Joshua Shaham, shaham.j@husky.neu.edu
Moses Gonzalez, mgonzalez1130@gmail.com
Zhuoli Liang, liang.zhu@husky.neu.edu

CS 5500 HW 4

October 1, 2014

The procedure

First, the test cases of each individual in the group were combined to create one large test program. Then, this new program was used to check each members' implementations for errors.

The test cases were partitioned into several components:
- toString() was checked to see if the SearchableStrings ADT returned the string it represents.
- equals(input) was checked for several cases: when the input is null, when the input is not a string, when the SearchableStrings' string matches the input, and when the SearchableStrings' string differs from the input.
- hashCode() was checked to see if the SearchableStrings ADT returned the same Object-defined hash code as the string it represents.
- distance(input1, input2) was checked for each axiom in the specification and then checked against not only the examples given in the homework prompt but also many unique ones.
- bestMatch(input) was checked for each axiom in the specification, and then checked with against not only the examples given in the homework prompt but also many unique ones.

Joshua Shaham's implementation

A mistake was identified in the .equals(object) method:
"
```
public boolean equals(Object x) {
        if (x instanceof SearchableString) {
                if (x.toString() == s) return true;
                else return false;
        }
        else return false;
}
```
"
This bug was the result of using the equality operator == instead of the proper string.equals() method to compare two strings. A simple correction follows:
"
```
                if( x.toString().equals(s) ) return true;
```
"
Even though the logic is correct, no specific case was added to check if the input is null. Adding the null scenario would provide clarity with regards to the specification.

Tangentially, an extra method length() is defined which, while not unnecessary, provides no immediate use. It is never called. Thus, simplicity dictates that it should be removed.

In terms of style, this individual added unnecessary comments relating the function to it's specification, which can be deemed distracting.

Asides from these remarks, this implementation succinctly implements the SearchableString interface in a fashion that respects the principle of information hiding.

Moses Gonzalez's implementation

A mistake was identified in the .equals(object) method:

“

```
public boolean equals(Object obj) {
	if (obj instanceof SearchableStringsInstance) {
		return (thisString == obj.toString());
	} else {
		return false;
	}
}
```

“

The object is checked against the implementation-specific SearchableStringsInstance, instead of against the SearchableStrings class.

Tangentially, the non-inherited methods (the helper functions in this instance) should have comments that describe their purpose. For such a basic assignment, it is simple to see why they are not necessary. However, it is a good practice that will be appreciated on larger projects, especially when the code needs to be edited either by a different coder or after a sufficiently large time.

Asides from these remarks, this implementation succinctly implements the SearchableString interface in a fashion that respects the principle of information hiding.

Zhuoli Liang's implementation

Albeit no errors were detected for this individuals implementation, there were a few stylistic gripes:

The first instance of this manifests in the adherence to the Design Recipe - which is redundant in Java because the Design Recipe is built into value definitions, e.g.,

“

```
// GIVEN: SearchableString
// RETURN: int
...
public int bestMatch(SearchableString obj) {
	blah
}
```

“

In lieu of the given/return comments, a remark describing what the method does would be helpful. However, in this instance, the methods are inherited, so no description is necessary.

A second instance appears in the distance() function:

"

```
...
public int distance(SearchableString obj, int i) {
	String s1=this.str;
	String s2=obj.tostring();
	int dis=-1;
	if(i<0) {
		dis=s1.length()+s2.length()+1;
	}else if((0<=i) && (i<=s1.length()-s2.length())){
		dis=f(s1.substring(i,i+2.length()),s2)
				+s1.length()-s2.length();
	}else if(0<=i && (i>s1.length()-s2.length())){
		dis=s1.length()+s2.length()+1;
	}

	return dis;
}
```

"

This block of code could benefit from extra indentation. Moreover, defining and returning 'dis' is unnecessary, since one of the conditionals MUST be true! A simple fix would remove the "int dis=-1;" and "return dis;" lines, and instead have the conditional statements return 'dis' value, e.g.,

"

```
if(i<0) {
	return s1.length()+s2.length()+1;
}
```

"

The last stylistic gripe would be about the lack of a base class that implements the SearchableString, which is then extended by SearchableStrings. Presently, the assignment was simple enough that no such base class was necessary. In a less trivial example, such a base class would be imperative to appreciate Java's polymorphism.

Asides from those remarks, the implementation ran smoothly, and adhered to the principle of information hiding.