# Practice Problems

Review, with SOME solutions

# Multiple Choice

1. Select the best functional requirement from the list of requirements below.
   a) A warning dialog should pop up if the student's assignment does not compile.
   b) The system needs to manage the student grades.
   c) The marker should be able to use this tool to aid the assignment evaluation process.
   d) The marker must be able to edit comments in the marking report.

2. You should use this design pattern when extension by subclassing is impractical and you need to add responsibilities to individual objects dynamically.

   a) Composite
   b) Adapter
   c) Decorator
   d) Singleton
   e) None of the above

3. Which design principle is violated in the following code snippet?

```java
public class FurnitureTable {

import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import com.sample.model.Content;
import com.sample.model.FurnitureEvent;
import com.sample model.FurnitureListener;
import com.sample.model.ContentManager;
import com.sample.model.Home;
import com.sample.model.HomePieceOfFurniture;
import com.sample.model.SelectionEvent;
import com.sample.model.SelectionListener;
import com.sample.model.UserPreferences;
import com.sample.model.HomePieceOfFurniture….;
import com.sample.model.FurnitureCatalog;
...
```

   a. Open/Closed Principle
   b. Law of Demeter
   c. Liskov Substitution Principle
   d. High Cohesion
   e. Low Coupling
   f. None of the above

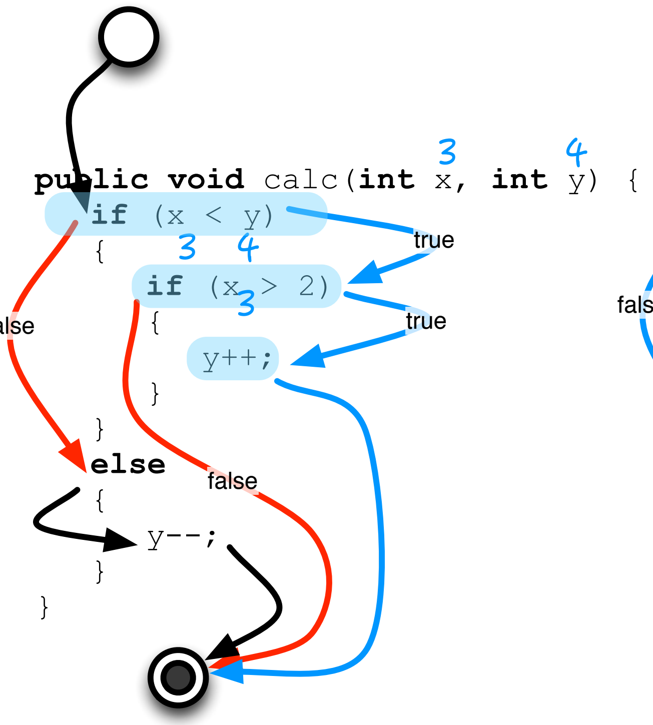
5. You have to test a *factorial* function, but you don't have access to the code. Which of the following is the best set of test inputs?

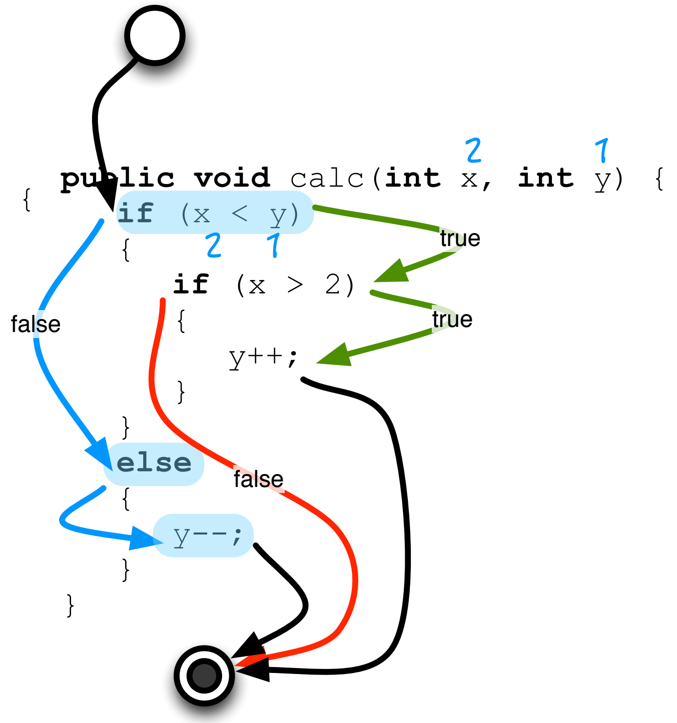
   a) 1, 2, 5, 200
   b) -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5
   c) -4, 0, 1, 5, 250
   d) All the positive integers until the program crashes

