0. Name (2 pts):

CS 2334: Programming Structures and Abstractions

Final Exam

Solution Set

Thursday, December 17, 2009

Problem	Topic	Max	Grade
0	Name	2	
1	Recursion	35	
2	Ethics	40	
3	Exceptions and Assertions	30	
4	Inheritance	25	
5	Generic Programming and Generics	20	
6	Graphical User Interfaces	20	
7	Event-Driven Programming and MVC	10	
8	Collections Framework	20	
Total			

1. **Recursion** (35 pts)

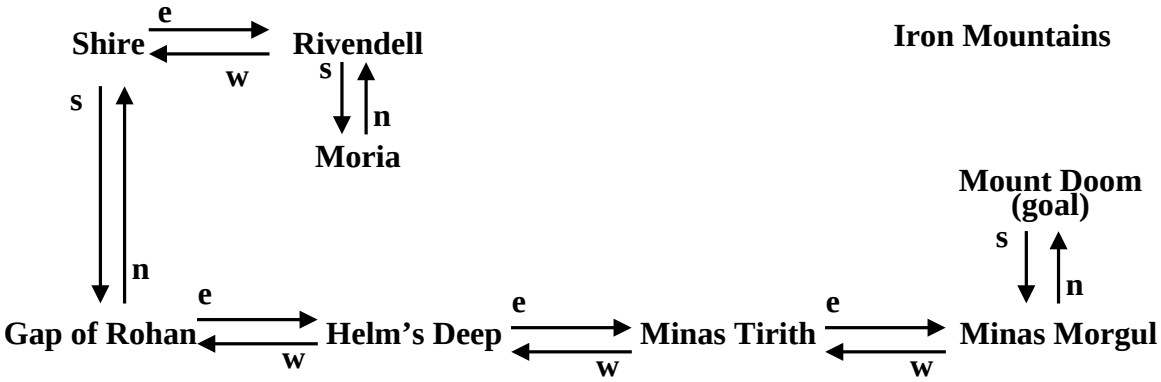
Consider the following class definition:

```
public class Place {
    public boolean goal;     // Is this place a goal?
    public Place north;      // Place to the North (null -> no place)
    public Place south;
    public Place east;
    public Place west;



};
```

We will use this class to represent places on a map. For example, consider the following map:



Here, each of the arrows represent references from one place to another (with n,s,e,w corresponding to North, South, East and West). When no arrow is given for a particular direction, assume that the reference is null.

We will design a method:

```
public boolean isConnected(Place p);
```

that will return **true** if the specified place is somehow connected to a goal place, and **false** otherwise. For example, isConnected(Shire) would return **true** (because Mount Doom is a goal), but isConnected(Iron Mountains) would return **false**. As you answer the following questions, state any other assumptions that you must make.

(a) (5 pts) List the base case(s).

Assume: the Place class also has a boolean property called **visited**. Before the method is called, all visited properties are set to false.

p is null → answer is false

p.visited is true → answer is false

p.goal is true → answer is true

(b) (5 pts) List the recursive case(s).

All cases that are not a base case fall into a single case: true if any of the four directions are true.

(c) (5 pts) List the preconditions for isConnected().

p is a valid place.

(d) (5 pts) List the postconditions for isConnected().

Returns true if **p** is the goal or if there exists a non-visited path from **p** to the goal. Otherwise, false is returned.

(e) (15 pts) Give the pseudocode for the method.

```
public boolean isConnected(Place p)
{
    if(p == null) return false;
    if(p.visited == true) return false;
    if(p.goal == true) return true;
    p.visited = true;
    return (isConnected(p.north) ||
            isConnected(p.south) ||
            isConnected(p.east) ||
            isConnected(p.west));
}
```

2. **Ethics in Computer Science** (40 pts)

(a) (10 pts) True or False and briefly explain: if privacy is a fundamental human right, an individual has the right to sell his/her personal data (thus waiving the right to further control these data).

I will accept arguments in either direction. Here is one line of thinking: according to the book, fundamental human rights are rights that always always exist and cannot be removed – by law or by individual choice. So, if we recognize that privacy is a fundamental human right and that dispersion of personal data is a violation of privacy, then any dispersion of personal data violates this (including the sale by the individual).

By this line of reasoning, the answer is False.

(b) (15 pts) Researchers at the recent International Conference on Data Mining (ICDM'09) presented work that will allow stores, using video cameras, to track the movements of single customers. Specifically, the marketing team will have access to a time-indexed path (a trajectory) that the customer took during the visit – from entering the store to the checkout line. By seeing where customers tend to go and how long they spend at certain store displays, the team will be able to better plan the layout of the store and the design of displays, so as to improve sales.

What are the key issues and ethical principles that should be considered in the design and deployment of this system?

- Privacy: depending on how the system is implement, the individuals that are being tracked could potentially be uniquely identified. For example, the system may store pictures of the faces of the individuals or the system may even be able to use the time of check out coupled with a log of credit card transactions.
- Technological opacity: these data may be collected without the knowledge (or consent) of the customers.
- Sharing of information: these data could be shared with other entities – with or without the consent of the individuals. Also, there may not be a way for the customers to know or control how this information is used by these third parties.

(c) (15 pts) Briefly explain why the following claim is "muddled" from an ethical and legal perspective: software is property.

We have traditionally thought of property as being physical *things*. Replication of

this type of property is often nontrivial and its use can be physically controlled by an owner. Intellectual property law attempts to protect nonphysical property. Because a creator exerts effort in creating the software, it is reasonable to expect that the creator should have some notion of ownership (including transfer of that ownership). Because copyright law is about protecting written work, it is often used to protect this ownership in the case of software. However, in order to receive protection, the work has to be in a *fixed form*. It is rarely the case that software is fixed in form – the process of fixing bugs and adding features can be an ever continuing one.

