This quiz is to be taken **by yourself** with closed books, closed notes, no electronic devices.

1) The following code is an example of what design pattern?

```java
public class Foo {
    private volatile static Foo foo;
    private Foo() {}
    public static Foo x() {
        if ( foo == null ) {
            synchronized (Foo.class) {
                if ( foo == null )
                    foo = new Foo();
            }
        }
        return foo;
    }
    // Other methods ...
}
```

The main advantage(s) to the above pattern is we have: (Circle all that apply)

A) Lazy creation of the object  
B) Eager creation of the object  
C) Only synchronize the first time through  
D) No need to synchronize since we are using volatile static  
E) Users of this class only need to create one Foo object on their own to invoke Foo's other methods  
F) Synchronize each time to handle multiple threads

2) Suppose you have the following ant targets in a build.xml script (the bodies of the targets are not important):

```
<target name="come-to-class" />  
<target name="get-enough-sleep" />  
<target name="get-cup-of-coffee" />  
<target name="stay-awake-in-class" depends="get-enough-sleep" />  
<target name="pay-attention-in-class" depends="come-to-class,stay-awake-in-class" />  
<target name="study-hard" />  
<target name="get-good-grades" depends="pay-attention-in-class,study-hard" />  
```

List all the targets in the order they would be run by ant if you entered the command:

```
ant get-good-grades
```

(Continued on other side)
3) Match the general characteristics with the type of Factory pattern.

<table>
<thead>
<tr>
<th>Relies on inheritance</th>
<th>Relies on object composition</th>
<th>Allow a class to defer object instantiation to its subclasses</th>
<th>Not a bona fide design pattern</th>
<th>Create families of related objects without having to depend on their concrete implementation</th>
<th>A variant of this is used to model the Singleton pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>______</td>
<td>______</td>
<td>______</td>
<td>______</td>
<td>______</td>
<td>______</td>
</tr>
</tbody>
</table>

4) The Decorator pattern: (Circle all that apply)

A) uses subclassing to extend behavior
B) enables adding new behaviors at runtime
C) enables adding new behaviors at compile time
D) is used in the java.io package
E) uses abstract (vs. concrete) classes to wrap a component with any number of decorators
F) changes behavior of their components by adding new functionality usually before or after method calls to the component

5) Duplicate code is a common bad smell. What refactoring technique(s) is/are commonly used when the duplicate code is in several related classes with a common superclass?

What refactoring technique(s) is/are commonly used when the duplicate code is in several unrelated classes?

6) Give two advantages for using a Null Object vs. always checking/using null? For example, with a List what advantage does a Null List object provide vs. coding with only the null reference?

1) ______

2) ______

7) What is the name of the bad smell when one class spends most of its time/operations requesting/requiring private members from another class?

What refactoring technique would you use to take care of this bad smell?