

Section 4:

Graphs and Testing

Justin Bare and Deric Pang

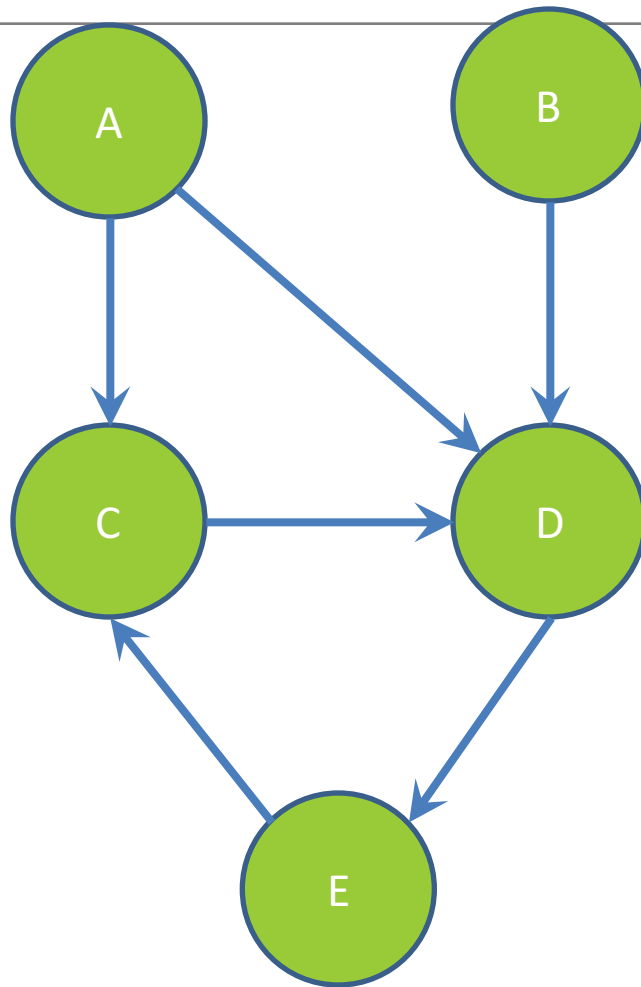
with material from Erin Peach, Nick Carney, Vinod Rathnam, Alex Mariakakis, Krysta Yousoufian, Mike Ernst, Kellen Donohue



AGENDA

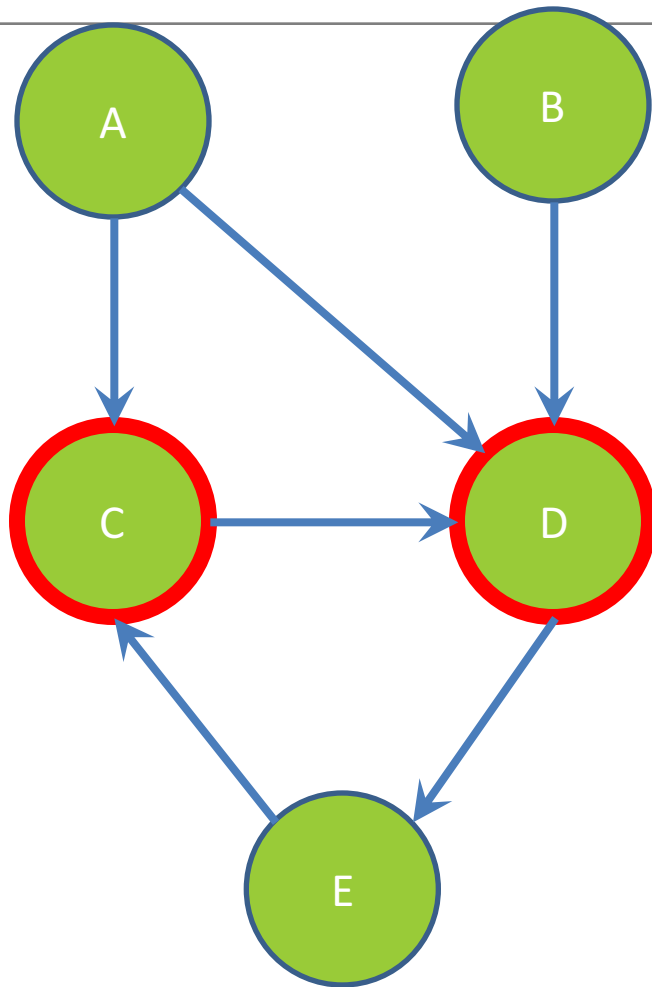
- × Graphs
- × JUnit Testing
- × Test Script Language
- × JavaDoc

