Automated Testing

Finding Bugs

Testing Principles

Check compliance with requirements
Check compliance after changes (regression)
Methodology to isolate problems
Reduce costs of maintenance

Good practices

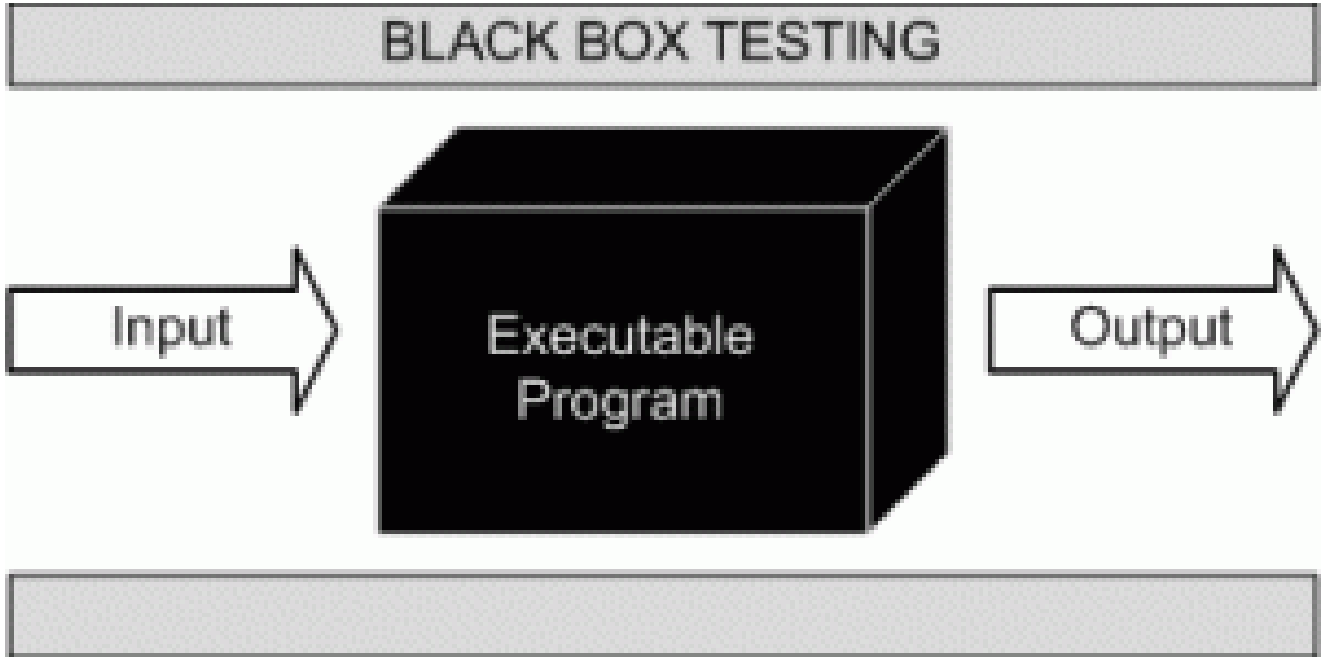
Requirements are clearly testable
Components are easily divisible into individual pieces
Write tests offline
Create tests for ranges of values and boundaries
Create tests for types
Create tests for multiple states
Think of all the extremes

Functional
Behavioral
Performance
Integration
System

Types of Test

What types of tests we could write

Functional Test



Triangle Example

A programmer wrote the following function for a math library. Come up with some test cases for the function defined below. Hint: there can be up to 21

```
public static String TriangleType(int sideA, int sideB, int sideC);
```

Triangle Example

A programmer wrote the following function for a math library. Come up with some test cases for the function defined below. Hint: there can be up to 21

```
public static String TriangleType(int sideA, int sideB, int sideC);
```

Behavioral Test (state)

```
int posX;
int posY;

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

Performance Testing

```
long startTime = System.nanoTime();
DrawScreen();
long estimatedTime = System.nanoTime() - startTime
```

Integration Testing

