Annotations, JUnit

Please go through material for these topics. Complete the reading, exercises, and any videos linked. If the instructions ask to turn in any exercises, please do so through slack to your instructor.

Outline

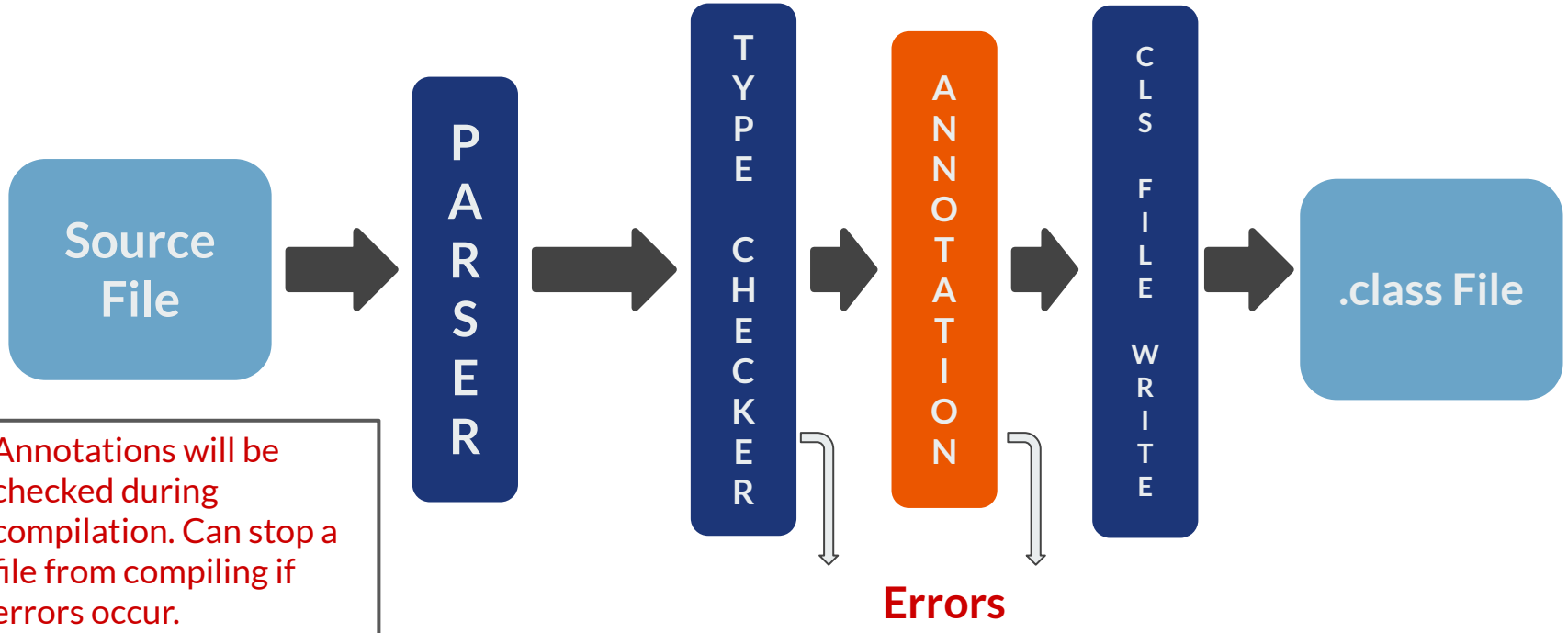
1. Annotations
    - What are annotations
    - @Override example
2. JUnit
    - JUnit Video
    - Test Driven Development
    - JUnit in Java
    - Exercise
3. Open Book Quiz

Annotations

Annotations

- **Annotations** start with `@` symbol followed by some keyword/text, it marks code/element so it is processed in some special way
- ***Indicate to a compiler, dev tool, deployment tool, runtime environment, or framework that binds to the associated code***
- Can be declared on elements like classes, constructors, methods, etc.
- Common annotation found everywhere is `@Override` to check if method is really overriding method from parent class at compile time

