Spring 2012 Course: CS 498, Section DM

Software Testing

Problem Set 4
Assigned: April 9, 2012
Due: Tuesday, April 17, 2012 (at 12:30pm, beginning of class)

This problem set covers the material from Chapter 4 of the textbook. There are five problems, worth total of 120 points. You need only 100 points to get the maximum score for this problem set, which accounts for 15% of the final grade. If you have more than 100 points, you will get credit for the future problem sets and the project. For the subproblems labeled **[MP]**, commit your code into SVN. For the non-MP problems, you can either include your solutions in SVN (if so, please use simple ASCII documents to write your solutions) or you can bring your (hand-written or printed) solutions to the lecture. **Important note:** you cannot collaborate on the first three problems, but (1) you can discuss Problem 4 and (2) members of each group should collaborate on Problem 5 without groups collaborating with one another!

Problem 1 [16 points]: (This is a modified version of Exercise **1** after Section **4.1**, page 159; the entire text is here.) Answer the following questions for the method **search** below:

```
public static int search(List list, Object e)
// Effects: If list or e is null, throw NullPointerException
//          else if the element e is in the list, return an index of e in the list; else return -1
//          for example, search([3,3,1], 3) returns either 0 or 1 while search([1,7,5], 2) returns -1
```

Base your answers on the following characteristic partitioning "Location of e in list":

Block 1: e is first entry in list
Block 2: e is last entry in list
Block 3: e is in some position other than first or last

(a) [4 points]: "Location of e in list" fails the disjointness property. Give an example that illustrates this.
(b) [4 points]: "Location of e in list" fails the completeness property. Give an example that shows this.
(c) [4 points]: Describe the input domain for the method **search**.
(d) [4 points]: Supply one or more new partitions that capture the intent of "Location of e in list" but do not suffer from any disjointness or completeness problems. Do NOT use only partitions with two blocks.

Problem 2 [44 points]: (This is a modified version of Exercise **4** after Section **4.2**, page 163; the entire text is here.) Answer the following questions for the method **intersection** below:

```
public static Set intersection(Set s1, Set s2)
// Effects: Return a (non null) Set equal to the intersection of sets s1 and s2
//          A null argument is treated as an empty set
```

Characteristic 1: Type of s1
Block 1: s1 = null
Block 2: s1 = {}
Block 3: s1 has at least one element

Characteristic 3: Relation between s1 and s2
Block 1: s1 and s2 represent the same set
Block 2: s1 is a subset of s2
Block 3: s2 is a subset of s1
Block 4: s1 and s2 do not have any elements in common

(a) [4 points]: Does the partition "Type of s1" satisfy the completeness property? If not, give a value for s1 that does not fit in any block. Does the partition "Type of s1" satisfy the disjointness property? If not, give a value for s1 that fits in more than one block.

(b) [4 points]: Does the partition "Relation between s1 and s2" satisfy the completeness property? If not, give a pair of values for s1 and s2 that does not fit in any block. Does the partition "Relation between s1 and s2" satisfy the disjointness property? If not, give a pair of values that fits in more than one block.
(c) [4 points]: Describe the input domain for the method **intersection**.
(d) [4 points]: Change the blocks for the partitions "Type of s1" and "Relation between s1 and s2" such that they do not suffer from any disjointness or completeness problems.
(e) [4 points]: Create a partition "Type of s2" analogous to "Type of s1". Choose a representative input for each block from the three partitions "Type of s1", "Type of s2", and "Relation between s1 and s2".
(f) [4 points]: Describe the constraints among the three partitions used above.

Construct test cases that obey the constraints among the three partitions and satisfy the following criteria:
(g) [4 points]: [MP] All Combinations
(h) [4 points]: [MP] Each Choice, but not Pair-wise
(i) [4 points]: [MP] Pair-wise, but not All Combinations
(j) [4 points]: [MP] 3-wise (Hint: consider if you can concisely answer this question.)
(k) [4 points]: [MP] Base Choice; discuss your pick for base choice
Each test case should have two input sets and an expected output. Write your tests in JUnit (search for **TODO** in SVN). Hint: Consider writing a program that will help you in writing tests in JUnit.

Problem 3 [20 points]: This problem considers the **TestPat** class from page 56, Chapter 2. (This is the same class from Problem 4 in Problem Set 2 and Problem 5 in Problem Set 3.)
(a) [4 points]: Describe the input domain for the method **main**.
(b) [4 points]: Describe the input domain for the method **pat**.
(c) [4 points]: Develop a partition for each argument of **pat** (**subject** and **pattern**) based on its type.
(d) [4 points]: Develop a partition for the relationship between the arguments **subject** and **pattern** based on the semantics of the **pat** method.
(e) [4 points]: Construct test inputs that satisfy "Base Choice" criteria for your partitions. You do **not** need to derive the expected outputs and can use the test cases from Table 2.5 on pages 59-60.

Problem 4 [20 points]: [MP] (*) Write a method that computes a (small, but not necessarily minimal) list of tuples that satisfy "Pair-wise" coverage for a given list of partitions. For more details, search for **TODO** in SVN. **(*)**This problem is **hard**; feel free to discuss it on the mailing list and to search online!

Problem 5 [20 points]: For some part of JPF (it can be the Config class), do the following:
(a) [4 points]: Choose some feature (e.g., some methods) and describe its input domain. There should be at least two parameters. Example parameters for Config include the following: some properties of the files (e.g., what directory they are in), what command-line arguments are passed in, what search strategy is used, what listeners are provided, etc.
(b) [4 points]: Develop a partition for each parameter based on its type (e.g., number of command-line arguments can be 0, 1 or more, a class name can be provided at the command line or in the default file or in some specific file, an option can be on or off, etc.). Develop another partition for the parameters based on the functionality of the code under test.
(c) [4 points]: Construct test inputs from your partitions to satisfy some coverage criterion (obeying the constraints for the feature you selected). Discuss what criteria are appropriate to use. How many test inputs would there be in "All Combinations"? What values are appropriate for "Base Choice"? What are the boundary values for your partitions?
(d) [4 points]: Write test cases based on your test inputs and commit them into SVN.
(e) [4 points]: Update the potential bug count for your group in the table on the Projects page on Wiki: https://wiki.engr.illinois.edu/display/cs498dmsp12/Projects You should prepare at least two bug reports that you think could be submitted to the JPF developers (but do not submit those reports yet).