Assignment 9 – Student

Learning Purpose:

Practice

- Arrays
- Use of static fields
- Random number generation
- Formulating Algorithms
- Formatted output
- toString method

Sample Output:

```
S00123457: Maria Johnson

Sum  Frequency  Percentage
 2      2799       7.8%
 3      5535      15.4%
 4      8312      23.1%
 5     11102      30.8%
 6     13889      38.6%
 7     16892      46.9%
 8     13894      38.6%
 9     10943      30.4%
10      8309      23.1%
11      5547      15.4%
12      2778       7.7%
```

Max Points:

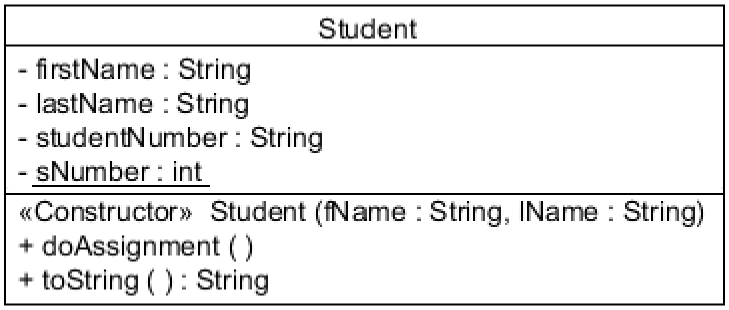
30 points + 5 bonus points

Turn in:

Turn in **A9_*YourName*.zip** via Virtual Campus.

Instructions:

Create a new project called Student and add the following two files: **Student.java** and **StudentTest.java**.



```
Student
- firstName : String
- lastName : String
- studentNumber : String
- sNumber : int
«Constructor» Student (fName : String, lName : String)
+ doAssignment ( )
+ toString ( ) : String
```

Student.java

The class Student represents a student, who has a first name and last name and a student number. A student also has the ability to do an assignment.
In this case the assignment has to do with probability and dice.

Ad Fields:

- **firstName** and **lastName** are of type String and represent the student's name
- **studentNumber** is also of type String. It is a 6-digit number with leading "S00"
- **sNumber** is a static field of type int. This field belongs to the class (type) rather than any specific instance. sNumber is initialized with the value 123456. Every time a new student is created sNumber is incremented. The sNumber will be used to create unique studentNumbers.

Ad constructor:

The class Student has one constructor with two parameters fName and lName.
The studentNumber is not passed as argument but rather created based on the current value of sNumber.
In the constructor the static field sNumber is incremented by one. The student Number is the uppercase letter 'S' followed by two zeros and the current sNumber.

Ad methods:

doAssignment:

The method doAssignment takes no parameter and returns no value.
It simulates the rolling of 2 dice and keeps track of the count in a one-dimensional array of type int. The array elements indicate how often a given count (2 – 12) has been 'rolled'

Even though two dice are rolled there should only be one instance of type Random. It is first used to 'roll' the first die and then again to 'roll' the second die. To get a good distribution the dice will be rolled 100,000 times.

When rolling two dice the sum will be a value from 2 to 12. However, not every sum has the same probability of being rolled. There are three ways of rolling a count of 4, 6 ways of rolling a count of 7 but only one way of rolling a count of 12.

Once the dice have been rolled 100,000 times the result need to be displayed in tabular form.

- First column: the count;
- Second column: how often that count was rolled;
- Third column how often the count was rolled in percent. In the third column the output should include the % sign.
  Tip: In order to output % within a format string write %%

BONUS POINTS:
You can earn 5 extra points when you add a fourth column listing the actual probability for each of the counts – expressed as a fraction.

toString:
Every class has a toString method. The purpose of the toString method is to return a String representation of the object. It is called implicitly every time an object needs to be converted to a String (e.g. in a print method). The default implementation of toString includes the name of the object's class but a student object would be better represented by the student's name and studentNumber. That's why we override the method. You can use the code below and copy it in your class under // method(s)

```
@Override
public String toString()
{
   return String.format("%s: %s %s", studentNumber, firstName, lastName);
}
```

The first line @Override is an annotation. It lets everyone know that we intentionally modify an existing method.
In the 4th line we call the method format of class String. It has a format string with format specifiers just like printf. The difference is that it does not write anything on the console. Instead, it returns the information as a string.

DiceNumbersTest.java

- Create an instance of type Student called myStudent.
- Display the data of the new object
  We use the toString method to do that. I could write: `System.out.printf("%s\n", myStudent.toString());`
  However, it is not necessary to call the toString method in a print statement. I will be called automatically (implicitly)
  You can achieve the exact same results by calling:

  ```
  System.out.printf("%s\n", myStudent);
  ```
- Call the method doAssignment