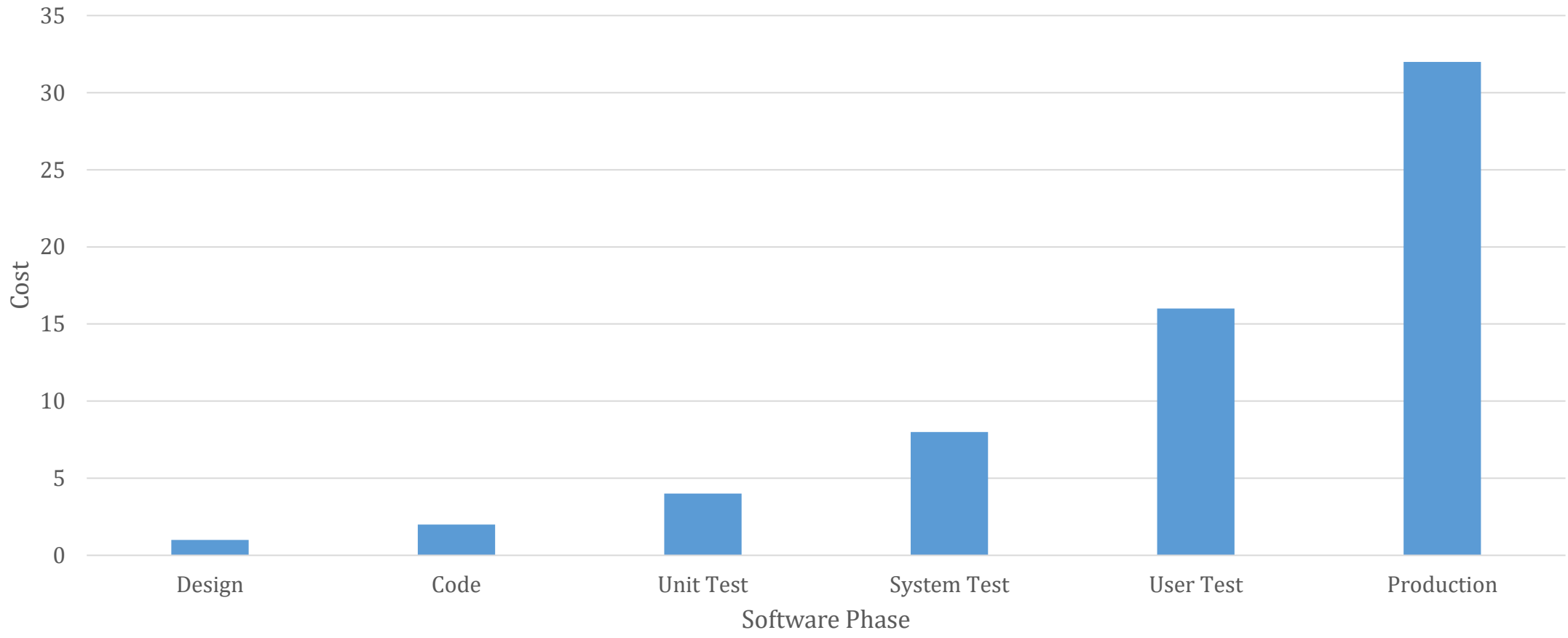
Unit Testing

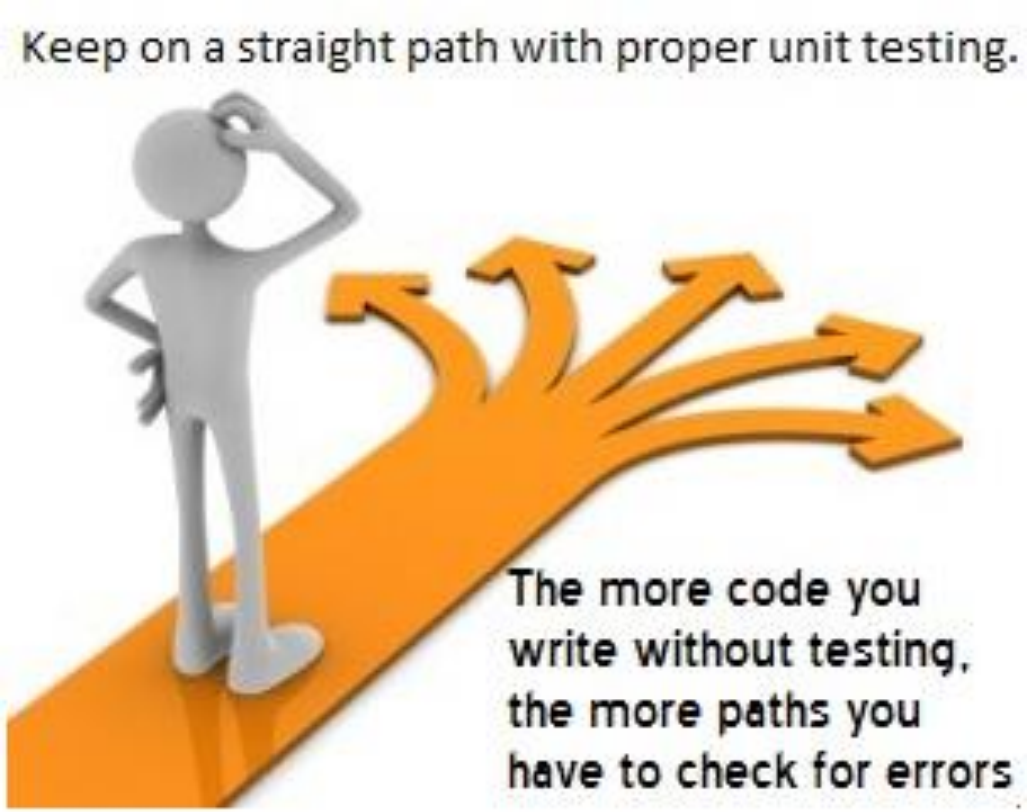
Types of Testing

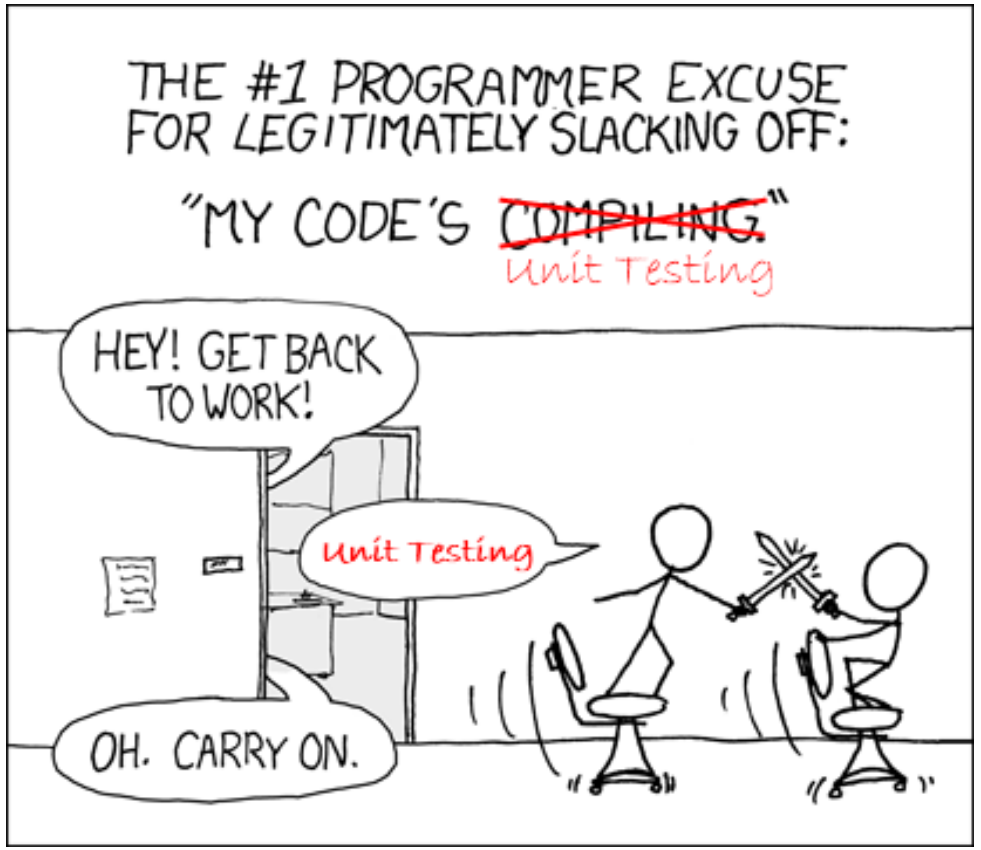
- Unit Testing
  - Testing very small functional components – e.g. methods or single classes
- System (or "Integration") Testing
  - Testing the interaction of components with each other
  - Assumes Unit Testing is complete and successful
- User (or "Beta") Test
  - Allow users to run new code in real "live" situations