Compiling With Annotations



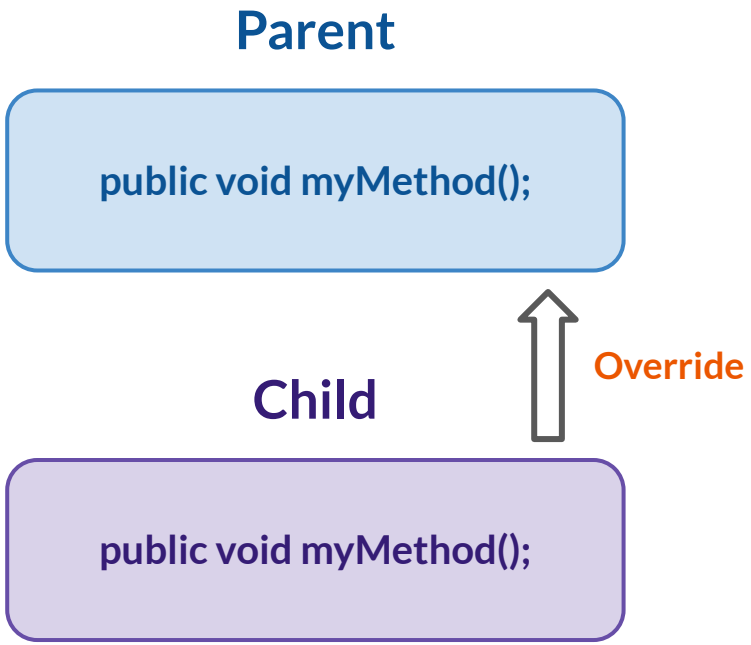
Annotations will be checked during compilation. Can stop a file from compiling if errors occur.

Types of Annotations

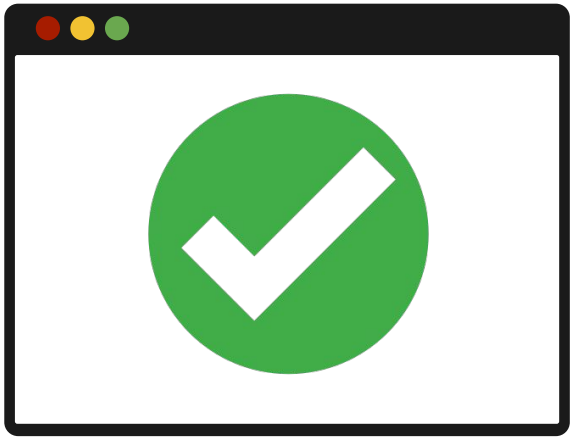
- **Marker** - take ***no parameters***, used to mark element to be processes in some way (`@Annotation`)
- **Single Element** - provide single piece of data
  - Represented with ***data=value*** pair: `@Annotation(data="data")`
  - Or with ***value only***: `@Annotation("data")`
- **Full Value or Multi-Value** - have ***multiple data members*** (`@Annotation(data1="data1" data2="data2")`)

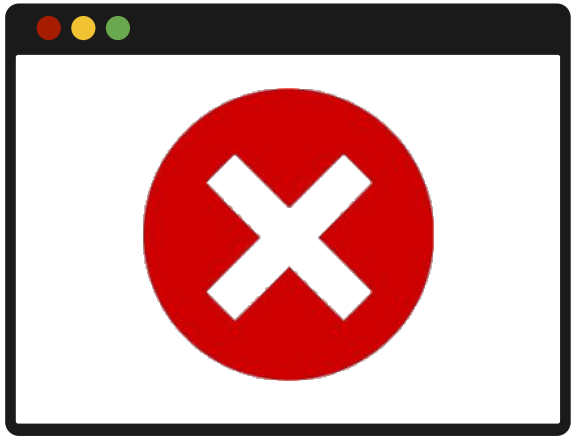
@Override Example

- Run the following file: `Annotations.java`
- Take note of how the annotation `@Override` is checking your code
- Compilation error will occur if you place this annotation on a method that is not properly overriding



