VU Software Engineering	Exercise 4: SE Teil 2
194.020, WS2023	Groups: 12.12. and 19.12.2023

Anmerkung: Bitte bedenken Sie, dass das Erklären der Lösung bei einigen Beispielen nur dann gut möglich ist, wenn man auch die Angabe sieht. Bitte gestalten Sie Ihre Abgabe daher so, dass man direkt bei der Lösung dort wo notwendig daneben auch gleich die Angabe sieht.

Exercise 1: Equivalence Classes & Boundary Values

A local flower shop offers discounts to its most valuable customers. Function `discountPercent` calculates the percentage of discount that should be applied to a purchase.
Purchases of more than 10 flowers are given 10% discount. Customers with membership cards are given 5% discount. (The discounts are cumulative.) Function `discountPercent` throws an `InvalidOrderException` if the number of flowers is zero.

```
1 public int discountPercent(int flowers, boolean membershipCard);
```

a) What are the partitions for each parameter? How many partitions are there in total?

b) What partitions can be combined?

c) What are the boundary values? Which are the on and off points?

d) Construct test cases for function `discountPercent` according to your analysis. Give inputs and expected outputs.

Exercise 2: Specification-Based & Structural Testing

a) Which of these software testing activities correspond to specification-based testing, which correspond to structural testing, and which correspond to neither?

   (a) Asking a colleague to check if the tests match the documentation.
   (b) Measuring which statements are executed by each test case.
   (c) Doing test-driven development.
   (d) Testing with random data to find crashes.
   (e) Constructing test cases to cover all branches.

b) Which of the following statements are correct?

   (a) MC/DC is a stronger property than branch coverage.
   (b) Programs that have 100% path coverage do not contain any kind of bugs.
   (c) Boundary values are extracted from the source code.
   (d) Loop coverage is a stronger property than branch coverage.
   (e) A test suite constructed from boundary values has 100% branch coverage.

Exercise 3: Basic-Block & Branch Coverage

```
1  public int compute(int[] x) {
2    if (x == null) {
3      return 0;
4    }
5    int sum = 0;
6    for (int i = 0; i < x.length; i++) {
7      if (x[i] % 2 == 0) {
8        sum += x[i];
9      }
10   }
11   return sum;
12 }
```

a) Draw the control flow graph. Count the basic blocks and branches.

b) Define test cases that achieve 100% basic-block coverage, but not 100% branch coverage.

c) Define test cases that achieve 100% branch coverage.

Exercise 4: Path & Loop Coverage

a) Consider function `max`.

```
1  public int max(int a, int b, int c) {
2    int max = a;
3    if (max < b) {
4      max = b;
5    }
6    if (max < c) {
7      max = c;
8    }
9    return max;
10 }
```

(a) Count the paths in function `max`.

(b) List a set of test cases that achieve 100% path coverage.

(c) Which of these test cases are sufficient for 100% branch coverage?

(d) Which of these test cases are sufficient for 100% basic block coverage?

(e) How many paths does function `max4` have? How many test cases are necessary to reach 100% path coverage?

```
1  public int max4(int a, int b, int c, int d) {
2    int max = a;
3    if (max < b) {
4      max = b;
5    }
6    if (max < c) {
7      max = c;
8    }
9    if (max < d) {
10     max = d;
11   }
12   return max;
13 }
```

b) Consider function `sumRange`.

```
1  public int sumRange(int[] array, int l, int r) {
2    if (array == null || array.length != 4 || l < 0 || 4 <= r) {
3      throw new IllegalArgumentException();
4    }
5    int sum = 0;
6    while (l < r) {
7      sum += array[l];
8      l++;
9    }
10   return sum;
11 }
```

(a) Construct a minimal set of test cases that achieve 100% loop coverage.

(b) Do these tests reach 100% branch coverage?

Exercise 5: Condition + Branch Coverage

```
1 public String triangle(int a, int b, int c) {
2   if (a+b<c || a+c<b || b+c<a) {
3     return "invalid";
4   }
5   if (a*a+b*b==c*c || a*a+c*c==b*b || b*b+c*c==a*a) {
6     return "right␣angled";
7   }
8   return "other";
9 }
```

a) Count the number of condition values + branches.

b) How much branch coverage does the test (a=1, b=1, c=1) reach?

c) How much C+B coverage does the test (a=1, b=1, c=1) reach?

d) Construct test cases that reach 100% C+B coverage.

Exercise 6: MC/DC

```
1 public int compute(int a, int b) {
2   if ((a * b == 20 || a + b == 12) && a < 10) {
3     return a;
4   } else {
5     return b;
6   }
7 }
```

a) Construct test cases that reach 100% MC/DC. List for each test case which conditions are true and which are false.

b) List the independence pair for each condition.

