CS 342 Software Design

- Project #1 Discussion
- Writing Test Cases with JUnit

https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

```
public Iterator<I> createIterator()
```

```
public class GLIterator<I> implements Iterator<I>
```

```
GenericList<I>.Node<I> theList;
```

Clicker Question: How did you test your code?

A) Tested each method as it was written
B) Tested each class as it was written
C) Tested at the end and not before
D) Didn’t bother, my coding skills are flawless

Software Design = Testing Software

- A software test is a piece of software, which executes another piece of software.
- It validates if that code results in the expected state (state testing) or executes the expected sequence of events (behavior testing).
- Many benefits from automating the process.
- Most languages have several testing frameworks

Regression Testing:

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes.

Unit Testing:

- A unit test is a piece of code written by a developer that executes a specific functionality in the code to be tested and asserts a certain behavior or state.
- The percentage of code which is tested by unit tests is typically called test coverage.
- Targets a small unit of code, like a method, removing external dependencies

What would we test for project #1?

At your table, think about and discuss what test cases you would write for project #1. Lets focus on just the GenericStack class.

Share your list on the screen at your table.

JUnit: Popular Testing Framework for Java

How is it incorporated in the code: By convention, Maven keeps a separate source folder and test folder on the main branch. Keeps test code separate from production code:

src/main/java - for Java classes

src/test/java - for test classes

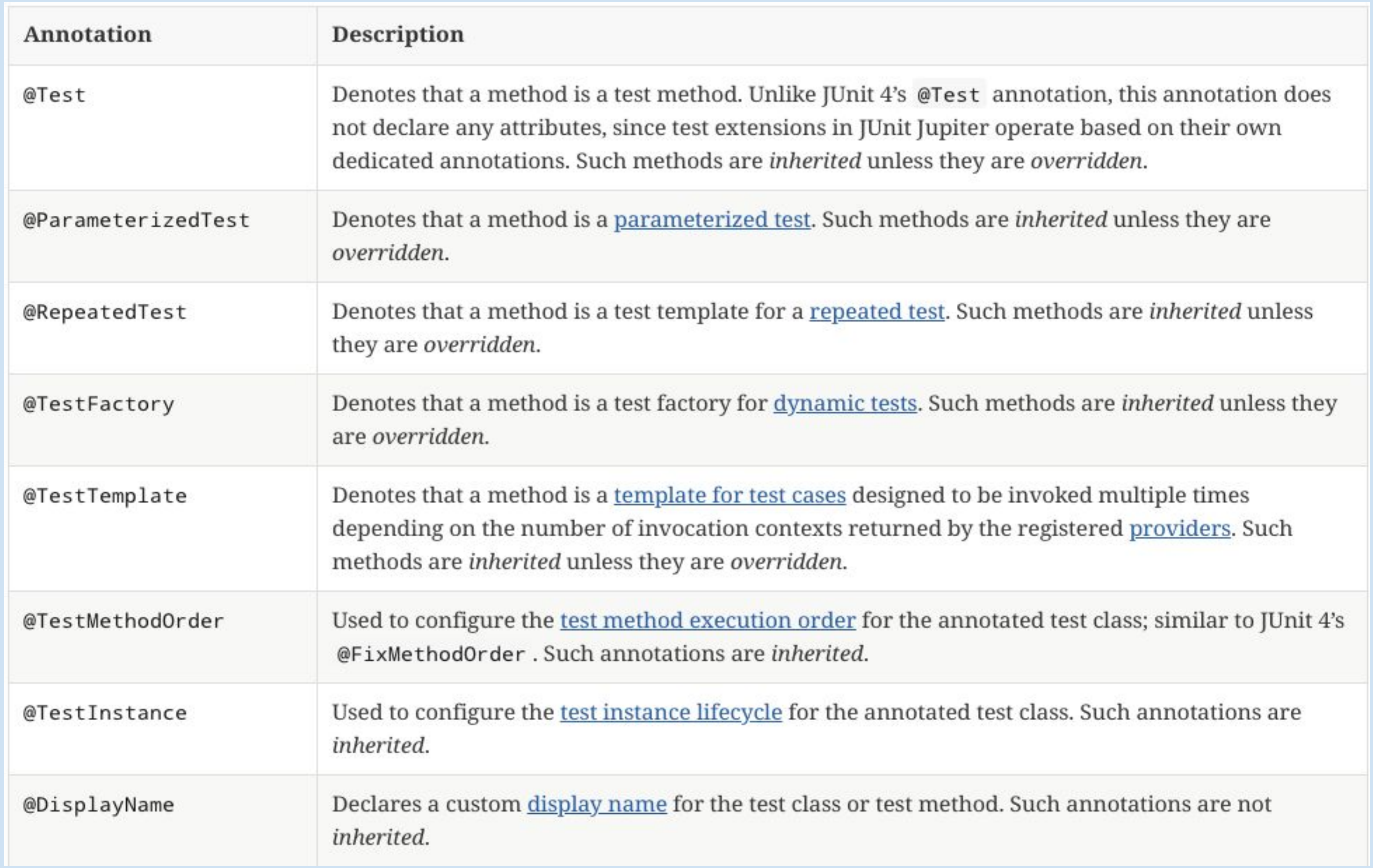
JUnit WebPage: https://junit.org/junit5/

