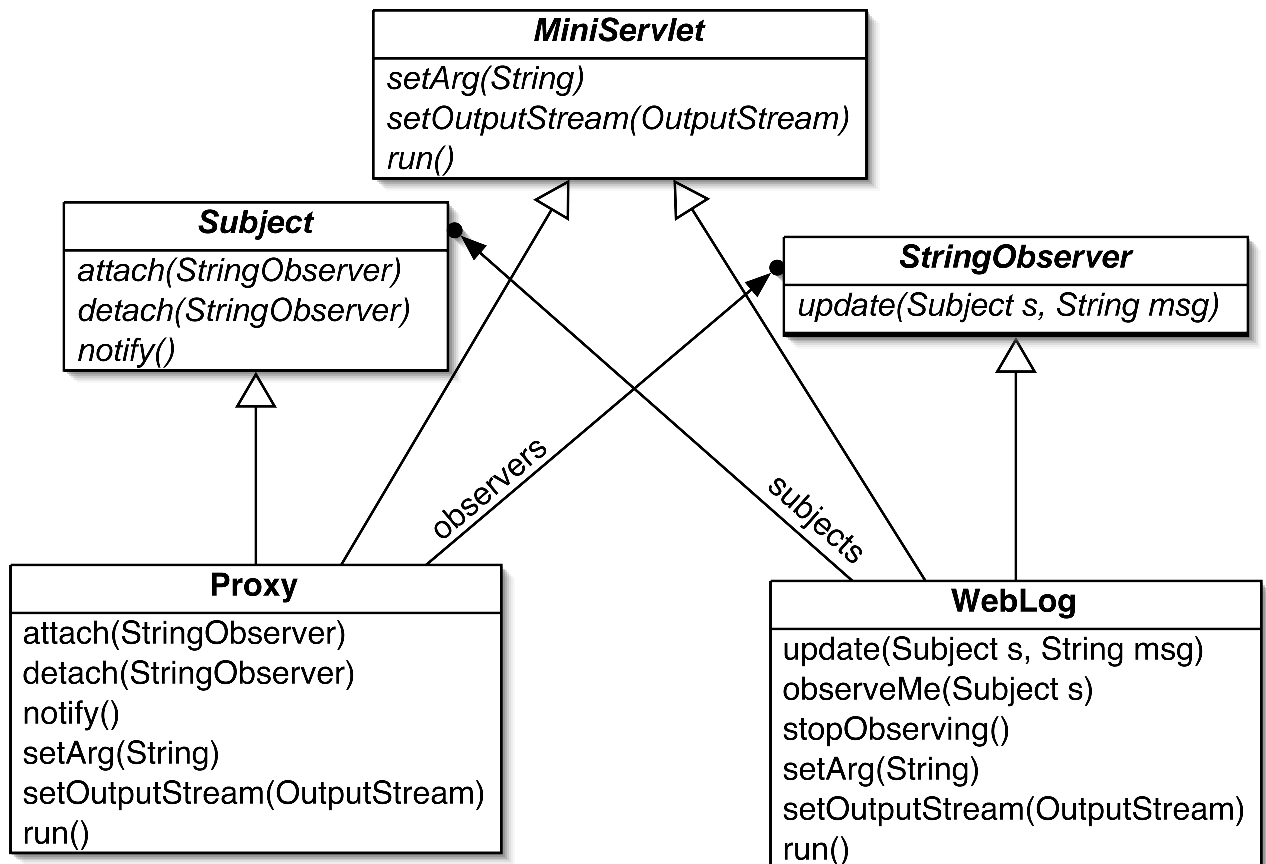
For your reference, here are the `MiniServlet` interface and a portion of the interfaces for `Set` and `Iterator`. You can use `HashSet` as a concrete implementation of `Set`.

```
public interface MiniServlet extends Runnable {
    public void setArg(String arg);
    public void setOutputStream(OutputStream out);
}
public class HashSet implements Set;

public interface Set {
    // Returns true and adds o to the set if not already present; otherwise returns false
    boolean add(Object o);
    // Returns true and removes o from the set if present; otherwise returns false
    boolean remove(Object o);
    Iterator iterator();
}

public interface Iterator {
    boolean hasNext();
    Object next();
}
```

Here is a class diagram for the observer pattern as used in this question.



```
public class Proxy implements MiniServlet, Subject {
    public static final Proxy INSTANCE = new Proxy();
    private String arg;
    public void setArg(String s); { arg = s; }
    public void setOutputStream(OutputStream out) { ...  }
    public void run() {
        notify(); // log request with any observers
        ...
    }

    // hint:  you'll need to add an instance variable here

    HashSet observers = new HashSet();




    // Should ask the WebLog to listen to this proxy
    private void Proxy() {
        // fill in

        WebLog.INSTANCE.observeMe(this);

    }
    // Invoked when o wants to listen to this object
    public void attach(StringObserver o) {
        // fill in

        observers.add(o);





    }
    // Invoked when o no longer wants to listen to this object
    public void detach(StringObserver o) {
        // fill in


        observers.remove(o);





    }
    // Should send arg to all observers
    public void notify() {
        // fill in

        for (Iterator i = observers.iterator(); i.hasNext(); )
            ((StringObserver) i.next()).update(this, arg);




    }
```

```
public class WebLog implements MiniServlet, StringObserver {
    public static final WebLog INSTANCE = new WebLog();
    public void setArg(String arg); { ...  }
    public void setOutputStream(OutputStream out) { ...  }
    public void run() { ...  }
    public void update(Subject a, String msg) { ...  }

    // hint:  you'll need to add an instance variable here

    HashSet subjects = new HashSet();



    private WebLog() {
        // fill in








    }

    // Invoked to tell this to observe subject s
    public observeMe(Subject s) {
        // fill in

        s.attach(this);
        subjects.add(s);


    }

    // Invoked when this WebLog should stop listening to any Subjects
    public void stopObserving() {
        //fill in

        for (Iterator i = subjects.iterator(); i.hasNext(); )
            ((Subject) i.next()).detach(this);






    }
}
```