JUnit & Test Driven Development





Video on TDD & JUnit

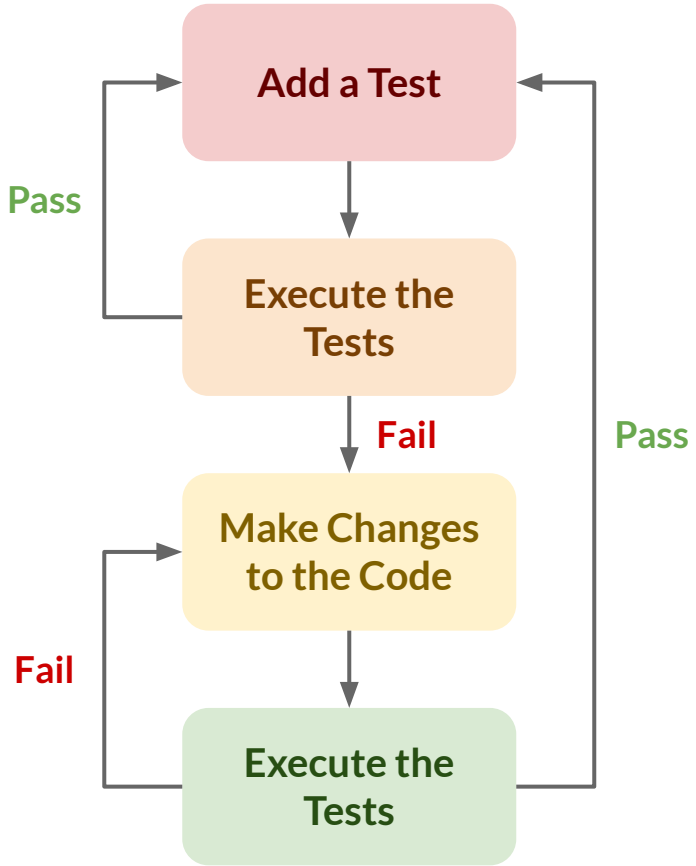
- Please review the following video on JUnit: https://youtu.be/rwQakhL3wMI
- Feel free to review attached code used in video:
  - `Calculator.java`
  - `CalculatorTest.java`
  - `DivideByZeroException.java`
- Following slides in this section can be used as additional reference

Test Driven Development

- **Test Driven Development (TDD)** is a software development approach where testing is done to validate code as it is being developed
- Test cases usually written first, once code is built they run through tests
  - **Pass Test** → no further action, move on to next test
  - **Fail Test** → refactor/rewrite code so it passes test
- Try not to write new code until test passes
  - Can avoid duplication of code, small amounts of code written at a time

Five Steps of Test-Driven Development

1. Add a **new test**
2. **Run all tests** and see if any of them fail
3. If failure occurs, rewrite some code (**refactoring**)
4. **Run tests again**, if failure occurs, do more refactoring
5. Once tests all pass, **repeat cycle** starting at step one

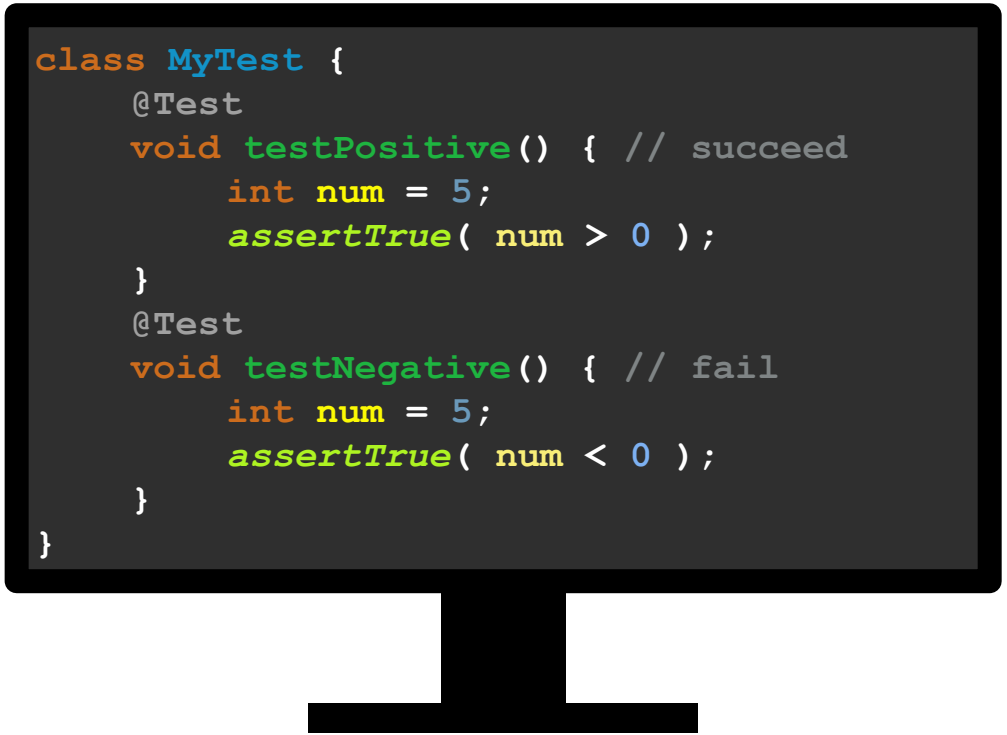


JUnit

- **JUnit** is a Unit Testing Framework supported in Java
- Can follow the flow of TDD in JUnit
  - Developers write tests before writing code
  - Easy way to run every test
- ***As code is changed, every test should be rerun*** in case something breaks with new updates
- Each test case is a method, these methods are executed one by one
- Other types of methods can be written to do setup and cleanup of resources after running tests

