**CS2340 Exam 2 Fall 2013**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**GT Login Id:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |
| --- | --- | --- |
| **Topic** | Possible | Earned |
| **OO Design Principles** | **24** |  |
| **OO Design Patterns** | **24** |  |
| **User Interface Evaluation** | **24** |  |
| **OO Misc Topics** | **28** |  |
| **TOTAL** | **100** |  |

**I certify that I have complied with the Georgia Institute of Technology honor code during this examination. I have neither received nor given help (other than the single sheet of paper authorized).**

**Signed:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **OO Design Principles.**  ***Choose the one best answer***

A. Consider the following code snippet:

public interface IWorker {

void doWork( );

void scheduleWork( );

void calculatePay( );

}

This code:

a. Complies with the interface segregation principle.

b. Violates the interface segregation principle.

c. Complies with the program to interfaces, not concrete class principle.

d. Both A and C are correct.

B. Consider the following class definition:

public class Employee {

public int calculatePay(int hours) { }

public Date getBirthday( ) { }

public void printEmployeeReport( ) { }

}

This code:

a. Complies with the Liskov Substitution Principle.

b. Violates the Liskov Substitution Principle.

c. Complies with the Single Responsibility Principle.

d. Violates the Single Responsibility Principle.

e. None of the above are correct.

C. Consider the following code snippet:

public void calculateArea(Shape s) {

if (s instanceOf Rectangle) { s.doSomethingSpecial( ); }

else s.area();

}

This code:

a. Complies with the Liskov Substitution Principle.

b. Violates the Liskov Substitution Principle.

c. Complies with the Single Responsibility Principle.

d. Violates the Single Responsibility Principle.

e. None of the above are correct.

D. When designing an application, following the OO design principles helps achieve:

a. Low cohesion and High coupling

b. Low cohesion and Low coupling

c. High cohesion and High coupling

d. High cohesion and Low coupling

e. None of the above, coupling and cohesion do not apply to OO design.

E. When examining code during a code review, you see several switch statements that need to decide what to do based on the type of the Tile. What issue should you raise?

a. Nothing, switching on type is a standard coding practice

b. Change the switch statement to embedded if statements to improve performance

c. Replace the switch statement with polymorphism to support open-closed principle

d. Create subclasses of Tile (one for each type) that each know their special behavior.

e. Both c and d are correct.

F. Your team is using Model-View-Presenter for your design. During a code review you look at the constructor for one of the presenters and notice it takes in concrete classes like:

EmployeeEditPresenter(CompanyModel model, EmployeeEditView view)

What issue would you raise?

a. None, the presenter has to know about the model and the view

b. Should change the model to an interface to comply with interface segregation principle.

c. Should change the model and the view to intefaces to comply with dependency injection

d. Should not pass these as parameters but create them in the constructor.

G. Dependency Injection is very popular because:

a. It reduces hard coded dependencies in your code

b. It allows you to substitute mock classes during testing

c. It makes the code easier to maintain and change

d. All the above answers are correct

e. None of the above answers are correct

H. Combatting the morning after syndrome in software development (unable to build on a stable codebase) can be helped by following the:

a. Code freeze protocol

b. Using a version control system like Git or Subversion

c. Following the Acyclic Dependency Principle

d. Using Responsibility-Driven Design

2. **OO Design Patterns.** ***Select the one best answer.***

A. Your design team has decided that you need to be able to create macros in your application. The easiest way to do this is to:

a. Encapsulate the different actions available as objects. A macro would then be a collection of these objects.

b. Use the Command design pattern

c. Use the javax.cmd.macro package

d. Both a and b are correct.

B. The Java thread pool executor is an example of what design pattern:

a. Factory

b. Command

c. Active Object

d. All of the above

e. None of the above

C. Our mule game has 3 kinds of Mules, but the only real difference is how production is calculated. We should:

a. Use inheritance and create the three mule subclasses.

b. Use the strategy pattern, and put the unique behavior in its own class.

c. Use the state pattern to control production

d. Have a separate Production class, that asks the Mules for their internal data and does the calculation.