Exercise 7: DU-Pairs Coverage

```
1  public int range(int a, int b, int c) {
2    int max = a;
3    int min = a;
4    if (a < b) {
5      max = b;
6    } else {
7      min = b;
8    }
9    if (max < c) {
10     max = c;
11   }
12   if (c < min) {
13     min = c;
14   }
15   return max - min;
16 }
```

a) List all DU pairs for variables `max` and `min`.

b) Construct test cases that reach 100% DU-pairs coverage. For each test, list all DU pairs it covers.

Exercise 8: Measuring DU-Pairs Coverage

```
1  public void countFlips(boolean[] coinFlips, boolean countHeads) {
2    int heads = 0;
3    int tails = 0;
4    int result = 0;
5    for (boolean isHeads: coinFlips) {
6      if (isHeads) {
7        heads = heads + 1
8      } else {
9        tails = tails + 1;
10     }
11   }
12   if (countHeads) {
13     result = heads;
14   } else {
15     result = tails;
16   }
17   return result;
18 }
```

a) Draw the control flow graph for function `countFlips` and apply the algorithm for computing reaching definitions for variables `heads`, `tails` and `result`.

b) List the DU pairs for variables `heads`, `tails` and `result`.

c) Instrument the code as shown in the lecture to measure DU-pairs coverage. What is the state of maps `defCover` and `useCover` after running the test case (`coinFlips=[true, true], countHeads = false`)? You may assume the maps start freshly initialized.

Exercise 9: Property-Based Testing

a) An Austrian drink wholesaler would like to apply property-based testing to their web shop. The company offers beverages with alcohol ranging from 0% to 53%. The policy of the web shop states that customers under 16 are only allowed to buy non-alcoholic beverages (0% alcohol). Customers between 16 and 17 are allowed to buy drinks with under 20% of alcohol. There are no restrictions for customers of age 18 or older.

Apply property-based testing to function `canOrderDrink`.

```
public boolean canOrderDrink(int age, int alcoholPercent) { ... }
```

In particular, design property-based tests for the following requirements:

1. People under the age of 16 are allowed to order alcohol-free drinks.
2. From the age of 16 to 17, people are allowed to order drinks whose alcohol percentage is under 20.
3. From the age of 18, people are allowed to order any kind of drink.
4. People under the age of 16 are not allowed to order any alcoholic drink.
5. From the age of 16 to 17, people are not allowed to order drinks with an alcohol percentage of 20 or above.

Use the following template for each test and replace X by the number of the related property.

```
@Property
void testPropertyXCanOrderDrink (
    @ForAll
    @IntRange(min = /*TODO*/, max = /*TODO*/ )
    int age,
    @ForAll
    @IntRange(min = /*TODO*/, max = /*TODO*/ )
    int alcoholPercent ) {
  /*TODO*/
}
```

b) Which of the following statements are correct about property-based testing?

- Writing properties is easier than constructing tests manually.
- Property-based testing should always be used instead of example-based testing.
- With property-based testing, it may be difficult to get an adequate distribution of input values.
- With property-based testing, it is always inexpensive to generate the desired data.
- A property-based testing framework tries to find a counterexample to break the defined properties.

c) Consider function `compute`.

```
public void compute(int a, int b);
@Property
void computeTest(
  @ForAll @IntRange(min = 1, max = 100) int a,
  @ForAll @IntRange(min = 1, max = 100) int b
) {
  // ...
}
```

Function `compute` has a bug and incorrectly throws an exception if inputs `a` and `b` are equal. Assume that for each run of property `computeTest` the values for `a` and `b` are sampled independently and uniformly in the given range.

- What is the probability that a generated pair of input values reveals the bug?
- What is the probability if the `max` value for both variables `a` and `b` is increased to 1000?

Exercise 10: Test Doubles

Classify the following objects into one of the five kinds of test doubles.

a) An external API server that returns pre-defined responses and verifies that specific requests were made during testing.

b) A database connection wrapper that records every query made to a particular table.

c) A database connection that returns pre-defined data for specific queries.

d) A logger that does not perform any logging and is only used to fulfill a method requirement.

e) A file system that emulates the behavior of a real file system without actually writing to disk.

f) An HTTP server that returns pre-defined responses to specific requests.

g) An email service that captures and stores outgoing emails and triggers pre-defined incoming email events.

h) A database connection that ignores all operations and is not used during testing.

i) A logger that records information about logged messages during testing and checks for the existence of certain string patterns.

j) A data prediction unit that, in contrast to its production implementation, uses a simplified algorithm to decrease the runtime of the tests.