6. What type of coverage would you get if you tested the following code with the values

   `x = 3, y = 4` and `x = 2, y = 1.`

```
public void calc(int x, int y) {
    if (x < y)
    {
        if (x > 2)
        {
            y++;
        }
    }
    else
    {
        y--;
    }
}
```





   a. Statement coverage but not branch or path coverage
   b. Statement and branch coverage but not path coverage
   c. Statement, branch and path coverage
   d. None of the above

# True/False+Why

1. You cannot use XP if you cannot create automated tests for your software.
2. Testing is used to demonstrate that software is free of errors.
3. It is expected that a team of experienced software developers using an agile development process will produce smelly code.
4. Design patterns encapsulate existing, well-proven re-usable blocks of design and therefore should be applied without modification.
5. Reusing previously tested and deployed software components always leads to higher software quality.
6. Clear and specific requirements can be very useful when planning acceptance testing.
7. Recall is easier than Recognition when it comes to user interface operation.

# Short Answer

1. You decided to start a new software company with some friends. Unfortunately, most of your friends who are starting the company with you are inexperienced programmers and don't have great programming style. They tend to leave out comments, format code in a hard to understand way, and generally deliver code with poor style. What could you do to improve this situation and why would it work?

2. For each of the following design patterns, provide a two-three sentence description, an example of when it could be applied (a different example than the ones in the notes), and at least one benefit of using it.

   a. Abstract Factory
   b. Decorator
   c. Singleton

3. For each of the following design principles
   - define it in 2 sentences or less
   - explain how following it helps you create better designs

   a. information hiding
   b. open/closed principle
   c. Law of Demeter

4. Define each of the following, and identify a key way in which each plays a role in the development of quality software.
   *pair programming*
   *code reviews*
   *regression tests*
   *system tests*

5. A software developer on your team refuses to refactor, claiming that since the code works it's a waste of time to refactor because you could all be working on new features instead. What arguments would you make in favour of refactoring code when necessary?

6. Explain why a test set which achieves branch coverage for a particular program also achieves statement coverage for that program. You can assume there is no "dead code" in the program (i.e. statements which could never be executed).

   *Hint: you could (but don't have to) use a short proof-by-contradiction e.g. you could start with "Assume for the sake of contradiction, there is a test set which achieves branch coverage but not statement coverage for some program …."*

# Longer Answer …

7. Implement a simple multi-player turn-based game framework using the Template Method pattern. You only need to implement one base class, not any sub-classes or other data-structures.

*The logical flow for the template (implemented in the base-class):*

*1. Setup. This step will determine the number of players.*

*2. Repeat, for each player in some fixed order:*

*Player takes their turn, if they win, immediately goto Finish.*

*3. Finish.*

*Subclasses would implement the logic of Setup, Player's turns, and Finish, not the base-class.*
*Subclasses would only implement individual steps, not any of the overall control-flow*

```java
public abstract class TemplateMethodQuestion {

    public void run() {
        int numPlayers = setup();
        int turn = 1;
        while(true) {
            boolean won = playerTurn(turn);
            if(won) {
                break;
            }
            turn++;
            if(turn > numPlayers) {
                turn = 1;
            }
        }
        finish();
    }

    public abstract int setup();

    /*currentPlayer argument is optional here, could potentially
    keep track of this in the sub-class*/
    public abstract boolean playerTurn(int currentPlayer);

    public abstract void finish();
}
```