Cost of Fix


35
30
25
20
15
10
5
0
Cost
Design
Code
Unit Test
System Test
User Test
Production
Software Phase

Test Early, Test Often





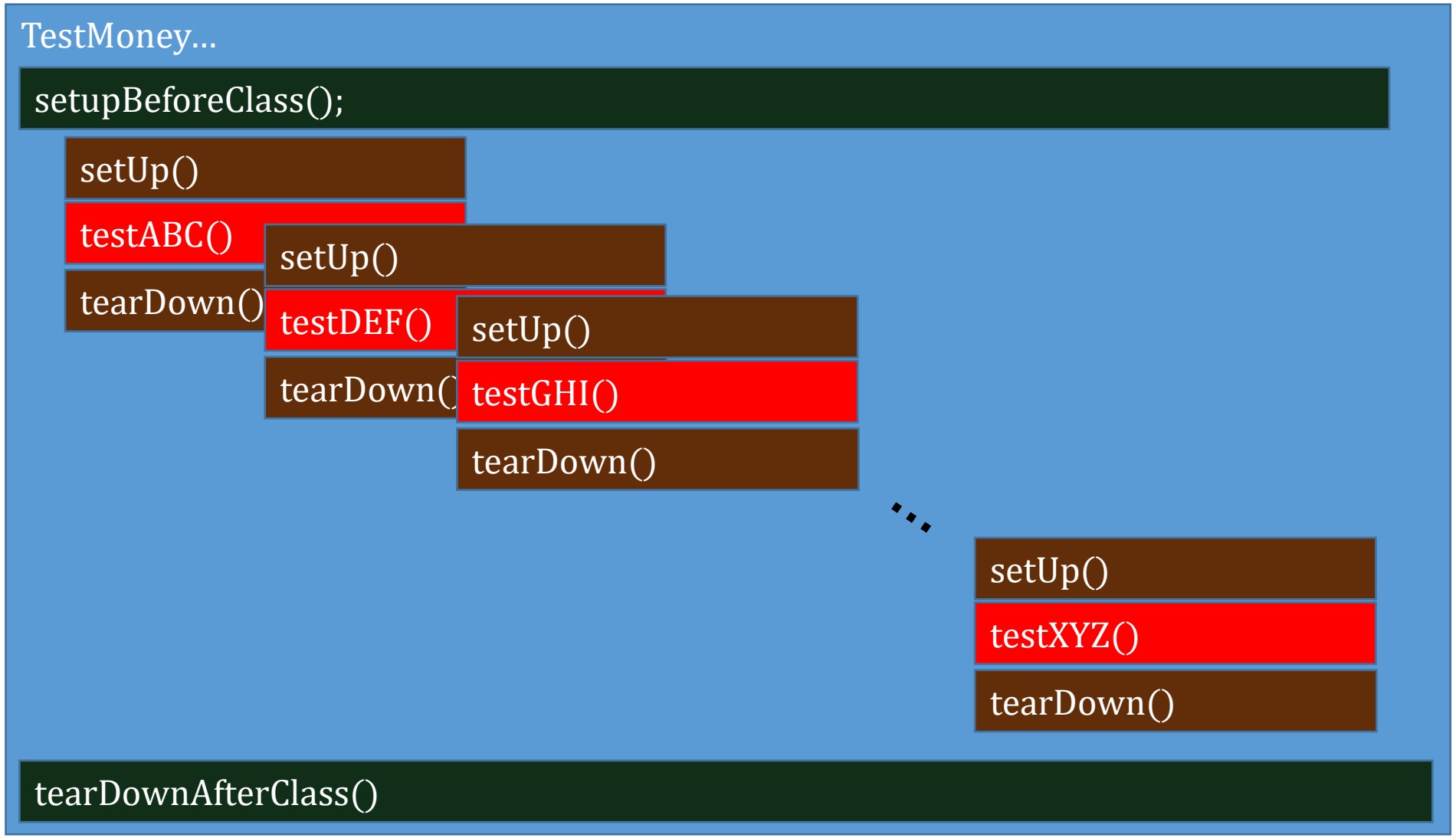
JUnit Testing

- Formal methodology to test a single Java Class
  - Typically write one or more tests for each method in a class
  - Tests prove that all reasonable method invocations run correctly
    - Typically, produce the correct return values
    - May also check for "side effects" (fields updated, etc.)
    - Method should work for extreme cases as well as typical cases
- Reproducible, even when run by somebody else, with Quantified results
- Tests easy to extend
- Supported by Eclipse

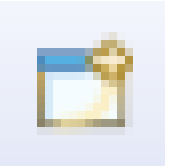
JUnit Class Organization

```
package hw03
class TestMoney {
        // All the code to test the Money class
    @beforeall static void setupBeforeClass() { // run at the start of the entire test
    }
    @beforeeach static void setUp() { // run before each test
    }
    @test void testABC() { // Unit Test for ABC
    }
    @aftereach static void tearDown() { //run each test
    }
    @afterall static void tearDownAfterClass() { // run at the end of the entire test
    }
}
```