```
Setup Sprites

Bomberman.placeBomb();
for(AllSprites){
    if(SpritesLocatedAtBlast){
        kill sprites
    }
}

if(Correct Number of Sprites Died)
{
    Test Passed!
}else{
    Test Failed!
}
```

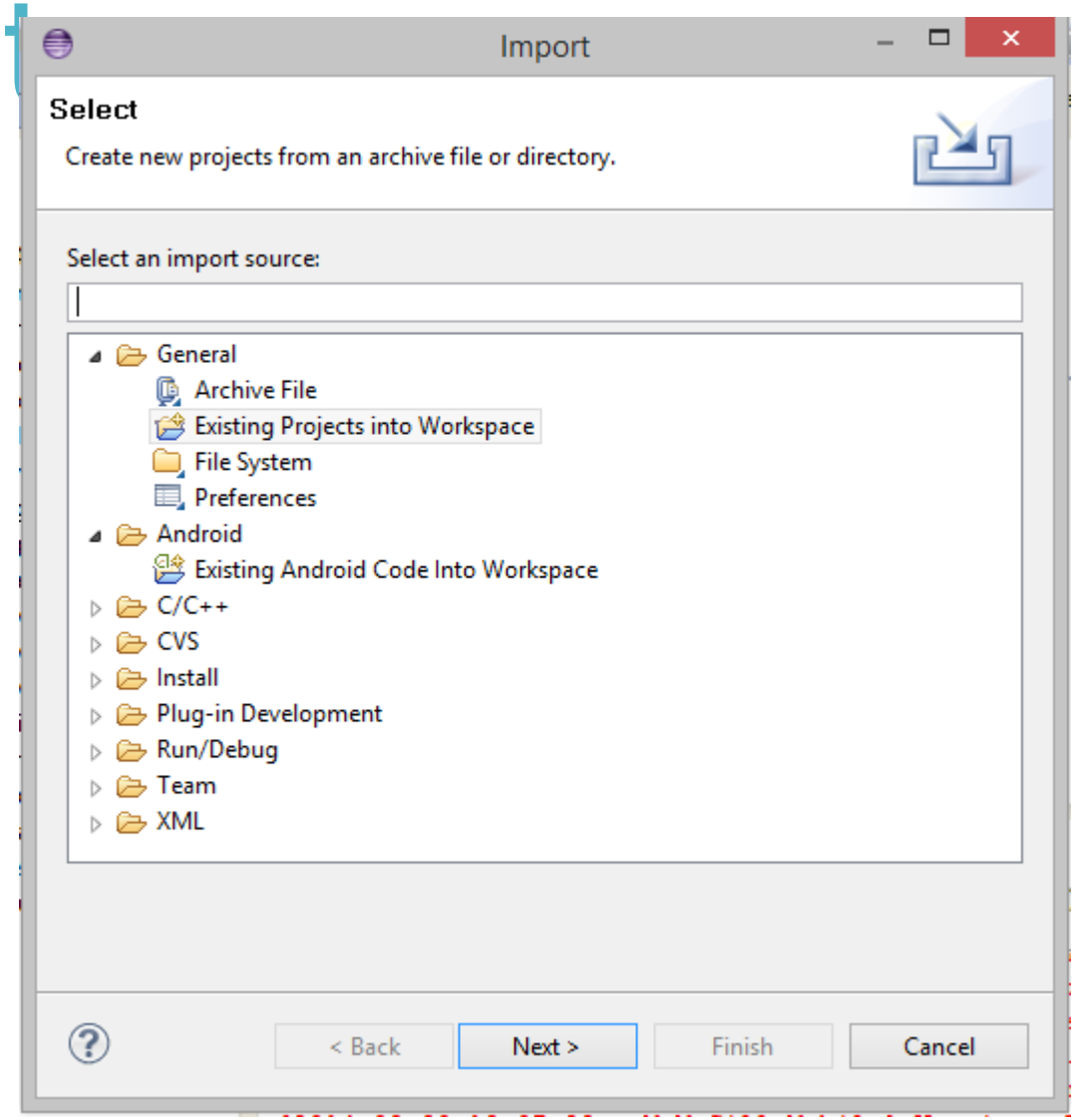
Getting Set up

JUnit

Testing Library

Importing a project

File -> Import -> Existing project into Workspace -> Browse for project



Setting up a Test

File -> new-> Junit Test case

Junit Structure

