Homework 0

Arik Rinberg 317156669

Barak Zan 305634487

Question 1 – Coding Style Guidelines

Naming Conventions

Package naming conventions

Package names should be all lowercase, with consecutive words with no separation. For example $com.technion.mypackage$ and **not** $com.technion.myPackage$ or $com.technion.my\_package$.

Class naming conventions

Class names should be upper camel case, without underscores. The class should be descriptive without shortening the words (except for known abbreviations). For instance a class name for event handling should be $EventHandler$ and not $Events$.

Function name conventions

Method names should be lower camel case for non-static methods, and upper camel case for static methods, and should be descriptive. For instance "$getSumOfElements()$" and not "$total()$".

Variable name conventions

Variables should be lower camel case and will be nouns. The variable names should be descriptive of what they are, and not abbreviated. For example, "$int\ sumOfElements$" and not "s" or "$sumElmnts$".

Private variables will start with an underscore.

Variables representing constants should be all uppercase such as "$EPSILON$".

Line styling and indentations

Line length

The length of a line will not exceed 120 characters.

Indentation and curly brackets

Line indentation will be done with tabs to represent the scopes. Open curly brackets will be opened in a new line.

For instance:

```
if (cond)
{
	// do something…
}
```

Blocks in the code will separated by a new line.

Comments and Documentation

Above every class and method should be a clear documentation.
Documentation of methods must include specifications as described in the course.
Comments should be added to the code when the code isn't self-explanatory.

Question 2 – Comment Reader

Input to CommentReader

```
/*
 * Comment on a class
 */
public class TestClass {
	private int i1; // A first number
	private double d2; // Another number

	/* this explain a bit */ private int i3; /* explain more */
	/* a comment */ // Another comment

	/**
	 * description for test class
	 */
	public TestClass(double d)
	{
		i1 = 1;
		d2 = d;
		i3 = 1;
	}

	// Will this be fine /*
	public int sum()
	{
		// Returns a value
		return i1 + i3;
	}

	/* and another
	 * multi
	 * line /* explaining the line
	 */
}
```

Output of CommentReader

```
 * Comment on a class

 A first number
 Another number
 this explain a bit  explain more
 a comment  Another comment
*
	 * description for test class

 Will this be fine /*
 Returns a value
 and another
	 * multi
	 * line /* explaining the line
```

Question 3 – Ball Container

Different implementations for getVolume()

Our first implementation to getVolume() was:

```
public double getVolume()
{
	double sum = 0;
	for (Ball ball : balls)
	{
		sum += ball.getVolume();
	}
	return sum;
}
```

Our second implementation to getVolume() was holding a variable in the class and updating it on every add and remove, and zeroing the variable when clearing the container.

The first implementation is better if the calls to getVolume() are infrequent. As there is no extra work to be done on update and remove, on the rare occasions we call getVolume(), iterating over the ArrayList won't matter so much. Also this implementation uses less memory than holding an extra variable.

However, if the calls to getVolume() are frequent, we don't want to waste time constantly iterating over the ArrayList, so using an updated variable is more efficient.

Change to @requires for add()

Needed alterations to the specification

The addition of the @requires doesn't need changes to the specification, as nowhere in the specification referenced what happens if the parameter ball was null.

Needed alterations

The addition of the @requires doesn't need changes to the implementation. We can now forgo the check if ball is null but we don't have to.

Strength of specification after alteration

The specification is now weaker, as the @requires is longer and the @effects remains the same.

Question 4 – Multiple choice

Choice between two specifications

The correct answer is **א**, it cannot be established which specification is stronger. For a specification S to be stronger than a specification S' we require that (1) the @requires be stronger and (2) the @effects be more specific. In the question $S1$ has a stronger @requires but $S2$ has a stronger @effects, so we cannot establish a stronger specification.

Specification of the method find()

$I1$ implements $S1$

The correct answer is **ד**, in this case $I1$ also implements $S2$ and $S4$. It implements $S2$ because they have the same effect, and it implements $S4$ because if the color is the list it will return the location of the color. It does not implement $S2$ on account of the error thrown if the color is not found.

$I2$ implements $S2$

The correct answer is **ו**, the implementation only implements $S2$. The implementation may rely on the fact that the list is ordered in order to find the location. For $S1$ the list may be not ordered, and therefore the implementation may return the wrong value. For $S3$, the specification throws an error if not found, therefore $I2$ doesn't implement it. Finally for $S4$, in the same way as $S1$ the function may return the wrong value as the list may be not ordered.

$I3$ implements $S3$

The correct answer is **ג**, the implementation also implements $S4$. For $S4$, if color is in the list the location will be returned which is the only return value specified. For $S1$ and $S2$, if the color isn't in the list, the implementation will throw an error rather than return -1, so $I3$ doesn't implement these two specifications.

$I4$ implements $S4$

The correct answer is **ו**, the implementation only implements $S4$. The specification doesn’t specify what happens if color isn't in the list, therefore the implementation may return the value -2 which is not as specified in all the other specifications.