# Longer Answer…

8. Consider the following method:

```
String triangleType (int x, y, z) {
    String r="scalene";
    if (x==y || y==z)
        r="equilateral";
    if (x==z)
        r="isosceles";
    if (x <= 0 || (x+y) <= z)
        r="";
    return r;
}
```

a. Draw the CFG for the method according to the previous described logic here.

b. Give a test set of minimal size for `triangleType` which provides branch coverage.

1. (x==y) && (x == z) && (x <= 0)

   { x = -1, y = -1, z = -1}

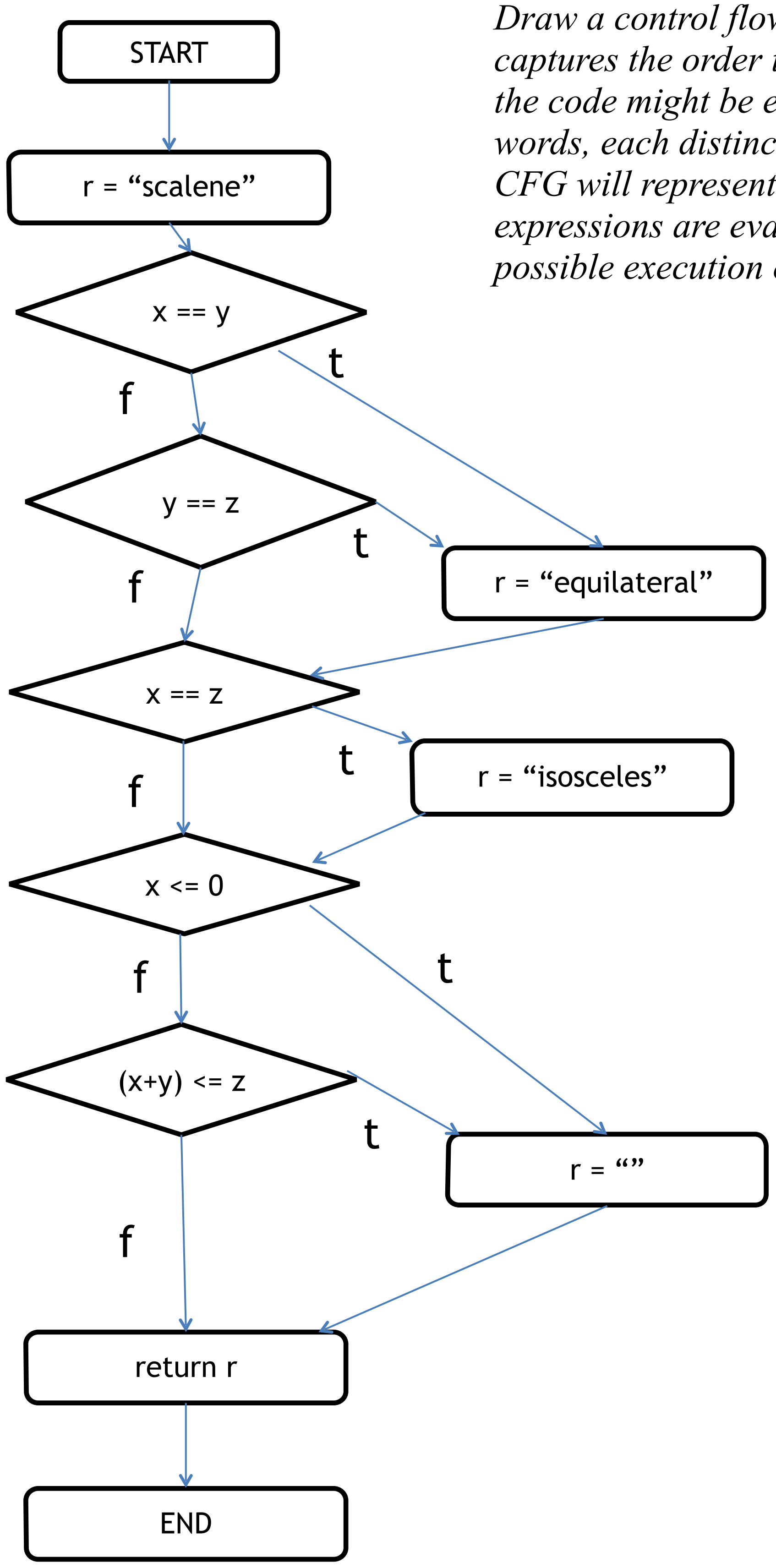
2. (x != y) && (y == z) && (x != z) && (x > 0) && ((x+y) > z)

