Name:

NetID:

Final Exam

CS 427 Software Engineering I (Fall 2014)

TIME LIMIT = 3 hours
COVER PAGE + 6 PAGES

Upon receiving your exam, print your name and netid neatly in the space provided above; print your netid in the upper right corner of every page.

This is a closed book, closed notes examination. You may not use calculators or any other electronic devices. Any sort of cheating on the examination will result in a zero grade. We cannot give any clarifications about the exam questions during the test. If you are unsure of the meaning of a specific question, write down your assumptions and proceed to answer the question on that basis.

Do all the problems in this booklet. Do your work inside this booklet, using the back of pages if needed. The problems are of varying degrees of difficulty so please pace yourself carefully, and answer the questions in the order which best suits you. Answers to essay-type questions should be as brief as possible. If the grader cannot understand your handwriting you may get 0 points.

Part One: _____/50

Part Two:

White Box Testing _____/7 **Total Score _____/100**

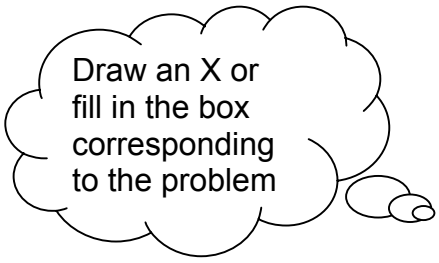
Black Box Testing _____/10

Class Invariants _____/5

Design Patterns _____/12

Refactoring _____/16

Name _____ NetID _____



#	a	b	c	d	e
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

#	T	F
21		
22		
23		
24		
25		

Part One (2 points each)

The first section of the test is multiple choice or true/false. Once you determine an answer, draw an X or fill in the box on the answer sheet (on the first page). The five columns (a-e) for the first 20 questions represent the multiple-choice answers. For 21-25, check the T (true) or F (false) column.

1. Which of the following indicates the main purpose of verification?
    a. Make sure that the right product is being built
    b. Make sure that the product is being built right
    c. Make sure that the product delivery is not delayed
    d. Make sure that the budget is not overrun

2. Which concept below expresses the overall size of a work item but is unitless and denotes only relative values?
    a. Ideal time
    b. Elapsed time
    c. Story point
    d. Velocity

3. Which type of testing between the two below is **more effective** to detect missing-logic fault?
    a. Black-box testing
    b. White-box testing

4. Which type of the requirement is "the iTrust system shall be written in Java and JSP"?
    a. Functional requirements
    b. Non-functional requirements
    c. Constraints
    d. Security requirements

5. Which of the following best describes an example of a failure?
    a. A programmer makes a mistake and forgets to write a conditional check in his/her code
    b. The execution of a test case causes the violation of an assertion in the test case
    c. A static analysis tool detects that some code portions may have a possible deadlock

6. Which coverage below is also called branch coverage?
    a. decision coverage
    b. method coverage
    c. statement coverage
    d. condition coverage

7. The following requirement is a _____ requirement.
    The system development must use the SVN version control system.
    a. functional
    b. non-functional
    c. constraint

8. Which of the following is <u>not</u> a stakeholder of the iTrust system?
    a. Developers of the iTrust system
    b. Testers of the iTrust system
    c. Developers of the OpenEMR system, an open source medical record application
    d. Doctors who are using the iTrust system in their hospital

9. Which statement expresses the essence of when the State pattern should be used?
    a. Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and the algorithm changes automatically.
    b. Encapsulate a request as an object, thereby letting you parameterize clients with different requests.
    c. Define a family of algorithms, encapsulate each one, and make them interchangeable so that the algorithm can vary independently from the clients that use it.
    d. Represent an operation to be performed on the elements of an object structure; allow the definition of new operations without changing the classes of the elements on which it operates.
    e. Allows an object to change its behavior in response to internal state changes.

10. Which statement best expresses the purpose of test driver?
    a. Invoked by a module under test
    b. Specify the expected output for a specified input of a test
    c. Invoke a module under test
    d. Simulate the environment that a module under test interacts with

11. Which of the following is an aspect that software configuration management (SCM) is **NOT** concerned with:
    a. Tracking and meeting release deadlines
    b. Keeping track of code changes
    c. Keeping track of reported errors and their status
    d. Managing external library dependencies needed to build your system

12. . Which one of the following is **NOT** correct regarding version control?
    a. If two people change the same line, the version control system may force the one merging the changes to perform the merge manually.
    b. When developing with a version control system, the developers should break their work into small steps and merge frequently.
    c. One good reason to use branch is to fix a bug across many versions of the application.
    d. **Unlike manual merge, automatic merge by the version control system ensures that no bugs will be introduced during merging.**