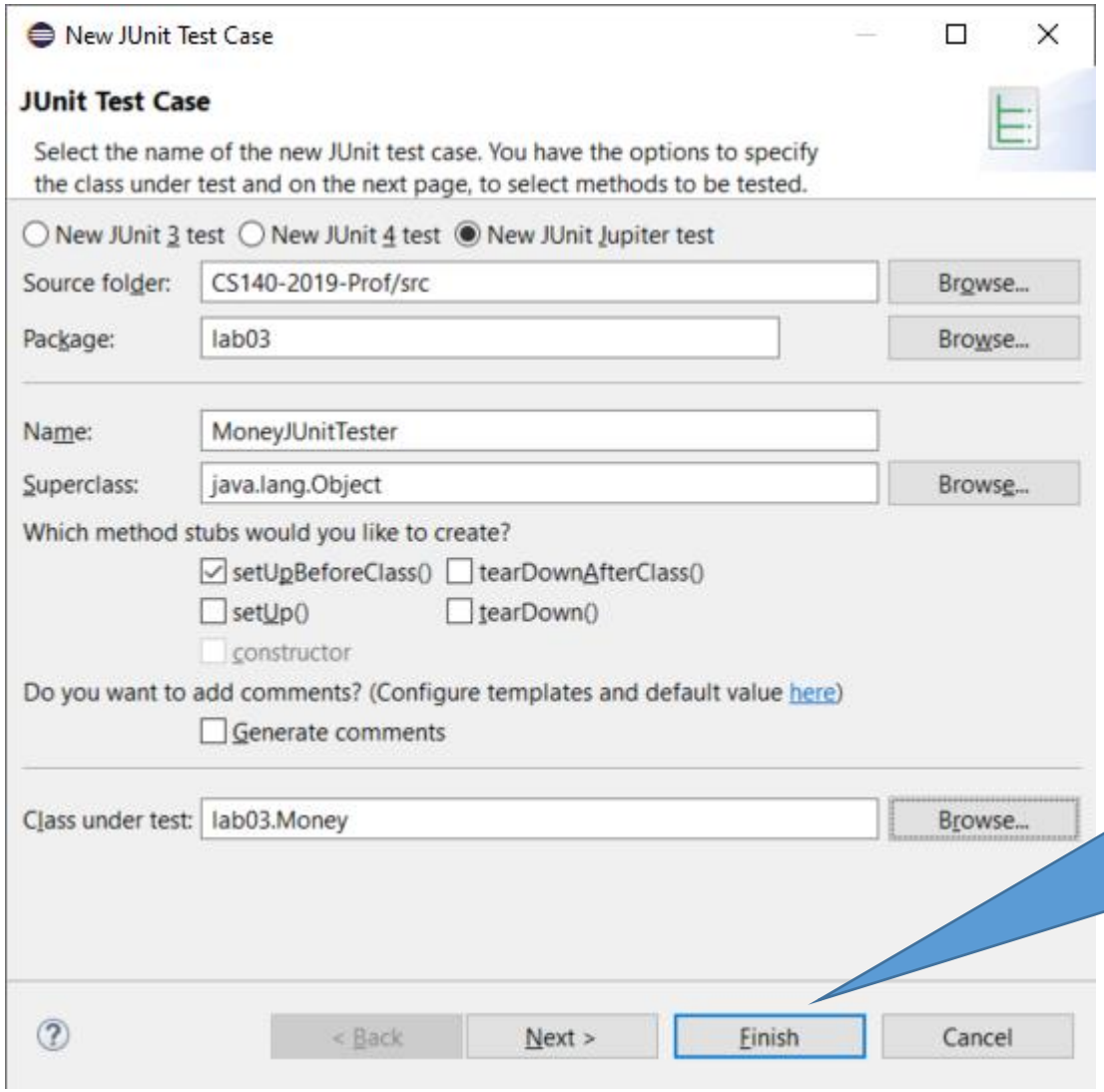
JUnit Execution



Create a Junit Test Class

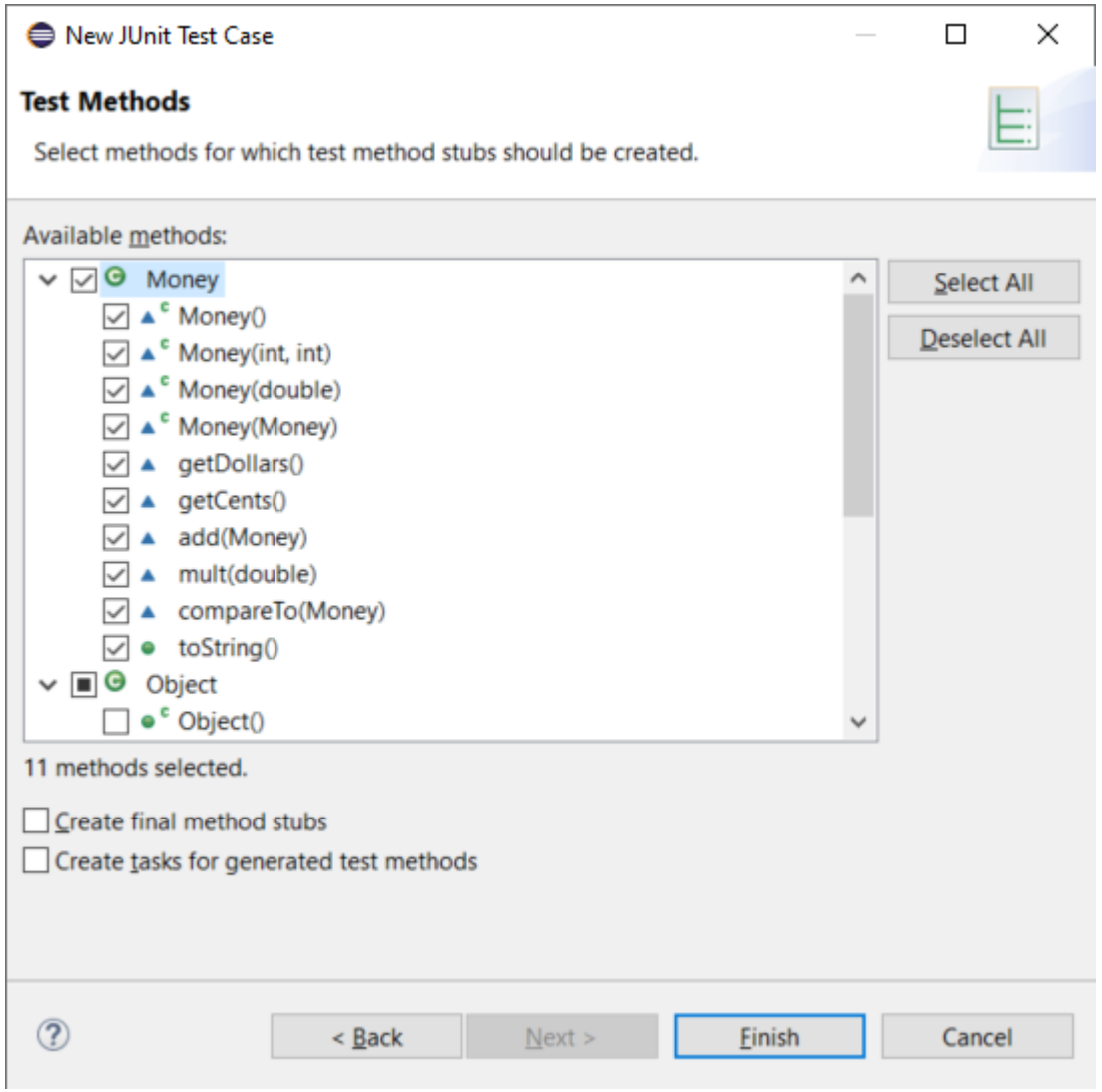
- In Eclipse,
- Select package, right click “New” (or click on the  icon)
- Select “Junit Test Case”…

