Department of Computing

Curtin University

Software Engineering Testing (SET)

Week 12 Laboratory/Tutorial

The following exercises are intended to be done in a laboratory/tutorial session with a teaching assistant or instructor present. The exercises have been designed to reinforce concepts taught in SET.

1) Consider the given code and answer following questions.

```
class Super {
  // Broken - constructor invokes overridable method
  String str = "Hello";
  public Super() { m(); }  // fault
  public void m() {
    // Location "A"
    System.out.println(str);
  };
}
class Sub extends Super {
  private final Date date;  // initially null field; set by constructor
  public Sub() { date = new Date(); }
  public void m() {
    // Location "B"
    System.out.println(date);
  }

  public static void main(String[] args) {
    Super s = new Super();    // Test 1  - No Failure:  prints "Hello"
    Sub t   = new Sub();      // Test 2  - Failure:    prints null
  }
}
```

(a) Describe the state at Location "A" for Test 1. (Note that location "A" is not reached on Test 2).
(b) Describe the state at Location "B" for Test 2.
(c) Does Test 1 satisfy reachability for the fault?
(d) Does Test 2 satisfy reachability for the fault?
(e) Test 2 satisfies infection for the fault. Describe the infection.

2) Consider the given code and answer the following questions.

```
public static void computeStats (int [ ] numbers)
{
int length = numbers.length;
double med, var, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
         sum += numbers [ i ];
        }
        mean = sum / (double) length;
        med  = numbers [ length / 2 ];

        varsum = 0;
        for (int i = 0; i < length; i++)
        {
         varsum = varsum  + ((numbers [ i ] - mean)*(numbers [ i ]-mean));
        }
        var = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( var );

        System.out.println ("length:                " + length);
        System.out.println ("mean:                 " + mean);
        System.out.println ("median:              " + med);
        System.out.println ("variance:             " + var);
        System.out.println ("standard deviation: " + sd);
}
```

(a) Draw the control flow graph for the `computeStats( )` method (*NOTE:* with Defs and Uses sets).

(b) List the test requirements for Edge-Pair Coverage. (Hint: Get 8 TR's of length 2).

(c) List the test requirements for Prime Path Coverage (list any 8 TR's).

3) Consider input domain testing for the Java method `max ( )`:

```
/**
 *  return the max element in a collection
 *  @param c - Collection to be searched
 *  @return - max element in c
 *  @throws - NullPointerException if c is null
 *  @throws - IllegalArgumentException if c is empty
 */
public static <T extends Comparable<? super T>> T max (Collection<? extends T> c)
```

A client could use this method as follows:

```
Collection < ScheduledFuture > futures = ...  // some delay objects
ScheduledFuture sf = max(futures);            // find the maximum one

or

Set <String> s = ... // ["ant", "cat", "dog", "bee"]
max(s)               // "dog"

or

Set t = ...          // ["ant", 42, 3.14159]
max(s)               // compiler warning plus ClassCastException
```

A possible input domain model is:

```
   Characteristic:  Whether collection is empty
        c is empty
        c is not empty

   Characteristic:  Whether collection has multiple elements
        c has multiple elements
        c does not have multiple elements

   Characteristic:  Comparing collection elements
        all elements of collection are mutally comparable
        some elements of collection are mutually comparable
        no elements of collection are mutually comparable
```

(a) There is an obvious characteristic missing that one would expect from an interface-based input domain model. Give this additional characteristic, along with its accompanying partition.

(b) Does the partition "Comparing collection elements" satisfy the disjointness property? If not, give a value for c that fits in more than one block.

(c) Does the partition "Comparing collection elements" satisfy the completeness property? If not, give a value for c that does not fit in any block.

(d) If the "Base Choice" criterion were applied to the partitions (exactly as written), how many test designs would result?

4) Consider the following grammar for a `phoneNumber`:

```
phoneNumber   ::= exchangePart dash numberPart
exchangePart  ::= special zeroOrSpecial other
numberPart    ::= ordinary ordinary ordinary ordinary
ordinary      ::= zeroOrSpecial | other
zeroOrSpecial ::= zero | special
```

```
zero          ::= "0"
special       ::= "1" | "2"
other         ::= "3" | "4" | "5" | "6" | "7" | "8" | "9"
dash          ::= "-"
```

Consider also the following mutation of the grammar:

```
exchangePart ::= special ordinary other
```

(a) Classify the following as either phoneNumbers (or not).

- 123-4567
- 012-3456
- 109-1212
- 246-9900
- 113-1111

(b) If possible, find a string that appears in the mutated grammar, but not in the original grammar.

(c) If possible, find a string that appears in the original grammar, but not in the mutated grammar.

(d) If possible, find a string that appears in the both grammars.