13. Which of the following is **NOT** a benefit of pair programming
    a. Higher quality of code with fewer defects is produced
    b. Staff training and transition are eased
    c. Programmers have a more enjoyable time
    d. A pair completes their tasks in less than half the time of a solo completing the same tasks

14. Which one of the following is **NOT** a key element in bug reporting?
    a. Check reproducibility of the failure as part of writing a bug report
    b. Document a crisp sequence of actions that will reproduce the failure
    c. **Change multiple variables that may alter symptom at the same time to speed up the debug process**
    d. Look for related failures in SUT

15. Which one of the following in NOT one of the big ideas of eXtreme Programming?
    a. Use working code as the main written product
    b. Design software architecture carefully before the coding phase
    c. Release code frequently
    d. Work closely with the customer

16. You just joined a team developing a large and complex software system, and you don't know how to start. An experienced developer in the team suggests you to "**Read all the Code in One Hour**". What does she most likely mean that you should do?
    a. Be prepared that the system is quite large, so it will take you some time to learn it
    b. Challenge you to demonstrate that you can read code very fast and spot problems
    c. Assess the state of the system by means of a brief, but intensive code review
    d. Learn the code base on your own and as quickly as possible

17. Which of the following is true about refactoring?
    a. Refactoring is a creational design pattern
    b. Refactoring is changing of the structure of a program without changing its behavior
    c. **Refactoring is changing the structure of a program while fixing bugs**
    d. Refactoring is an optimization applied to a slow piece of code to make it faster

18. Which of the following is typically **NOT** a code smell?
    a. Duplicated code
    b. Large parameter list
    c. **Many classes**

d. Long method

19. Software metrics are **NOT** used for which of the following:
    a. Prioritize work
    b. Measure programmer productivity
    c. Make plans based on predicted effort
    d. **Measure computational complexity of an algorithm**

20. Which of the following is **NOT** a standard for component/object interfacing
    a. CORBA
    b. COM
    c. DCOM
    d. JSON
    e. **JSP**

True/False (2 points each)

For each of the sentences below, mark an X in the appropriate box on the answer sheet. Naturally, mark an X in the T box if the statement is true and an X in the F box if the statement is **false**.

21. Multiple use cases are often combined to form a single user story.

22. Functional/system testing is a type of black-box testing.

23. For the following method, if test cases can achieve 100% statement coverage, the same set of test cases can definitely achieve 100% branch coverage:

```
public int method (int x) {
   if (x > 10) {
      x = x + 3;
   } else {
      x = x + 4;
   }
   return x;
}
```

24. A use case describes what a system does when interacting with an actor and how the system performs the functionality.

25. Risk Management is an evaluation of the current software development process to identify errors and inefficiencies.

White Box Testing [10 Points]

Consider the following method and answer the **two** questions below (the integers at the beginning of executable statements indicate the line numbers for those executable statements):

```
    int cp(int temperature, Boolean hasHeadache, int age) {
1       int type = 0;
2       if ((temperature >= 100) || (age > 60)) {
3           type++;
        }
4       if (hasHeadache) {
5           type = type + 2;
        }
6       return type;
    }
```

1. Write JUnit-style asserts for test cases (together with the already provided first test case as below) for covering **all branches AND all conditions**.
   Note: write each newly covered branches and conditions followed by the JUnit-style assert such as `3F, …: assertEquals(creditLimit (…), …)` where you should replace … with the actual test input, or expected output. A newly covered branch by the test case should be denoted with the line number postfixed by (T) or (F) indicating True or False, respectively. A newly covered condition should be denoted with the condition expression postfixed by (T) or (F) indicating True or False, respectively. Below we already list the first test case. (3 points)

   2(T), 4(F)
   temperature >= 100(T), hasHeadache(F)
   assertEquals(1, cp(120, false, 30))

Answer: There can be multiple ways of writing these test cases (different orders or different test cases). Below are one such example test case sequence.

4(T)
hasHeadache(T)
assertEquals(3, cp (120, true, 30))

2(F)
temperature >= 100(F), age > 60(F)
assertEquals(2, cp (90, true, 30))

age > 60(T)
assertEquals(3, cp (90, true, 70))

Grading guide: each entry 2.5 point; There are different ways of writing test cases; but all branches and all conditions shall be covered by these test cases together with the provided one.

2. Write one JUnit-style assert for a test case to cover a boundary value of a predicate in Statement 2 (2.5 points)

Answer: Example solution: assertEquals(cp(100, false, 30), 1)
Requirement: temperature is either 100, 99, or 101, or age is 60 or 61. Note that some 1 or 2 off boundary values are also fine.

