SECTION 3
GRAPHS & TESTING

Agenda
- Graphs
- Testing

Graph
= collection of nodes (vertices) and edges

Nodes: states or objects within the graph

Edges: connection between two nodes

Directed graph vs Undirected graph
- Directed graph
  = edges have a source and destination
- Arrows as edges
- Parent and child nodes related by an edge

Some examples

Linked Lists

Binary Trees

Directed graph vs Undirected graph
- Directed
- Undirected

What are some examples?
Directed graph vs Undirected graph

Directed:
- Build systems
- Course prerequisites

Undirected:
- Facebook friends
- Map of U-District Restaurants

Cyclic vs Acyclic

Special type of graphs: Directed Acyclic Graphs (DAGs)
Why do we need them?

Worksheet

What is Testing?
- Implementation tests
- Specification tests

When to do which one?

Implementation vs. Specification

- Implementation tests:
  - How you decide to implement the object.
  - See if each component (unit) is working properly.

- Specification tests:
  - Testing your API against the specifications.
  - Usually larger than unit tests.

Black box vs. Clear box

- Black box:
  - Written with knowledge of only the Specification.

- Clear box:
  - Written with full knowledge of an implementation.
Worksheet

A JUnit test class (Demo)

- A method with @Test is a JUnit test.
- All @Test methods run when JUnit runs.

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

public class TestSuite {
    @Test
    public void Test1() { ... }
}
```

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 still run.
  - Results show information about failed tests.

Using JUnit assertions

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Case for failure</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>assertTrue(test)</code></td>
<td>the boolean test is false</td>
</tr>
<tr>
<td><code>assertFalse(test)</code></td>
<td>the boolean test is true</td>
</tr>
<tr>
<td><code>assertEquals(expected, actual)</code></td>
<td>the values are not equal</td>
</tr>
<tr>
<td><code>assertNotSame(expected, actual)</code></td>
<td>the values are the same (by ==)</td>
</tr>
<tr>
<td><code>assertNull(value)</code></td>
<td>the given value is not null</td>
</tr>
<tr>
<td><code>assertNotNull(value)</code></td>
<td>the given value is null</td>
</tr>
</tbody>
</table>

- And others: [http://www.junit.org/apidocs/org/junit/Assert.html](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 (Demo)

- Verify that a method throws an exception when it should:
  - Passes only if specified exception is thrown
- Only time it's OK to write a test without a form of `assert`:

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

Setup and teardown

- Methods to run before/after each test case method is called:
  ```java
  @Before
  public void name() { ... }
  @After
  public void name() { ... }
  ```
- Methods to run once before/after the entire test class runs:
  ```java
  @BeforeClass
  public static void name() { ... }
  @AfterClass
  public static void name() { ... }
  ```
Setup and teardown

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

Ground 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; failing assertion halts test
4. Be Thorough
   - Test big, small, boundaries, exceptions, errors

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:

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

Summary

- **Demo** will be uploaded
- **JUnit documentation** online
- Reminder: you can generate the **JavaDoc API** for your code
  - Located under `build/docs/javadoc` in project folder.
  - IntelliJ Gradle Instructions in the Editing/Compiling Handout.