Junit Test Case Wizard, Screen 1

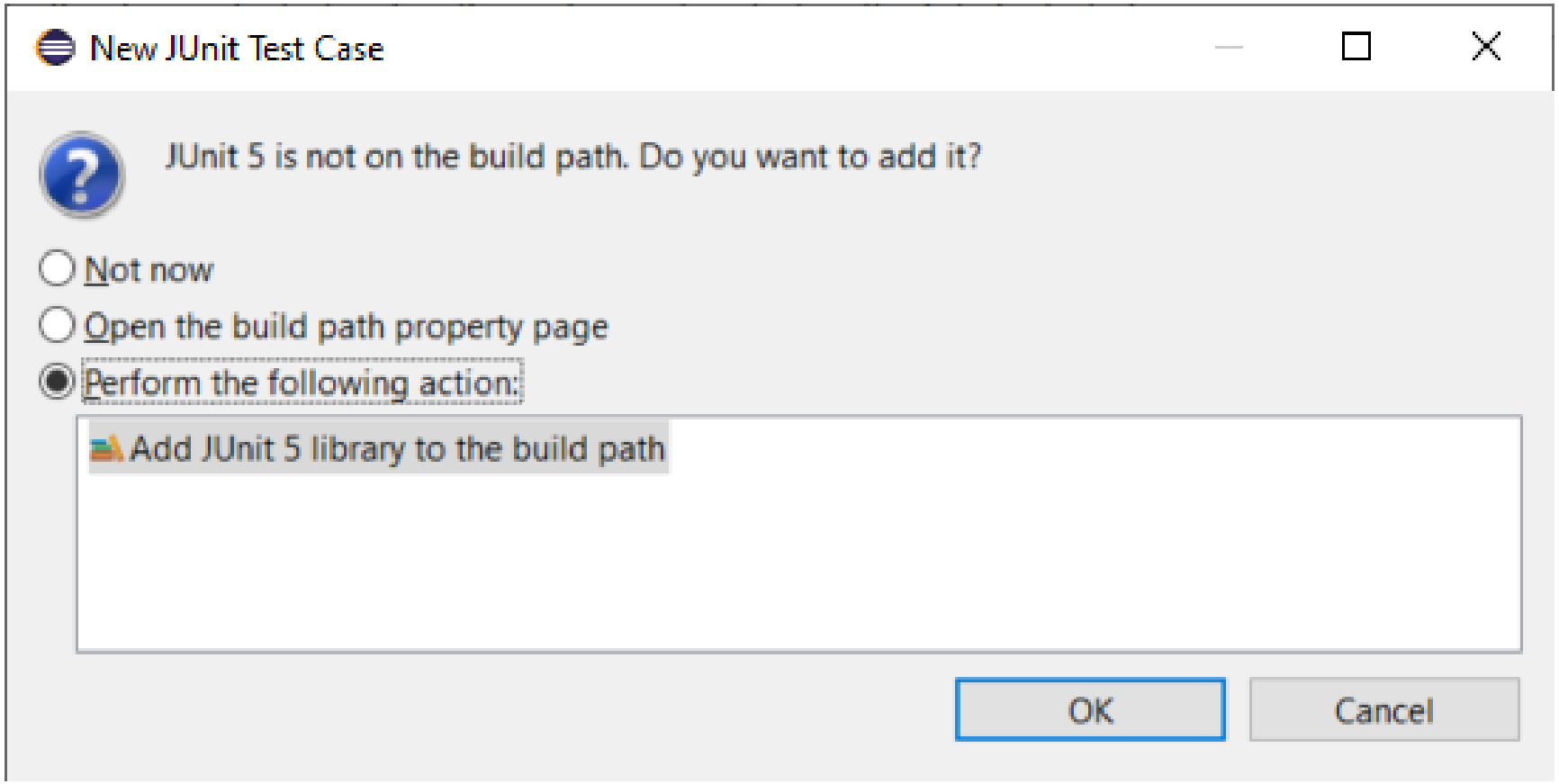


Note:
Default is "Finish",
but choose "Next"!

