Fall 2011 Software Engineering Qualifying Exam

This is an open book exam. Basic calculators are allowed, but no computers or other devices that have communication capability are allowed; this means that cell phones cannot be used as calculators. There are a total of 100 points on this exam. Be sure to show your work in case your answer may deserve partial credit, but do not add spurious or frivolous content in hopes that something you say might be right. Content in an answer that is irrelevant to the problem may cause point deductions.

[40pts] 1. UML and Modeling

For the parts in this section, use the following problem description.

We need to build an integrated information and inventory management system for an automobile parts store. The store stocks and sells auto parts. To be able to keep track of parts (and find them for customers), the system also needs information about automobiles. Automobiles are classified into various types: trucks, sedans, and hatchbacks. Trucks have both a weight and volume capacity, sedans have a trunk volume capacity, and hatchbacks have a cargo capacity. All vehicles have a manufacturer, a model name and number, and a year. All vehicles are made up of many components, all of which we call parts. Some parts are atomic (indivisible) parts, while others are composite parts that are themselves made up of parts. The store sells both kinds of parts. Vehicles can also have optional components. Parts have a manufacturer, a part number, a name and description. Information from the manufacturer on a part also includes a list of equivalent parts. For example, a Delphi VR321 headlight might be equivalent to a GMC RE91 headlight.

One of the functionalities that the system will have to support is a customer kiosk, where a customer can walk in, click on or enter their vehicle make, model, etc., and zoom down through composite components to get to the part they need, then pull up a list of all equivalent parts, their prices, and their availability (how many are in stock in the store).

[25pts] A. Draw a problem-domain UML class diagram for this problem, including classes and their attributes, and associations between classes. Where appropriate be sure to indicate name, direction, and cardinality for associations.

> *Answers will vary. The entities in the problem should be captured, with their relationships.*

[5pts] B. Would building this system be a high-risk or a low-risk project? Explain your answer.

> *Answers will vary. The explanation is the important part. The information structure in automobiles is well-known, as should be basic inventory needs of a store. So it should be fairly low risk.*

[5pts] C. Are their any design patterns you would use in building this project? Explain your answer.

> *Answers will vary. Most obvious is the Composite pattern.*

[5pts] D. What risks would their be in adding a customer web interface to the system, in addition to the in-store kiosk? Explain your answer, including what you would do to minimize those risks.

> *Answers will vary. Security is the most obvious concern.*

[60pts] 2. Program Analysis and Verification

The following questions deal with the program shown below, and with the sample execution shown below the program. The class is meant to implement a hash table, and the sample main() program should print out each of the command line arguments with an "item: " in front of it, and nothing else.

```
public class HashTable
{
private final int MAX_ELEMS = 20;
private int /*! spec_public !*/ numElements;
private int /*! spec_public !*/ nextI;
private Object myData[];

public HashTable()
{
   numElements = 0;
   nextI = -1;
   myData = new Object[MAX_ELEMS];
}

private int hash(String s)
{
   int h=0;
   int i=0;
   while (i < s.length()) {
      h += s.charAt(i++);
   }
   return h % MAX_ELEMS;
}

public void insert(Object val)
{
   int index = hash(val.toString());
   while (myData[index] != null)
      index = (index+1) % MAX_ELEMS;
   myData[index] = val;
   numElements++;
}

public void start()
{
   nextI = 0;
}

public boolean hasNext()
{
   if (nextI >= 0) return true;
   return false;
}

public Object next()
{
   while (nextI < MAX_ELEMS && myData[nextI]==null)
      nextI++;
   if (nextI >= MAX_ELEMS) {
      nextI = -1;
      return null;
   }
   return myData[nextI++];
}

public static void main(String args[])
{
   int i=0;
   HashTable h = new HashTable();
   while (i < args.length) {
      h.insert(args[i++]);
   }
   h.start();
   while (h.hasNext()) {
      System.out.println("item: " + h.next());
   }
}

};
```

A sample run of the program is:

```
shell> java HashTable hello there and goodbye
item: goodbye
item: and
item: hello
item: there
item: null
```

[15pts] A. Write JML *requires* and *assures* clauses for the *insert()* and *next()* methods, using your knowledge of how a hash table such as the one implemented ought to behave (they may or may not be implemented correctly). In your JML annotations you should capture the ``intent'' of the operation.