GRAPHS



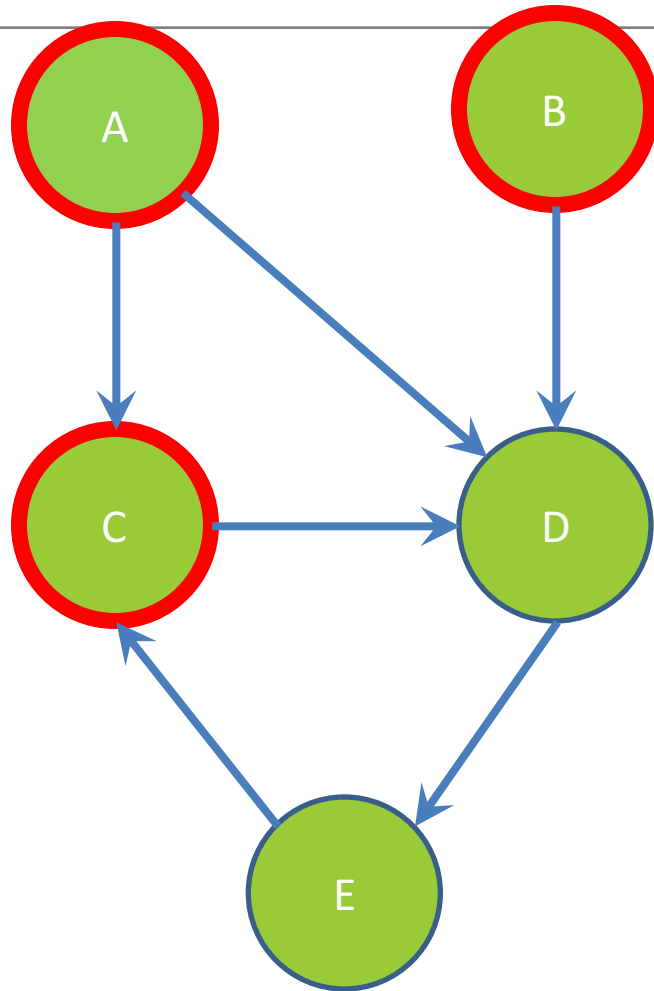
**Nodes and
Edges**

GRAPHS



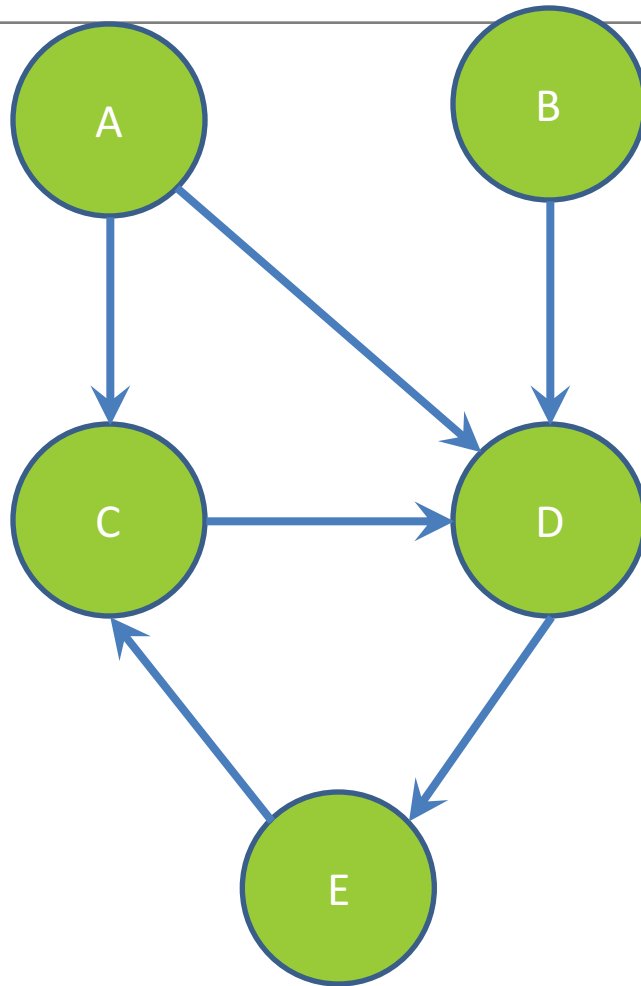
Children of A

GRAPHS



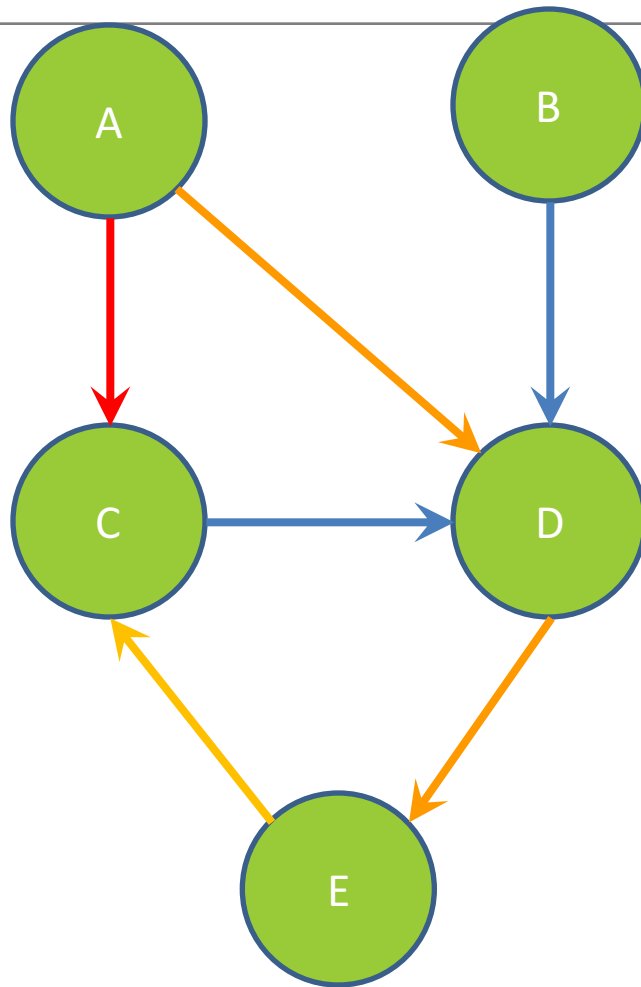
Parents of D

GRAPHS



**Paths from
A to C:**

GRAPHS



Paths from
A to C:

A -> C

A -> D -> E -> C

Shortest path
from A to C?



Testing

INTERNAL VS. EXTERNAL TESTING

× Internal : JUnit

- + How you decide to implement the object
- + Checked with implementation tests

× External: test script

- + Your API and specifications
- + Testing against the specification
- + Checked with specification tests

A JUNIT TEST CLASS

- ✗ A method with `@Test` is flagged as a JUnit test
- ✗ All `@Test` methods run when JUnit runs

```
import org.junit.*;
import static org.junit.Assert.*;

public class TestSuite {
    ...

    @Test
    public void TestName1() {
        ...
    }
}
```

USING JUNIT ASSERTIONS

- × Verifies that a value matches expectations
 - × `assertEquals(42, meaningOfLife());`
 - × `assertTrue(list.isEmpty());`
- × If the assert fails:
 - + Test immediately terminates
 - + Other tests in the test class are still run as normal
 - + Results show “details” of failed tests (We’ll get to this later)

USING JUNIT ASSERTIONS

| Assertion | Case for failure |
|--|-------------------------------------|
| <code>assertTrue(test)</code> | the boolean test is false |
| <code>assertFalse(test)</code> | the boolean test is true |
| <code>assertEquals(expected, actual)</code> | the values are not equal |
| <code>assertSame(expected, actual)</code> | the values are not the same (by ==) |
| <code>assertNotSame(expected, actual)</code> | the values are the same (by ==) |
| <code>assertNotNull(value)</code> | the given value is not null |
| <code>assertNotNull(value)</code> | the given value is null |

- And others: <http://www.junit.org/apidocs/org/junit/Assert.html>
- Each method can also be passed a string to display if it fails:
 - `assertEquals("message", expected, actual)`

CHECKING FOR EXCEPTIONS

- × Verify that a method throws an exception when it should:
 - × Passes if specified exception is thrown, fails otherwise
- × Only time it's OK to write a test without a form of asserts

```
@Test (expected=IndexOutOfBoundsException.class)  
public void testGetEmptyList() {  
    List<String> list = new ArrayList<String>();  
    list.get(0);  
}
```

“But don’t I need to create a list before checking if I’ve successfully added to it?”

