

Spring 2009 Software Engineering Ph.D. Qualifying Exam

This exam is open book. There are 100 points on the exam, with each problem noting its point value. Be sure to read questions carefully and answer **all parts** to the question.

[40pts] 1. UML

Consider the problem of a university department keeping track of its students, faculty, and courses. All people have identifying information like name, phone number, etc. Faculty have a title (such as "associate professor") and an office. Students are either undergraduate students or graduate students; grad students also have an office and are either M.S. or Ph.D. students. Courses are either undergraduate courses or graduate courses. The complete set of courses in a department are not always offered each semester, so a particular course offering is a given course in a given semester. Students enroll for course offerings, and eventually get a grade for the course. Faculty teach the course offerings, and graduate students can be assigned as assistants to a course offering. Even after a semester has ended, all the information about which courses were offered for that semester, including the faculty who taught them and the students who took them, must be maintained.

[25pts] A) Draw a problem domain UML class diagram for this problem, being sure to provide proper labeling as needed, and having direction, cardinality, and modality on associations that need it. Classes should have appropriate attributes but do not need to have methods.

[15pts] B) Describe at least three different ambiguities in the problem description, and decide specific resolutions for each of the ambiguities that you point out.

Below is some annotated Java code that will be used for several problems. The code is self-explanatory. Thread.sleep() sleeps a thread for the specified number of milliseconds, and Math.random() returns a double value between 0.0 and 1.0. A call to join() on a thread waits for the thread to finish.

All of the try-catch blocks in the code are there solely to allow clean compilation of code; you can assume that no executions of the code in the try-catch blocks ever cause any exceptions.

```

00:
01: class T1 extends Thread
02: {
03:     public void run()
04:     {
05:         /*@ assume Fun.A != null && Fun.A.length >= 10;
06:         for (int i=0; i<10; i++)
07:         {
08:             Fun.A[i] = i-10;
09:             try {
10:                 Thread.sleep(Math.round(Math.random()*2));
11:             } catch (Exception e) {}
12:         }
13:     }
14: }
15:
16: class T2 extends Thread
17: {
18:     public void run()
19:     {
20:         /*@ assume Fun.A != null && Fun.A.length >= 10;
21:         for (int i=0; i<10; i++)
22:         {
23:             Fun.A[i] = i+10;
24:             try {
25:                 Thread.sleep(Math.round(Math.random()*3));
26:             } catch (Exception e) {}
27:         }
28:     }
29: }
30:
31: public class Fun
32: {
33:
34:     /*@ nullable @*/ public static int A[];
35:
36:     /*@ invariant A==null || (A.length >=10 &&
37:     (\forallall int i; 0 <= i && i <= 10; A[i] >= 0));
38:     @*/
39:
40:     public static void main(String args[])
41:     {
42:         A = new int[10];
43:         T1 t1 = new T1();
44:         T2 t2 = new T2();
45:         t1.start();
46:         t2.start();
47:         try {
48:             t1.join();
49:             t2.join();
50:         } catch (Exception e) {}
51:         for (int i=0; i<10; i++)
52:         {
53:             /*@ assert A[i] >= 0;
54:             System.out.println("A["+i+"] = "+A[i]);
55:         }
56:         return;
57:     }
58: }

```

[20pts] 2. JML

JML is a tool that can help provide some assurance that a program is working correctly.

The above program was compiled with the JML compiler "jmlc" and then ran twice with the JML runtime "jmlrac". Both times it ran successfully and printed out:

```
A[0] = 10
A[1] = 11
A[2] = 12
A[3] = 13
A[4] = 14
A[5] = 15
A[6] = 16
A[7] = 17
A[8] = 18
A[9] = 19
```

[5pts] A) What do the JML tools "jmlc" and "jmlrac" do with the annotations on the program?

[8pts] B) Given the results above (which show the program working), why are we **not** able to conclude then that the program is correct in relation to the specifications in the annotations?

[7pts] C) Indeed, the program is **not correct** with respect to the annotations in the program. Explain how it can violate the annotations.

[20pts] 3. ESC/Java

[5pts] A) ESC claims that it does *static modular* checking. Explain what this means (including both *static* and *modular*).

[5pts] B) ESC admits that its analysis is *unsound*. Explain what unsoundness is, give at least one example source of unsoundness in ESC, and explain why ESC is still a powerful analysis tool.

[5pts] C) When ESC analyses the Fun class, it passes the main() method with no warnings, yet the assert statement in the loop **can** fail. Why does ESC not give any warnings? (hint: the answer is related to part A.)

[5pts] D) When ESC analyses the T1 class, it does give a warning, shown below:

```
Fun.java:08: Warning: Possible violation of object invariant (Invariant)
Associated declaration is "Fun.java", line 37, col 5:
/*@ invariant A==null || (A.length >=10 && ...
```

Explain this warning.

[20pts] 4. Testing

[10pts] A) Write down all the def-use pairs for the elements in array A. Mark which ones are actually exercised in the two runs described at the beginning of problem 2.

[10pts] B) Explain why testing concurrent programs is particularly difficult, and how you would try to test this program so that you were sure that all def-use pairs identified in part A were actually exercised in your test cases.