How to Define a Test in JUnit:

- Create a public test class
- Write methods to test code using the @Test annotation
- Use assert statements to test actual results against expected results.
- Provide meaningful messages in the assert statements

Lets see how it works!

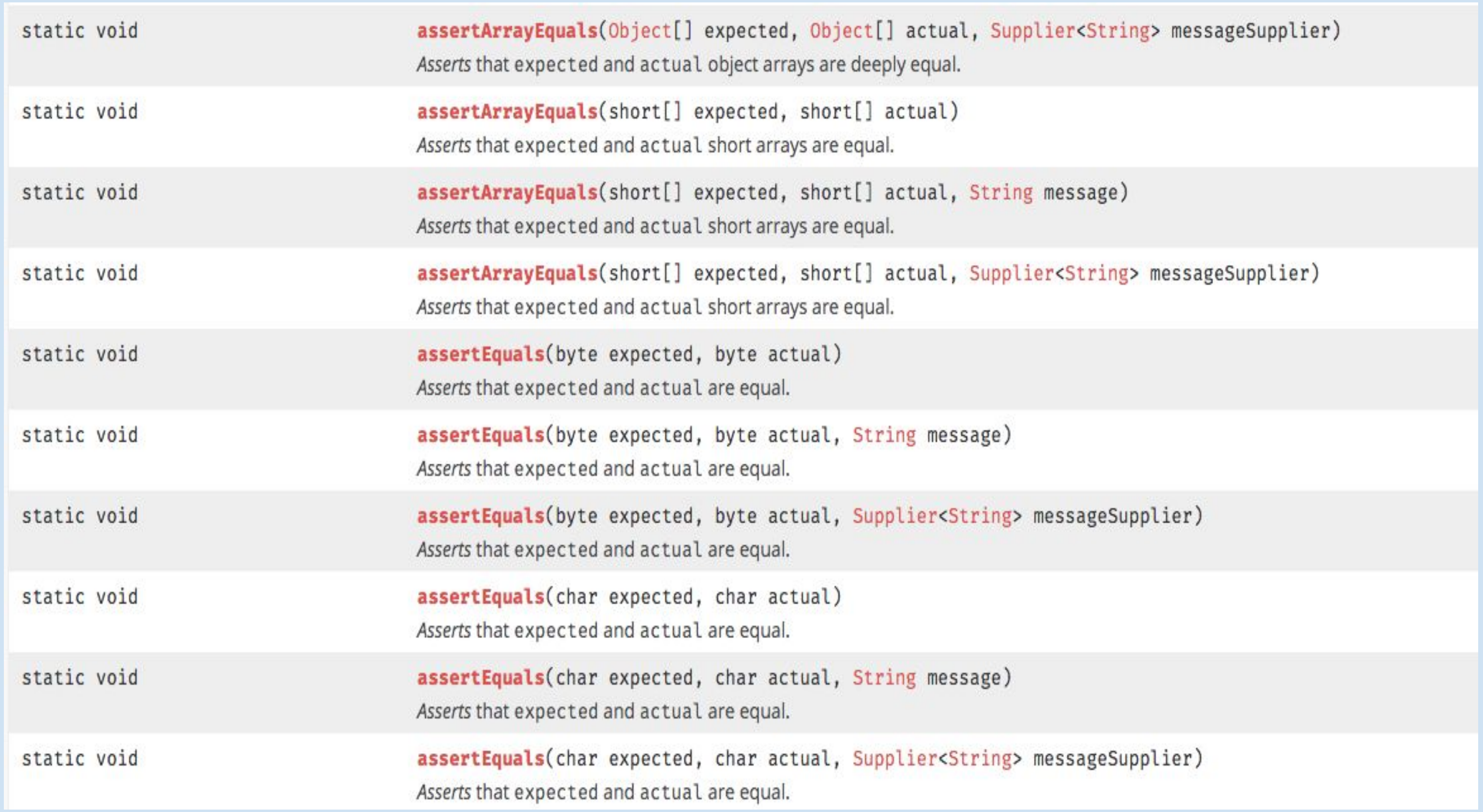
JUnit5



Annotation	Description
`@Test`	Denotes that a method is a test method. Unlike JUnit 4's `@Test` annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are *inherited* unless they are *overridden*.
`@ParameterizedTest`	Denotes that a method is a parameterized test. Such methods are *inherited* unless they are *overridden*.
`@RepeatedTest`	Denotes that a method is a test template for a repeated test. Such methods are *inherited* unless they are *overridden*.
`@TestFactory`	Denotes that a method is a test factory for dynamic tests. Such methods are *inherited* unless they are *overridden*.
`@TestTemplate`	Denotes that a method is a template for test cases designed to be invoked multiple times depending on the number of invocation contexts returned by the registered providers. Such methods are *inherited* unless they are *overridden*.
`@TestMethodOrder`	Used to configure the test method execution order for the annotated test class; similar to JUnit 4's `@FixMethodOrder`. Such annotations are *inherited*.
`@TestInstance`	Used to configure the test instance lifecycle for the annotated test class. Such annotations are *inherited*.
`@DisplayName`	Declares a custom display name for the test class or test method. Such annotations are not *inherited*.

https://junit.org/junit5/docs/current/user-guide/