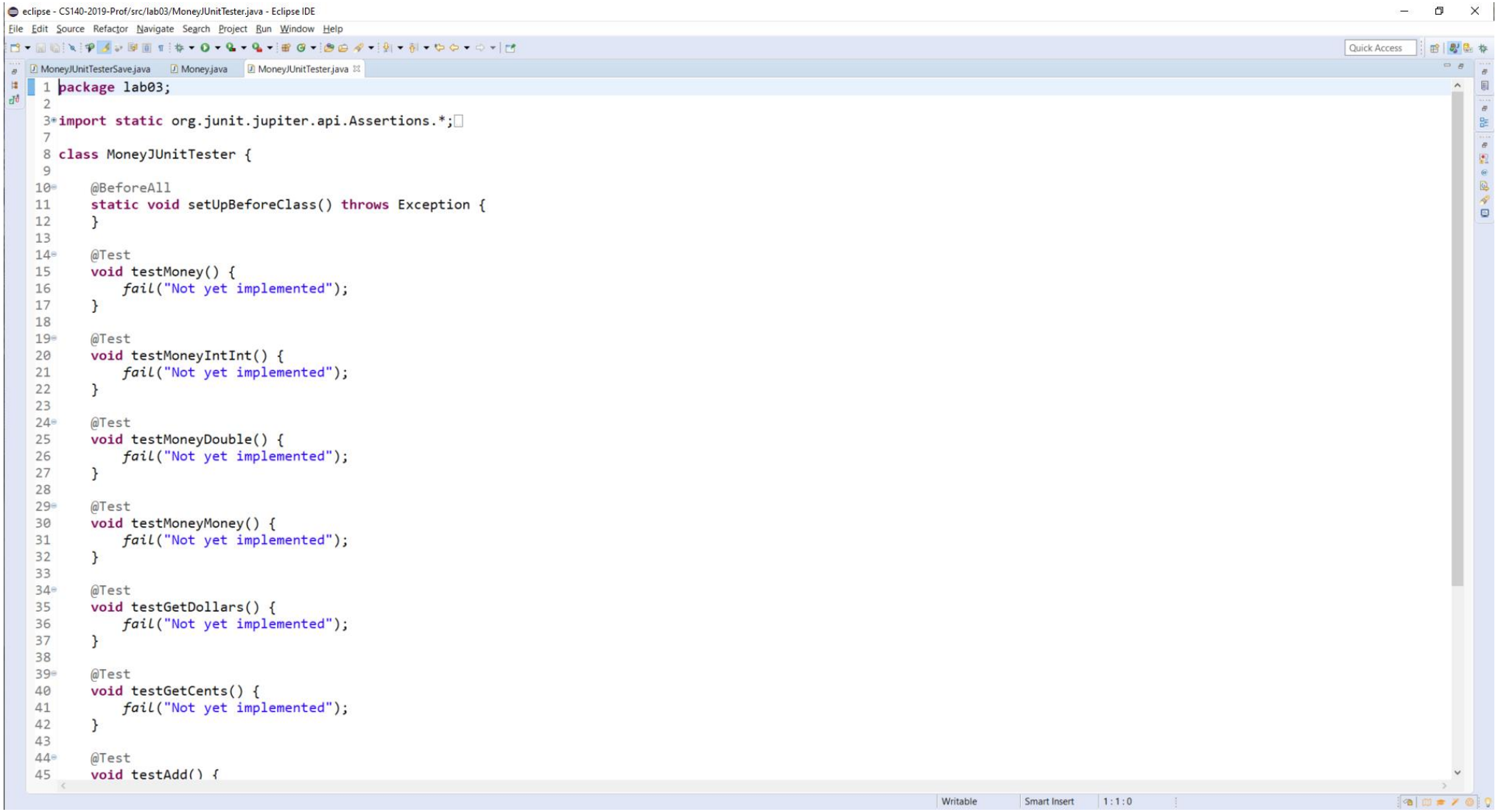
Select Methods to Test



First Time Using JUnit Testing?



Eclipse creates a Unit Test Class



Unit Test Class Organization

- Annotations on methods to define what they do
  - @Test – This is a separate “test” method
  - @BeforeEach – This method is invoked before each test
  - @AfterEach – This method is invoked after each test
  - @BeforeAll – This method is invoked before all test methods
  - @AfterAll – This method is invoked after all test methods
- Eclipse Wizard already provides these
- No "main" method… JUnit provides "main" for you
- See http://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations for more

What's in a @Test Method?

- If nothing is invoked, the test passes
- If fail(*reason*) is invoked, the test fails, for the reason specified
- There are several static "assert…" methods
  - If the assertion is true, the test passes
  - If the assertion if false, the test fails
  - Documented in http://junit.org/junit5/docs/current/api/org/junit/jupiter/api/Assertions.html

Using assertEquals

- assertEquals(expected,actual[,delta][,String reason])
  - Various different types are allowed for expected and actual
    - byte, char, double, float, short, int, long, Object
  - For floating point, third argument, "delta" for comparison
  - Final optional argument: string explaining more about what is wrong

- Note… there are also
  - assertTrue and assertFalse
  - assertNotEquals, assertNull, assertSame
  - assertArrayEquals

Testing Performance

- assertTimeout
  - First argument – Duration
  - Second argument – executable
- Fails if executable takes longer than duration
  - Does not stop until executable finishes
  - Use assertTimeoutPreemptive if you want to stop before executable finishes

Unit Test Philosophy

- Write unit tests BEFORE you write your code
- Write the code, run the tests, and fix all bugs found
- As time goes on (e.g. system testing, user testing, etc.) more bugs will be uncovered… add tests for these bugs to your unit tests as you find and fix these bugs
- Run unit test each time the code changes to ensure that your code continues to fix all bugs you have seen

Eclipse Demo

Unit Testing the hw03 Money class