   { x = 1, y = 5, z = 5}

3. (x != y) && (y != z) && (x != z) && (x > 0) && ((x + y) <=z)

   { x = 1, y = 2, z = 10}

*Draw a control flow graph which captures the order that expressions in the code might be evaluated. In other words, each distinct path through the CFG will represent the order that the expressions are evaluated for one possible execution of the method*



# Longer Answer…

9. You are testing a method that computes the cost for a golf game reservation. The regular cost is \$35. However, a senior (age 55 and over) and a junior (age 18 and below) pay a reduced rate at \$30. Also, if the time is on a weekday, an additional 10% discount is applied. Suppose that the method takes the age of a player and day of the week as the input. Define a test set for the method using equivalence classes and boundary value analysis. For each test case input, also provide the expected output.

Equivalence Classes:

1. ReducedRate && Weekday
2. !ReducedRate && Weekday
3. ReducedRate && !Weekday
4. !ReducedRate && !Weekday

Test cases:

Could have included more edges cases around each boundary (e.g. age = 54, age = 55, age = 56) but only one is required for exam.

a. Eq. Class 1

   **{ age = 10, day= “Wednesday”} = \$27**

b. Eq. Class 2

   **{ age = 25, day= “Wednesday”} = \$31.50**

c. Eq. Class 3

   **{ age = 60, day= “Saturday”} = \$30**

d. Eq. Class 4

   **{ age = 25, day= “Saturday”} = \$35**

e. Boundary between 1 and 2

   **{ age = 18, day= “Wednesday”} = \$27**

   **{ age = 55, day= “Wednesday”} = \$27**

f. Boundary between 1 and 3

   **{ age = 10, day= “Saturday”} = \$30**

   **{ age = 60, day= “Friday”} = \$27**

g. Boundary between 1 and 4

   **{ age = 18, day= “Saturday”} = \$30**

   **{ age = 55, day= “Friday”} = \$27**

h. Boundary between 2 and 3

   **Same as boundary between 1 and 3**

i. Boundary between 2 and 4

   **{ age = 25, day= “Saturday”} = \$35**

   **{ age = 25, day= “Friday”} = \$31.50**

j. Boundaries between 3 and 4

   **{ age = 18, day= “Saturday”} = \$30**

   **{ age = 55, day= “Sunday”} = \$30**

# Questions from Slides

Write a black-box test set for the ArrayList<E>.get(int index) method

Specification:
public E get(int index) Returns the element at the specified position in this list.

Parameters:index - index of the element to return

Returns:the element at the specified position in this list

Throws: IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())

*equivalence classes: one negative test, one >size test, one in between*

*boundary tests: -1,0,1 and size-1, size, size+1*

# Questions from Slides

Write a black-box test set for the
Comparable<Integer>.compareTo(Integer o) method

Specification: Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object
Parameters: o - the object to be compared.
Returns :a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
Throws: NullPointerException - if the specified object is null
ClassCastException - if the specified object's type prevents it from being compared to this object.

*equivalence classes: null object, incomparable object, <, =, >*