Designing Contracts

This week, we are switching gears by studying techniques that code developers can employ to ensure the correctness and robustness of the code they write.

1. Wait for the instructor to demo pre- and post- conditions with `calculateTax`. Now is your turn - come up with pre- and post- conditions for `add` and `remove` methods in `Basket` class:

```
1  public class Basket {
2    private BigDecimal totalValue = BigDecimal.ZERO;
3    private Map<Product, Integer> basket = new HashMap<>();
4    public void add(Product product, int qtyToAdd) {
5
6
7
8
9
10     // add the product
11     // update the total value
12
13
14
15
16
17
18
19   }
20   public void remove(Product product) {
21
22
23
24
25
26
27     // remove the product from the basket
28     // update the total value
29
30
31
32
33
34
35   }
36 }
```

Consider the following code:

```
class Board{
    Square[][] board;

    // constructors and other methods

    public Square squareAt(int x, int y){
       assert x >= 0;
       assert x < board.length;
       assert y >= 0;
       assert y < board[x].length;
       assert board != null;
       Square result = board[x][y];
       assert result != null;
       return result;
    }

    // other methods
}
```

2. Out of all the assertions which one or which combination of them could become the class invariant for the class `Board`?

3. Suppose we remove the last assertion (`assertresult!=null`), which states that the result can never be `null`. Are the existing pre-conditions of the `squareAt` method enough to ensure the property of the removed assertion?

4. What can we add to the class (other than the just-removed post-condition) to guarantee this property?

Consider the following scenario of inheritance for the method `getBirthYear(String name)` that is implemented in the parent and then overloaded by each of the children classes. The pre- and post-conditions for the parent and children classes are as follows:

Parent Class	Child 1 Class	Child 2 Class	Child 3 Class
`len(name) upto 50`	`len(name) upto 50`	`len(name) upto 25`	`len(name) upto 75`
`returns >= 1920`	`returns >= 1920`	`returns >= 1900`	`returns >= 1950`

5. Draw the inheritance diagram assuming that the `Client` class uses a list of `Parent` objects.

6. Explain which overloaded method would fail and why (i.e., it could be due to the pre-condition or the post-condition or both)? Explain why.

Consider the following documentation for `indexOf` method:

```
/**
 * Finds the index of the given value in the array starting at the
 * given index.
 *
 *
 * @param array
 *            the array to search through for the object, may be null
 * @param valueToFind
 *            the value to find
 * @param startIndex
 *            the index to start searching at
 * @return the index of the value within the array, INDEX_NOT_FOUND
 *         (-1) if not found or {@code null} array input
 */
public static int indexOf(final int[] array, final int valueToFind,
                                                int startIndex) {
    ...
}
```

7. What are the pre-conditions for this method?

8. What are some stronger and softer ways of dealing with the invalid cases for this method in our code?