Test File

- JUnit test files ***don't contain a main method***
- Have ***multiple test methods***, each called by JUnit when file is run
- Test cases will fail or succeed based on if condition checked
- Conditions are checked with assertion methods from `Assert` class



Assertion Class Methods

The methods from the Assertion class can be used to test conditions in a test case. When the test file is run, if the assertion is true, will pass, if not, will fail.

assertTrue()	Succeeds if condition passed is true.
assertFalse()	Succeeds if condition passed is false.
assertSame()	Succeeds if two objects compared are the same object.
assertNotSame()	Succeeds if two objects compared are NOT the same object.
assertNull()	Succeeds if object checked is null.
assertNotNul()	Succeeds if object checked is not null.

Examples on how to use this assertions: https://www.guru99.com/junit-assert.html

Assertion Class Methods

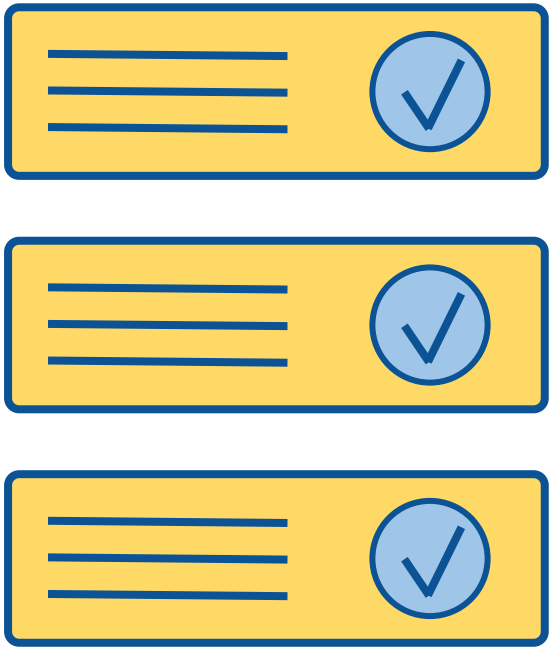
The methods from the Assertion class can be used to test conditions in a test case. When the test file is run, if the assertion is true, will pass, if not, will fail.

assertEquals()	Succeeds if two values compared are the equal. If comparing two objects, will use the equals() method to check for equality.
assertNotEquals()	Succeeds if two values compared are NOT equal. Still uses the equals() method.
assertThrows()	Succeeds if the code throws the specified exception.
fail()	Always fails if run.

Examples on how to use this assertions: https://www.guru99.com/junit-assert.html

Before & After Annotations

- When running tests, setup and cleanup may be necessary when running multiple tests
- In order to set up resources, set up `@BeforeAll`
- Any cleanup can be handled with `@AfterAll`
- Then if values need to be reset, setup required before each test, `@BeforeEach`
- Can also use `@AfterEach` if need code to run after each test

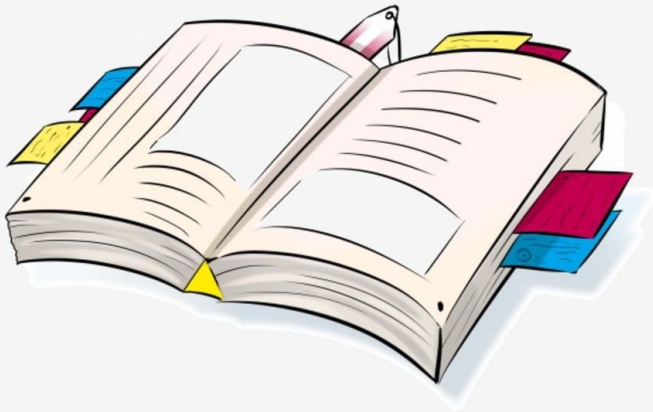


JUnit Annotations

Table Header	Table Header
`@Test`	Method will be executed as a test case. When test file is run, each method labeled with this annotation will fail or succeed.
`@BeforeAll`	Method will be run once before all the test cases are run. Useful to do any setup needed to run all your tests.
`@BeforeEach`	Method will be run before each test case.
`@AfterAll`	Method will be run once after all the test cases are run. May be used to do any cleanup or disconnect from any resources.
`@AfterEach`	Method will be run after each test case.

Open Book Quiz on Annotations & JUnit

- ➔ **Check README.md for link**
- ➔ This is an **open note**, multiple choice quiz
- ➔ Have it completed by the **start of class tomorrow at 10AM EST**
- ➔ If there are ***any questions, ask your instructor during this time or during office hours***, as they may not able available after hours



Cognixia™