Black Box [10 Points]

Write <u>five</u> black box test cases to test the functionality of the component outlined in the problem statement below. Write these test cases by completing the black box test case template. Mark the strategy for each test case as being either an equivalence class (EC), a boundary value analysis (BVA), or a diabolical (DT) test case.

The component is to identify a performance issue. The input to the component includes the number of seconds and the type (as time-sensitive or time-insensitive). The output of the component is an integer flag to indicate whether there is a performance issue (0 indicates no performance issue and 1 indicates a performance issue). The component first classifies the case to be slow if the number of seconds is greater than 10. The component outputs a flag of 1 if and only if the case is both slow and of the time-sensitive type; otherwise, the component outputs a flag of 0.

Note: You don't need to fill in the actual result column.

Test ID	Description	Expected Results	Actual Result	Strategy
SenNotIssueBound1	Seconds = 10 Type = time-sensitive	0	DON'T Fill Anything Here	BVA
SenIssueBound1	Seconds = 11 Type = time-sensitive	1	DON'T Fill Anything Here	BVA
InSenNotIssueBound1	Seconds = 10 Type = time-insensitive	0	DON'T Fill Anything Here	BVA
InSenNotIssue1	Seconds = 15 Type = time-insensitive	0	DON'T Fill Anything Here	EC
SenNotIssue1	Seconds = 4 Type = time-sensitive	0	DON'T Fill Anything Here	EC

[There are just a sample of the possible test cases. – 3 points per test case]
5 test cases, 1 point each for unique id& strategy, description, expected results. Description must be repeatable and specific.

1. **Explain why black box testing is needed after white box testing is conducted (2.5 points)**

2. **Explain why white box testing is needed after black box testing is conducted (2.5 points)**

Class Invariants [5 Points]

Consider the class

```
class ValueRange {
  int lower = Math.minInt();
  int upper = Math.maxInt();

  // what's the class invariant?

  void setLower(int i) { if (i < upper) lower = i; }
  void setUpper(int i) { if (i > lower) upper = i; }
}
```

Give three class invariants Inv1, Inv2 and Inv3 for `ValueRange` such as all the following constraints hold:

Inv1 logically implies Inv3
Inv2 logically implies Inv3
Inv1 does not logically imply Inv2
Inv2 does not logically imply Inv1

Clarification: "A logically implies B" means that if A is true then B is also true. For example, "x=7 implies x>0", and "(x integer and x<10 and x>7) implies (x=8 or x=9)".

Answer: Inv1 is "Math.minInt() < upper"
Inv2 is "lower < Math.maxint()"
Inv3 is "true"

Design Patterns [12 Points]

One popular feature on YouTube is to allow users to subscribe and unsubscribe to a channel. Whenever there is a new video published on to the channel, all the subscribers are informed of this new video.
In this question, we will use object-oriented design to model a simplified version of this Channel-User relationship. Here are the requirements:
1. There are two classes, Channel and User.
2. When a user subscribe to a Channel, the user becomes a subscriber.
3. A Channel can be subscribed and unsubscribed by calling its subscribe() and unsubscribe() methods.
4. A Channel maintains a list of all the subscribers.
5. A User maintains a list, called videoList, which contains the titles of all the videos from all the channels the user has subscribed to.
6. A Channel has a notifyNewVideo() method. When this method is called with the title of the new video as parameter, all its subscribers should add this title to their videoList.

Assumptions: You don't have to load a channel's videos before users subscribe to it. A user will not subscribe to a channel twice. There are no duplicate videos. All necessary packages have been imported. All the fields in Channel and User classes will be correctly initialized by their constructors.

You are asked to do the following 4 questions:

1. Read the comments and add code below to fulfil the requirements. (7 points)

```
//in Channel.java

public class Channel{
     //This list contains all the subscribers of the channel
     List<User> subscribers;

     public Channel() {
       //Assume it does what it is supposed to do; no need to code this
       ...
     }

     /**
     * Interface for subscribing to this channel
     * @param user the user to subscribe
     */
     public void subscribe(User user) {
       //Add code here; hint: one method call to be added below
       subscribers.add(user);//answer; remove this line for actual exam


     }

     /**
     * Interface for unsubscribing from this channel
     * @param user the user to unsubscribe
     */
     public void unsubscribe(User user) {
       //Add code here: one method call to be added below
        subscribers.remove(user); //answer; remove this line for actual exam



     }
```

```
	/**
	* Inform all subscribers of a newly published video on this channel
	* @param videoTitle the title of this new video
	*/
	public notifyNewVideo(String videoTitle) {
	   //Add code here
	   for(User user : subscribers) //answer; remove this line for actual exam
	      user.addNewVideo(videoTitle); //remove this line for actual exam



	}

}


//in User.java

public class User{
	//List of titles of all the videos from all the channels
	List<String> videoList;

	public User() {
	 // Assume it does what it is supposed to do; no need to code this
	  ...
	}

	/**
	* Add a video into the videoList
	* @param videoTitle the title of the new video to be added
	*/
	public void addNewVideo(String videoTitle) {
	  //Add code Here; hint: one method call to be added below
	  videoList.add(videoTitle); //remove this line for actual exam


	}

}
```