SETUP AND TEARDOWN

- × Methods to run before/after each test case method is called:

@Before

```
public void name() { ... }
```

@After

```
public void name() { ... }
```

- × Methods to run once before/after the entire test class runs:

@BeforeClass

```
public static void name() { ... }
```

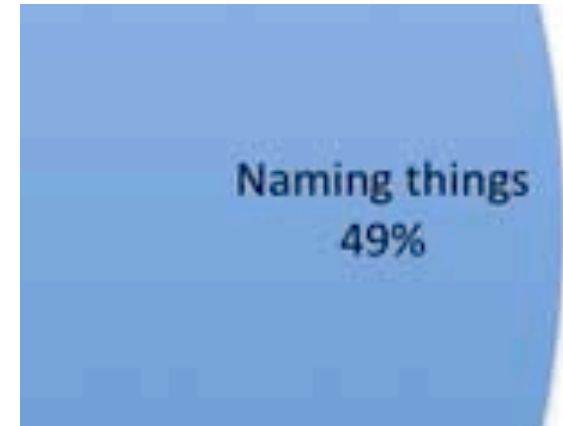
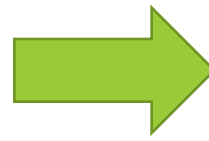
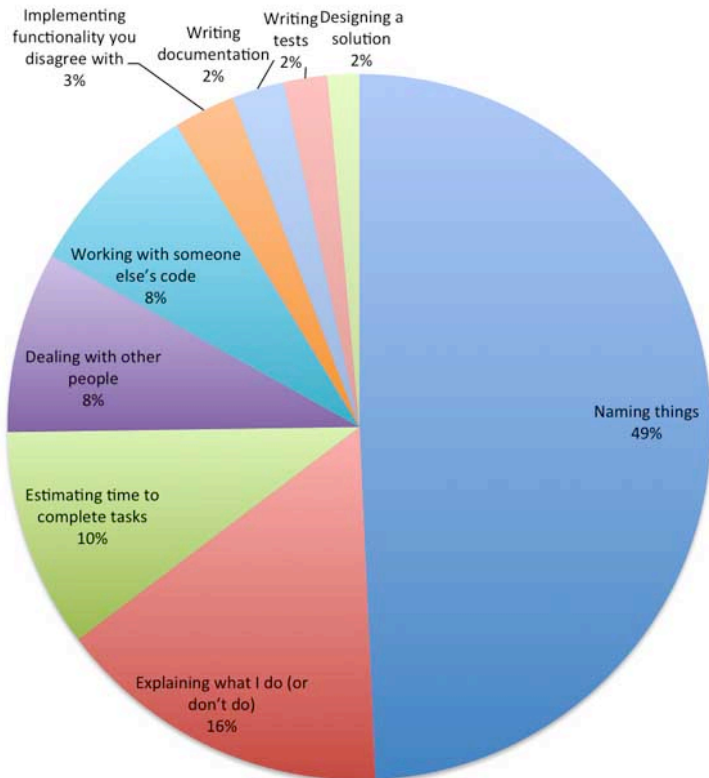
@AfterClass

```
public static void name() { ... }
```

SETUP AND TEARDOWN

```
public class Example {  
    List empty;  
  
    @Before  
    public void initialize() {  
        empty = new ArrayList();  
    }  
  
    @Test  
    public void size() {  
        ...  
    }  
  
    @Test  
    public void remove() {  
        ...  
    }  
}
```


