Prof. Dr. Peter Thiemann
Sergio Feo-Arenis
Sergiy Bogomolov

Summer Term 2014

Software Engineering

http://proglang.informatik.uni-freiburg.de/teaching/swt/2014/

Exercise Sheet 7

Exercise 1: Battleships (10 Points)

Consider the game battleships: http://en.wikipedia.org/wiki/Battleship_(game). A software engineer is commisioned to write a program that validates a ship configuration according to the rules. A configuration is valid if and only if:

1. It contains exactly ten ships:
   - One aircraft carrier (5 squares)
   - Two cruisers (4 squares)
   - Three destroyers (3 squares)
   - Four submarines (2 squares)
2. The ships are not placed adjacent to each other (there is at least one square free between any two ships).
3. The ships are straight, have no corners or indentations.
4. The ships cannot be placed diagonally.

The code should implement the following interface:

```
public enum GridState {Invalid, Valid, Viable}
public interface BattleshipValidator {
  // returns: Valid if the grid contains a valid configuration,
  // Viable if the configuration does not violate the rules but does
  // not yet contain all ships, Invalid otherwise.
  public GridState validate(boolean[][] grid);

  // adds a ship of the specified size to a grid at position (x,y)
  // with the specified orientation. Returns true if placing the ship
  // respects the rules.
  public boolean addShip(boolean[][] grid, int shipSize, int x, int y,
      bool isHorizontal);
}
```

- Provide a specification (in the form of requires and ensures clauses) for each of the interface methods. Try to be as formal and precise as possible.

Solution

For method `validate`:

- Requires:
  - `grid` is not `null`
  - The dimensions of `grid` are square (e.g. 10x10)
- Ensures: (we consider the grid as a graph, where each cell with value `true` is a node and there are implicit edges to all (up to 8) neighbors)
  - result is `Viable` iff:
    - every connected component has dimensions $X \times 1$ or $1 \times X$ and $2 \leq X \leq 5$
    - the number of connected components of size 2 is smaller or equal to 4
    - the number of connected components of size 3 is smaller or equal to 3
    - the number of connected components of size 4 is smaller or equal to 2
    - the number of connected components of size 5 is smaller or equal to 1
    - the total number of connected components N is less than 10
  - result is `Valid` iff:
    - every connected component has dimensions $X \times 1$ or $1 \times X$ and $2 \leq X \leq 5$
    - the number of connected components of size 2 is smaller or equal to 4
    - the number of connected components of size 3 is smaller or equal to 3
    - the number of connected components of size 4 is smaller or equal to 2
    - the number of connected components of size 5 is smaller or equal to 1
    - the total number of connected components N is exactly 10
  - result is `Invalid` otherwise.

For method `addShip`:

- Requires:
  - `grid` is not `null`
  - The dimensions of `grid` $n \times m$ are square (e.g. 10x10), i.e. $n = m$.
  - $2 \leq$ `shipSize` ≤ 5
  - $0 \leq$ `x` $< n$ and $0 \leq$ `y` $< n$
- Ensures:
  - `grid` is updated so that all cells between position $(\texttt{x}, \texttt{y})$ and
    - when `isHorizontal` is `true`, $(\texttt{x} + \texttt{shipSize}, \texttt{y})$
    - when `isHorizontal` is `false`, $(\texttt{x}, \texttt{y} + \texttt{shipSize})$

    contain the value `true`
  - result is `true` iff:
    - all cells in `grid` between position $(\texttt{x}, \texttt{y})$ and
      - when `isHorizontal` is `true`, $(\texttt{x} + \texttt{shipSize}, \texttt{y})$
      - when `isHorizontal` is `false`, $(\texttt{x}, \texttt{y} + \texttt{shipSize})$

      contained the value `false` on entry.
    - when `isHorizontal` is `true`, $\texttt{x} + \texttt{shipSize} < n$
    - when `isHorizontal` is `false`, $\texttt{y} + \texttt{shipSize} < n$
    - `isValid(grid)` after updating returns `Valid` or `Viable`
  - result is `false` otherwise.

Exercise 2: Roman numerals (10 Points)

Consider code that converts between arabic and roman numerals:

```
public interface RomanNumeralConverter {
  // returns the arabic numeral representation of the input string
  // or -1 if invalid
  public int toArabic(String roman);

  // returns a string with the roman numeral representation of the
  // input number, the input number should be positive and
  // no greater than 3000
  public String toRoman(int arabic);
}
```

- Provide test cases according to the *black box* testing principle for both interface methods.
- How many test cases are necessary for each method?

Solution

In both cases, the number of tests necessary depend on the technique used: error guessing, boundary value analysis, equivalence partitioning, etc.
For method `toArabic`: We use for example error guessing, thus we create test cases for:

- corner cases
- random "easy" inputs
- (almost) all inputs that are considered "difficult"

For this method, one would expect the difficult cases to be those where subtraction is required to obtain the value of a numeral, e.g. "CM" $\rightarrow$ 900. Additionally, the method should recognize all probable symbol inversions, e.g. "ID" or "VM" as invalid.

Input	Expected Output
" " (empty string)	-1
"I"	1
"II"	2
"III"	3
"IIII"	-1
"IV"	4
"VX"	-1
"ID"	-1
"VD"	-1
"IC"	-1
"VC"	-1
"DC"	-1
"CI"	101
...	-1

For method `toRoman`: We can consider the classical implementation of the conversion to derive test cases.

```
private static final int[]    VALUES  = { 1000, 900,  500, 400,  100,
    90,   50,  40,   10,  9,    5,   4,    1   };
private static final String[] SYMBOLS = { "M",  "CM", "D", "CD", "C", "
    XC", "L", "XL", "X", "IX", "V", "IV", "I"  };
```

```
public static String arabicToRoman(int arabic) {
  StringBuilder result = new StringBuilder();
  int remaining = arabic;
  for (int i = 0; i < VALUES.length; i++) {
    remaining = appendRomanNumerals(remaining, VALUES[i], SYMBOLS[i],
        result);
  }
  return result.toString();
}

private static int appendRomanNumerals(int arabic, int value, String
    romanDigits, StringBuilder builder) {
  int result = arabic;
  while (result >= value) {
    builder.append(romanDigits);
    result -= value;
  }
  return result;
}
```

From here, we test the use of every element in the table individually:

Input	Expected Output
1	"I"
4	"IV"
5	"V"
9	"IX"
10	"X"
40	"XL"
50	"L"
90	"XC"
100	"C"
400	"CD"
500	"D"
900	"CM"
1000	"M"

We also consider corner cases

Input	Expected Output
0	" " (empty string)
3000	"MMM"
3001	" " (empty string)

Finally, some combinations can be tested

Input	Expected Output
94	"XCIV"
301	"CCCI"
499	"CDXCIX"
944	"CMXLIV"
2949	"MMCMXLIX"

Submission

- Submit this sheet *before* the lecture of Thursdays.
- Late submissions will not be accepted.
- Deadline: Thursday 11:59 a.m.