2. What design pattern have you used above, if any? (3 points)

Observer Pattern

3. Besides title, we want to add the name of the uploader as another metadata. Define a bean called VideoBean to store these metadata. This class is subject to the same requirements for all the beans in iTrust project. You need to include all the variables and methods needed in this class, with understandable names. For methods, you only need to write the method signature and then write {...} to represent the method body. You do not need to override any method inherited from java.lang.Object class. (7 points)

```
//in VideoBean.java
public class VideoBean {









}
```

Answers for Questions 3:

```
class VideoBean {
            String uploader, title;

            public VideoBean(){...}
            public VideoBean(String author,String title){...}
            public void setUploader(String author){...}
            public String getUploader(){...}
            public void setTitle(String title){...}
            public String getTitle(){...}

}
```

Refactoring [16 Points]

Consider the following five Java classes (do not worry about the formatting):

```
public class DomainObject {
      public DomainObject (String name) { _name = name; };
      public DomainObject() {};
      public String name () { return _name; };
      public String toString() { return _name; };
      protected String _name = "no name";
}

public class Movie extends DomainObject {
     public static final int  CHILDRENS = 2;
     public static final int  REGULAR = 0;
     public static final int  NEW_RELEASE = 1;
     private int _priceCode;

      public Movie(String name, int priceCode) {
            _name = name; _priceCode = priceCode;
      }

      public int priceCode() { return _priceCode; }

      public void persist() { Registrar.add ("Movies", this); }

      public static Movie get(String name) {
            return (Movie) Registrar.get ("Movies", name);
      }
}

class Tape extends DomainObject
    {
    public Movie movie() { return _movie; }
    public Tape(String serialNumber, Movie movie) {
      _serialNumber = serialNumber;
      _movie = movie;
    }
    private String _serialNumber;
    private Movie _movie;
    }

class Rental extends DomainObject
    {
    public int daysRented() { return _daysRented; }
    public Tape tape() { return _tape; }
    private Tape _tape;
    public Rental(Tape tape, int daysRented) {
      _tape = tape;
      _daysRented = daysRented;
    }
    private int _daysRented;
    }
```

```
class Customer extends DomainObject
    {
    public Customer(String name) { _name = name; }

    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + name() + "\n";
        while (rentals.hasMoreElements()) {
            double thisAmount = 0;
            Rental each = (Rental) rentals.nextElement();

            //determine amounts for each line
            switch (each.tape().movie().priceCode()) {
                case Movie.REGULAR:
                    thisAmount += 2;
                    if (each.daysRented() > 2)
                        thisAmount += (each.daysRented() - 2) * 1.5;
                    break;
                case Movie.NEW_RELEASE:
                    thisAmount += each.daysRented() * 3;
                    break;
                case Movie.CHILDRENS:
                    thisAmount += 1.5;
                    if (each.daysRented() > 3)
                        thisAmount += (each.daysRented() - 3) * 1.5;
                    break;

            }
            totalAmount += thisAmount;

            // add frequent renter points
            frequentRenterPoints ++;
            // add bonus for a two day new release rental
            if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;

            //show figures for this rental
            result += "\t" + each.tape().movie().name()+ "\t" +
String.valueOf(thisAmount) + "\n";

        }
        //add footer lines
        result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
        result += "You earned " + String.valueOf(frequentRenterPoints) + "
frequent renter points";
        return result;

    }

    public void addRental(Rental arg) { _rentals.addElement(arg); }
    public static Customer get(String name) {
```

```
  return (Customer) Registrar.get("Customers", name);
}
public void persist() { Registrar.add("Customers", this); }
private Vector _rentals = new Vector();
}
```

Explain what refactorings are desirable for the code in the `statement()` method above. For each refactoring, state its name (if you don't remember the exact name then invent one and explain it briefly), show where it appears in the code (mark the code using your pen), and then sketch the new code below.

(Continue on this page if needed; use the back of these last four pages if more space needed)