Prepared by Linan Qiu <lq2137@columbia.edu> and Zeynep Ejder <ze2113@barnard.edu>

String Manipulation

`substring()`

```
// in a main method far far away...

String s = "Han Solo";
for (int i = 0; i < s.length(); i++) {
  System.out.println(s.substring(0, i + 1));
}
```

produces

```
H
Ha
Han
Han
Han S
Han So
Han Sol
Han Solo
```

Don't forget that white space chars count as proper chars.

`charAt()`

```
// in a main method far far away...

String s = "Han Solo";
s.charAt(2); // 'n'
```

`charAt()` returns a `char` not a String

`equals()` and `equalsIgnoreCase()`

```
// in a main method far far away...

String a = "Han Solo";
```

```
Strinb b = "han solo";

System.out.println(a.equals(b)); // false
System.out.println(a.equalsIgnoreCase(b)); // true
```

concat()

```
String s = "Hello World";
String newS = s.concat(", I'm Zeynep");
System.out.println(newS); // Hello World, I'm Zeynep
```

replace()

```
String newS = s.replace('a','e');
System.out.println(newS);  // Hen Solo

newS = s.replace('l','a');
System.out.println(newS);  Han Soao
```

toLowerCase()

```
System.out.println(s.toLowerCase()); // han solo
```

toUpperCase()

```
System.out.println(s.toUpperCase()); // HAN SOLO
```

indexOf()

(Warning this runs in $O(N)$)

```
// works for char
int index = s.indexOf('o');
System.out.println(index); // prints 6

// and string
int index = s.indexOf('Han');
System.out.println(index); // 0
```

This is often used for searching for a substring within a string

`split()`

```
String[] array = s.split(" ");
System.out.println(array[0]); // Han
System.out.println(array[1]); // Solo
```

Appending Strings to Strings

This is one of the most common tasks ever. Almost 80% (untested statistic) of the work you do involves manipulating strings. Do this the wrong way and you'll be in lots of late nights.

```
public class Pikachu {

    private String name;
    private int level;
    private String[] moves;

    public Pikachu(String name, int level, String[] moves) {
        this.name = name;
        this.level = level;
    }

    // bad toString()
    public String toString() {
        String returnString = name + "\n";
        returnString = returnString + level + "\n";
        for(int i = 0 ; i < moves.length; i++) {
            returnString = returnString + moves[i] + "\n";
        }

        return returnString;
    }
}
```

What's wrong with this `toString()` method? Well, whenever you do something like this

```
String a = "hello"; // created one string
a = a + " world"; // creates a copy of a then creates a " world" string, then creates a new
```

You are essentially copy over the original string, creates a second " world" string, then joining the two together. This may be fine for small strings, but imagine that our Pikachu has 1000 moves. Then you'll be copying ever increasing lengths

of strings, doing a lot of duplicate work. Poor Pikachu would die of exhaustion if you used that method. In fact, this is `O(n^2)`

Instead, use the `StringBuilder` class.

```
public class Pikachu {

    private String name;
    private int level;
    private String[] moves;

    public Pikachu(String name, int level, String[] moves) {
        this.name = name;
        this.level = level;
    }

    // bad toString()
    public String toString() {
        StringBuilder sb = new StringBuilder(moves.length * 16); // estimated length
        sb.append(name);
        sb.append("\n");
        sb.append(level);
        sb.append("\n");
        for(String move : moves) {
            sb.append(move);
            sb.append("\n");
        }

        return sb.toString();
    }
}
```

Each `append()` operation is `O(1)`. This makes the entire string building process `O(n)` instead of `O(n^2)`

Pika!

Almost 80% (again untested) of the work you do involves input and output. In 1004, you learned how to use `Scanner`. Generally, try not to use it, especially if you're pretty sure about the kind of data you're going to get. Why? Because `Scanner` is slow. It does a lot of preprocessing for the data (how else does it know whether the next guy is a `double`, `int` etc.)

How then do you deal with input and output? Easy! Use input and output streams.