For insert, requires should say that the table is not full and the value is not null, and ensures should say that the table has one index that references the value. For next, requires should say nextI is within bounds, and ensures should say that the return value is not null;

[5pts] B. The execution above shows that there is an error in the class; yet it is incorrect to state that

next() should never return null. Under what conditions should it return null? Under what conditions should it never return null?

> *If hasNext() returns false, then next() should return null; otherwise it should return a valid object reference. Essentially, hasNext() should not lie.*

[8pts] C. If you can, write JML annotations that capture your answer to (B). If you cannot, explain what is required that is hard or impossible to write within JML's framework.

> *The conditions in (B) require annotations that relate the execution of two different methods, but JML's annotation approach is method-by-method. This makes it hard to capture conditions like these. But it is not impossible. JML allows "ghost fields" which are additional object fields that can hold information solely used by JML. The result of the last call to hasNext() could be stored in a ghost field, and then used in the ensures clauses for the next() method.*

[5pts] D. A coverage tool tells us that the execution above did not execute line 30. What type of test case would you have to construct to achieve coverage of line 30?

> *Line 30 implements linear collision resolution for the hash table. A test case would have to have a collision, so it would have to have two strings that hashed to the same location in the table.*

[10pts] E. For the object field variable *nextI*, write down all of the definitions and all of the uses, using line numbers (add a letter for multiple definitions or uses on one line). Then for only the definitions that occur in the *next()* method, write down every def-use pair and classify it into one of the scopes defined in the paper "Dataflow Testing of Classes". The scopes are: intra-method, inter-method, and intra-class.

> *Defs: 12, 37, 49, 51, 54*
> *Uses: 42, 48a, 48b, 49, 50, 54*

DU pair	Class	DU pair	Class	DU pair	Class
49-42	intra-class	51-42	intra-class	54-42	intra-class
49-48a	intra-method	51-48a	intra-class	54-48a	intra-class
49-48b	intra-method	51-48b	intra-class	54-48b	intra-class
49-49	intra-method	51-49	intra-class	54-49	intra-class
49-50	intra-method	51-50	intra-class	54-50	intra-class
49-54	intra-method	51-54	intra-method	54-54	intra-class

[7pts] F. Which of the def-use pairs do you know were NOT exercised in the execution shown above? Would it be possible to provide a test case to this program that would exercise them? Would it be possible to write a program that used this class (or to simply re-write *main()*) that exercised them?

> *The def-use's (12-48b) and (51-48b) are definitely not exercised because they would cause an array out of bounds exception. It is not possible in the program as give to test these, because the driver in main properly calls hasNext() before invoking next(), and it will return false if nextI is less than 0. But yes it would be possible to re-write the driver so that these were exercised.*

[10pts] G. Define and describe soundness and completeness in the context of program verification. The tool ESCJava claimed that it was neither sound nor complete. Give examples of why it fails on both properties. Finally, if the tool is not sound nor complete, why is it valuable at all?

Soundness says that if a tool says a property holds, then it really does. Completeness says that a tool can detect all properties that hold (i.e., if it says the property does not hold, it really does not). One source of unsoundness in ESC is the way it analyzes loops. It only analyzes a fixed number of iterations, so if the property holds for that number, ESC will say yes, even though it might be possible that the property fails if the loop is executed more times. On source of incompleteness in ESC is the fact that it uses modular checking (method by method); thus it might warn that a property is not satisfied because a method might do something bad, when in fact the ways that the method is always called in the whole program, that bad thing cannot happen.