Software (like any digital entity) is exceedingly easy and inexpensive to copy and modify. This is different from the traditional notions of property, and even from notions of many forms of intellectual property.

3. **Exceptions and Assertions** (30 pts)

Consider the following code:

```
public class exceptionTest
{
    public void methodA() { ...    // Do some stuff
    };

    public void methodB() { ...    // Do some other stuff
    };

    public void bar() {
        try {
            System.out.println("Foo 5");
            methodB();
            System.out.println("Foo 6");
        }catch(ExceptionB e){
            System.out.println("Foo 7");
            throw e;
        };
        System.out.println("Foo 8");
    }

    static public void main(String[] args) {
        try {
            System.out.println("Foo 1");
            bar();
            methodA();
            System.out.println("Foo 2");
        }catch(ExceptionA e){
            System.out.println("Foo 3");
        };
        System.out.println("Foo 4");
    };
}
```

(a) (10 pts) Suppose that methodB() throws ExceptionB during execution. What output does this program produce and does the program terminate with or without an error? (assume that output is only produced by the println() calls that you see)

Foo 1
Foo 5
Foo 7
(program terminates with ExceptionB)

(b) (10 pts) Suppose that methodB() throws ExceptionA during execution. What output does this program produce and does the program terminate with or without an error?

Foo 1
Foo 5
Foo 3
Foo 4
(program terminates normally)

(c) (10 pts) Briefly describe the similarities and differences between Exceptions and Assertions.

Assertions are a form of exception (more specifically, a form of error). So: they behave as exceptions.

Assertions are for testing code internally and are not intended as a means of checking that other people are using your code properly. In addition, assertions can be disabled at run time.

4. **Inheritance** (25 pts)

Consider the following implementation:

```
public class A
{
    protected String name;

    public A(String name){
        this.name = name;
    }

    public String toString(){
        return("A: " + name);
    };
}

public class B extends A
{
    protected String name;

    public B(String name) {
        super("SUPER-B");
        this.name = name;
    }
}

public class C extends B
{
    private String name;

    public C(String name){
        super("SUPER-C");
        this.name = name;
    };

    public String toString() {
        return("C: " + name);
    };
};

public class driver
{
    public static void main(String args[]) {
    A[] objects = new A[4];

    objects[0] = new A("foo");
    objects[1] = new B("bar");
    C c = new C("baz");
    objects[2] = c;

    for(int i = 0; i < objects.length; ++i) {
        System.out.println(objects[i]);
    };

    B b = c;

    System.out.println(b);
    System.out.println(c);
    };
};
```

(a) (20 pts) What output does this program produce?

```
A: foo
A: SUPER-B
C: baz
null
C: baz
C: baz
```

(b) (5 pts) Could class A be implemented as an *interface*? Why or why not?

No. It contains both properties and concrete methods.

5. **Generic Programming and Generics** (20 pts)

(a) (5 pts) T/F: Generics are enforced at run-time.

False. (they are only enforced at compile time)

(b) (15 pts) Consider the following block of code:

```
public void main(String[] args)
{

   Comparator<B> comp = getComparator();

   A a = new A("A element");
   B b = new B("B element");
   C c = new C("C element");

   comp.compare(a, b);     // Will it compile?

   comp.compare(a, c);     // Will it compile?

   comp.compare(b, c);     // Will it compile?

   comp.equalsTo(b, c);    // Will it compile?
}
```

Assume that getComparator() returns a proper Comparator of the declared type and that this object implements just the methods required by the Comparator interface. Assume that classes A, B, and C are the same as from the previous question. State whether each of the four indicated method calls will compile or not.

Compares involving object **a** will not compile since they are not instances of class B. So - the first two calls will not compile. The last compare will compile because both **b** and **c** are instances of class B. The last will not compile because it is not required by the Comparator interface (and is therefore not provided by **comp**).

6. **Graphical User Interfaces** (20 pts)

(a) (10 pts) What information would the JButton class use in its implementation of the getPreferredSize() method? (we did not explicitly talk about this in class, but you should be able to speculate on some of the details).

The button potentially contains text (string and font matter here), and an icon (size matters). In addition, the layout of the button would matter.

(b) (10 pts) True or False and briefly explain: it is possible **and** useful for a single ActionEventListener object to be registered with multiple buttons.

True. This could be useful if very similar actions are taken for a set of buttons. The listener can tell which button was pressed because ActionEvent contains a reference to this source object.

7. **Event-Driven Programming and the Model View Controller Approach** (10 pts)

   (10 pts) Briefly explain the tension between the pure form of the Model View Controller approach and what is often implemented.

   The pure MVC approach explicitly separates each of these components, with the viewer only pulling data out of the model and the controller only changing the model. With things such as GUIs, however, the GUI will do both (and hence we often merge the concept of Viewer and Controller).

8. **Java Collections Framework** (20 pts)

Consider the following code:

```
import java.util.*;

public class collectionTest
{
    public static void main(String[] args)  {
    Set<Integer> col1 = new TreeSet<Integer>();
    col1.add(9);
    col1.add(3);
    col1.add(19);
    col1.add(11);
    col1.add(3);

    List<String> col2 = new LinkedList<String>();
    col2.add("Mirkwood");
    col2.add("Fangorn");
    col2.add("Pinnath Gelin");
    col2.add("Fangorn");
    col2.add("Blue Mountains");

    for(Object element: col1) {
        System.out.println(element.toString());
    };

    Iterator it = col2.listIterator();

    while(it.hasNext()) {
        System.out.println(it.next().toString());
    };
  }
}
```

(20 pts) What output does the program produce?

```
3
9
11
19
Mirkwood
Fangorn
Pinnath Gelin
Fangorn
Blue Mountains
```