Programmers' Hardest Tasks



Data Source: Quora/Ubuntu Forums
Total Votes: 4,522



Test Writing Etiquette

The Rules

1. Don't Repeat Yourself

- Use constants and helper methods

2. Be Descriptive

- Take advantage of message, expected, and actual values

3. Keep Tests Small

- Isolate bugs one at a time – Test halts after failed assertion

4. Be Thorough

- Test big, small, boundaries, exceptions, errors

LET'S PUT IT ALL TOGETHER!

```
public class DateTest {  
  
    ...  
    // Test addDays when it causes a rollover between months  
@Test  
    public void testAddDaysWrapToNextMonth() {  
        Date actual = new Date(2050, 2, 15);  
        actual.addDays(14);  
        Date expected = new Date(2050, 3, 1);  
        assertEquals("date after +14 days", expected,  
                    actual);  
    }  
}
```

How To Create JUnit Test Classes

- ✗ Right-click hw5.test -> New -> JUnit Test Case
- ✗ **Important:** Follow naming guidelines we provide
- ✗ Demo

JUNIT ASSERTS VS. JAVA ASSERTS

- × We've just been discussing JUnit assertions so far
- × Java itself has assertions

```
public class LitterBox {
    ArrayList<Kitten> kittens;

    public Kitten getKitten(int n) {
        assert(n >= 0);
        return kittens(n);
    }
}
```

ASSERTIONS VS. EXCEPTIONS

```
public class LitterBox {
    ArrayList<Kitten> kittens;

    public Kitten getKitten(int n) {
        assert(n >= 0);
        return kittens(n);
    }
}
```

```
public class LitterBox {
    ArrayList<Kitten> kittens;

    public Kitten getKitten(int n) {
        try {
            return kittens(n);
        } catch(Exception e) {
        }
    }
}
```

- ✗ Assertions should check for things that should never happen
- ✗ Exceptions should check for things that might happen
- ✗ “Exceptions address the robustness of your code, while assertions address its correctness”

REMINDER: ENABLING ASSERTS IN ECLIPSE

To enable asserts:

Go to Run -> Run Configurations... -> Arguments
tab -> input **-ea** in VM arguments section

Do this for every test file

Expensive CheckReps

- ✗ Ant Validate and Staff Grading will have assertions enabled
- ✗ But sometimes a checkRep can be expensive
 - ✗ For example, looking at each node in a Graph with a large number of nodes
- ✗ This could cause the grading scripts to timeout

Expensive CheckReps

- ✗ Before your final commit, remove the checking of expensive parts of your checkRep or the checking of your checkRep entirely
- ✗ Example: boolean flag and structure your checkRep as so:

```
private void checkRep() {  
    cheap-stuff  
    if(DEBUG_FLAG) { // or can have this for entire checkRep  
        expensive-stuff  
    }  
    cheap-stuff  
    ...  
}
```

EXTERNAL TESTS: TEST SCRIPT LANGUAGE

TEST SCRIPT LANGUAGE

- × Text file with one command listed per line
- × First word is always the command name
- × Remaining words are arguments
- × Commands will correspond to methods in your code

TEST SCRIPT LANGUAGE (ex .test file)

Create a graph

```
CreateGraph graph1
```

Add a pair of nodes

```
AddNode graph1 n1
```

```
AddNode graph1 n2
```

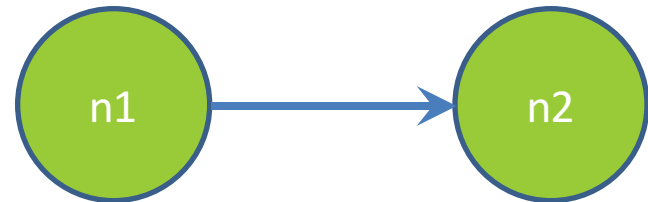
Add an edge

```
AddEdge graph1 n1 n2 e1
```

Print the nodes in the graph
and the outgoing edges from n1

```
ListNodes graph1
```

```
ListChildren graph1 n1
```



How To Create Specification Tests

- ✗ Create .test and .expected file pairs under hw5.test
- ✗ Implement parts of HW5TestDriver
 - + driver connects commands from .test file to your Graph implementation to the output which is matched with .expected file
- ✗ Run all tests by running SpecificationTests.java
 - + Note: staff will have our own .test and .expected pairs to run with your code
 - + **Do not** hardcode .test/.expected pairs to pass, but instead make sure the format in hw5 instructions is correctly followed

DEMO: TEST SCRIPT LANGUAGE



JAVADOC API

- × Now you can generate the JavaDoc API for your code
- × Instructions in the Editing/Compiling Handout
- × Demo: Generate JavaDocs