D. Your team has a database connection that shared across multiple classes. One team member suggests making it a Singleton.

a. I agree! Singletons are awesome.

b. Warn them that Singletons are essentially global data and may cause problems in the design later

c. Warn them that Singletons may make reuse of classes very difficult because the classes will be coupled to the Singleton.

d. Both B and C are correct

e. No answer is correct.

E. The design pattern that is most closely like the Java Listener interface in Swing is:

a. Observer

b. Active Interface

c. State

d. Handler

F. Using a single instance of a River, Mountain and Plains object to record unchanging information like amount of production, icon shapes, etc. and then having a separate Tile class that knows information that is different for every Tile (like owner, mule placement, etc) is an example of:

a. Delegation

b. Inheritance

c. Single Responsibility

d. Flyweight

3. **User Interface Design and Evaluation. *Match the best answer to the topic. There is only one answer per topic.***

|  |  |
| --- | --- |
| \_\_\_\_ Heuristic Evaluation | A. Designing a rotating control so that turning it to the right increases the value |
| \_\_\_\_ Cognitive Walkthrough | B. An idea formed by the user of how your system works |
| \_\_\_\_ Think Aloud | C. Having a command prompt where the user enters memorized commands |
| \_\_\_\_ Natural Mapping | D. Giving users a label that looks like a button. |
| \_\_\_\_ False Affordance | E. Putting a button bar on the screen so that the user can pick an action based on what seems needed |
| \_\_\_\_ Mental Model | F. Checking how easy your system is to use by a novice user. |
| \_\_\_\_ Recognition | G. Checking your user interface against a set of rules that are generally accepted as good design |
| \_\_\_\_ Recall | H. Having an actual user operate your system and provide you with feedback |

***4.***  **Other Topics *Choose the one best answer.***

A. One common mistake that people made in designing the Mark 4 coffee pot in class was:

a. Failure to focus on all the hardware devices in the pot

b. Failure to comply with the provided API

c. Failure to focus on behavior as opposed to just nouns

d. There were no mistakes, most designs correctly captured the coffee pot behavior

Assume we have a piece of code that looks like the following (used for next 2 questions):

public boolean check(int x, int y) {

if (x < 0 ) { return false;}

if (y < 0 ) { return false;}

if (x > 30 && y < 7) {return false;}

return true;

}

B. To achieve *statement coverage* you should use the following inputs:

a. (x = 0, y = 0), (x = 30, y = 7), (x = -1, y = 4)

b. It is impossible to achieve statement coverage with these conditions

c. (x= -1, y=-1), (x = 3, y = -1), (x=34, y = 3)

d. (x= -1, y=-1), (x = 3, y = -1), (x=34, y = 3), (x = 34, y = 9)

C. To achieve branch coverage you should add the following additional inputs to those in B above:

a. You do not need additional inputs, achieving statement coverage in this case also gives branch coverage

b. (x = 30, y = 7)

c. (x = -3, y = 9)

d. (x = 4, y = -10)

E. In the design critiques so far, the most common architecture presented has been:

a. Layered

b. Master-Controller

c. Event Driven

d. Model-View-Presenter

F. Your partner implements a thread to handle a separate task, but your main thread seems to still hang and not execute properly. The first thing to check is:

a. Thread priority not set correctly

b. Executor Service overloaded

c. Calling run() method directly rather than calling the start() method

d. Failure to implement Runnable interface

G. Given a method that is supposed to take as input an integer in the range of 0 - 1000 inclusive. To test all equivalence partitions, you would need the following test cases:

a. It is not possible to apply equivalence partitions to this input

b. -45, 18, 3432

c. 0, 500, 1000

d. None of the above answers are correct.

H. Given a method that takes an integer as above (0-1000), what inputs are required to test boundary conditions?

a. It is not possible to apply the boundary technique to this input

b. 0, 1000

c. -1, 0, 1000, 1001

d. None of the above answers are correct