Notes / Rules

In general there are no rules to this exercise other than your solutions **must be in java**. You can use whatever tools, resources and libraries you choose. You may be asked to defend and explain every line of the solutions however.

Readability and correctness are generally favored over absolute performance considerations. However, feel free to write more than one solution to the same problem if that highlights different trade-offs.

If something is unspecified or unclear in the problem description, make an assumption and note it down.

There are not considered to be right/wrong solutions. This is just a vehicle for conversation - to help our engineering team understand your style and thinking, and have to have an opportunity to hear you talk about code.

Candidates typically share their solutions by sharing their screen with Java IDE and walking our team through the code, watching it run, etc.. Often we use a google+ hangout to achieve this - as it allows for easy group video chat with screen sharing. If it is to be an in-person interview the simplest thing may be to bring your own laptop to the interview, but you may also consider sending us a zip file with the code and your IDE project files (Eclipse or IDEA) and all dependencies all set to go. If you have "logistical" needs to make this possible, please let us know.

Make sure to have fun!

1. hashCode() / equals() problem

Consider these two simple classes

```
public class Person {
	private final int age;
	private final String name;

	public Person(String name, int age) {
		this.name = name;
		this.age = age;
	}
}

public class Employee extends Person {
	private final String role;

	public Employee(String name, int age, String role) {
		super(name, age);
		this.role = role;
	}
}
```

The task is to write appropriate `hashCode()` and `equals()` methods for both classes. Nothing from the above source should be modified but additional fields and methods can be added. Instances of Person and Employee should never be equal to one another. An Employee is equal to another Employee if the role, age, and name are all equal between instances. A Person is equal to another Person instance if the age and name are both equal between instances.

2. Source file reintegration problem

Download this collections of java source files
http://dl.dropbox.com/u/23064251/merge.tar.gz

There are three directories in that tarball (“source”, “dest” and “model”)

The “source” directory contains a small number of java files. The task is to copy/move them from the “source” directory into the appropriate subdirectory of the “dest” based on information found in “model” directory

As an example consider this file from the “source” folder:

```
source/com/tc/text/Banner.java
```

Searching the “model” directory there exists a `Banner.java` at this path:

```
model/common-api/src/com/tc/text/Banner.java
```

Therefore the file should be moved from “source” to “dest” with the path discovered in “model”

```
dest/common-api/src/main/java/com/tc/text/Banner.java
```

There doesn’t need to be any particular interface for this program (ie. paths can be hard coded so as to not require any command line arguments). There is no specification for any output either, the task of moving the files to the appropriate location is only requirement. Any error/exceptional conditions (if any) can simply be printed in any format desired to System.err

3. Statistic Calculator

The task here is to implement this interface. If you want to write multiple implementations then that is good, or you can focus down on writing just one. Some notes regarding implementations:

- Implementations of this class must be 'thread-safe'. The definition of the term 'thread-safe' is left up to you.
- Implementations may choose to prioritize (or not) various aspects of the performance and or behavior of their instances - this is acceptable as long as the compromise can be explained and justified.
- You are free to choose appropriate behavior for any corner cases but are expected to be able to justify those behaviors.
- If you want to optimize for ‘performance’ (whatever you take performance to mean) here at the expense of code readability then that is fine, but you’ll need to be able to explain the code to us. If something appears counter-intuitive or overly complex then code comments can be useful.

```
package statistics;

/**
 * Tracks the statistics of a continual stream of values.
 */
public interface Statistic {

  /*
   * Called to update the statistic with a new sample value.
   */
  void event(int value);

  /*
   * Returns the mean of the received sample values.
   */
  float mean();

  /*
   * Returns the minimal received sample value.
   */
  int minimum();

  /*
   * Returns the maximal received sample value.
   */
  int maximum();

  /*
   * Returns an estimate of the variance of the total population
   * given the received sample values.
   */
  float variance();
}
```