Asserts



static void	assertArrayEquals(Object[] expected, Object[] actual, Supplier<String> messageSupplier) *Asserts* that expected and actual object arrays are deeply equal.
static void	assertArrayEquals(short[] expected, short[] actual) *Asserts* that expected and actual short arrays are equal.
static void	assertArrayEquals(short[] expected, short[] actual, String message) *Asserts* that expected and actual short arrays are equal.
static void	assertArrayEquals(short[] expected, short[] actual, Supplier<String> messageSupplier) *Asserts* that expected and actual short arrays are equal.
static void	assertEquals(byte expected, byte actual) *Asserts* that expected and actual are equal.
static void	assertEquals(byte expected, byte actual, String message) *Asserts* that expected and actual are equal.
static void	assertEquals(byte expected, byte actual, Supplier<String> messageSupplier) *Asserts* that expected and actual are equal.
static void	assertEquals(char expected, char actual) *Asserts* that expected and actual are equal.
static void	assertEquals(char expected, char actual, String message) *Asserts* that expected and actual are equal.
static void	assertEquals(char expected, char actual, Supplier<String> messageSupplier) *Asserts* that expected and actual are equal.

https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html

Write the JUnit5 test cases.

Take the test cases you thought of for the GenericStack class and write them as JUnit5 test cases with the proper annotations and asserts.

Share your test cases on the screen at your table.

DO NOT share your project code!!!!!!