```
 8  import Banking.*;
 9
10  public class CustomerTest {
11
12      Customer c;
13      int initialCash = 1000;
14      String awesomePassword = "JamesBrownRocks";
15
16      @Before
17      public void setUp() throws Exception {
18          c = new Customer(initialCash, awesomePassword);
19      }
20
21      @Test
22      public void test() {
23          int takeAmount = 100;
24          int amountTaken = c.takeCash(takeAmount);
25
26          assertTrue("Take equals Taken", takeAmount == amountTaken);
27      }
28
29  }
30
```

Import Classes under test

Declare Objects under Test

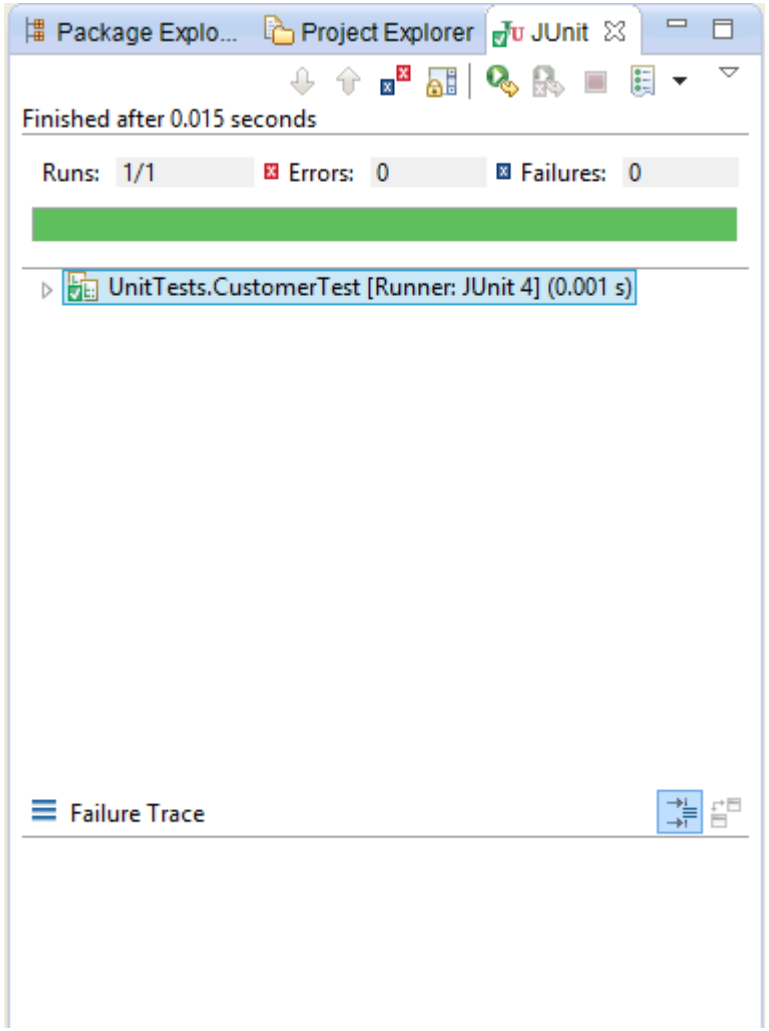
Flag to execute this method before test

Flag to indicate this method is a test case

Assertion decides if test will pass or fail

Running Tests

Right Click on Test class -> Run as -> JunitTest



Junit Annotations

Annotation	Description
@Test public void method()	The `@Test` annotation identifies a method as a test method.
@Test (expected = Exception.class)	Fails if the method does not throw the named exception.
@Test(timeout=100)	Fails if the method takes longer than 100 milliseconds.
@Before public void method()	This method is executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class).
@After public void method()	This method is executed after each test. It is used to cleanup the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.
@BeforeClass public static void method()	This method is executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as `static` to work with JUnit.
@AfterClass public static void method()	This method is executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as `static` to work with JUnit.
@Ignore	Ignores the test method. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included.

Junit Assertions

Statement	Description
fail(String)	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The String parameter is optional.
assertTrue([message], boolean condition)	Checks that the boolean condition is true.
assertFalse([message], boolean condition)	Checks that the boolean condition is false.
assertEquals([String message], expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([String message], expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message], object)	Checks that the object is null.
assertNotNull([message], object)	Checks that the object is not null.
assertSame([String], expected, actual)	Checks that both variables refer to the same object.
assertNotSame([String], expected, actual)	Checks that both variables refer to different objects.

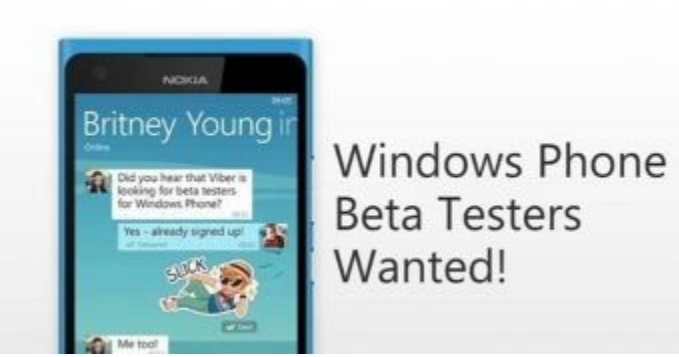
Concept questions

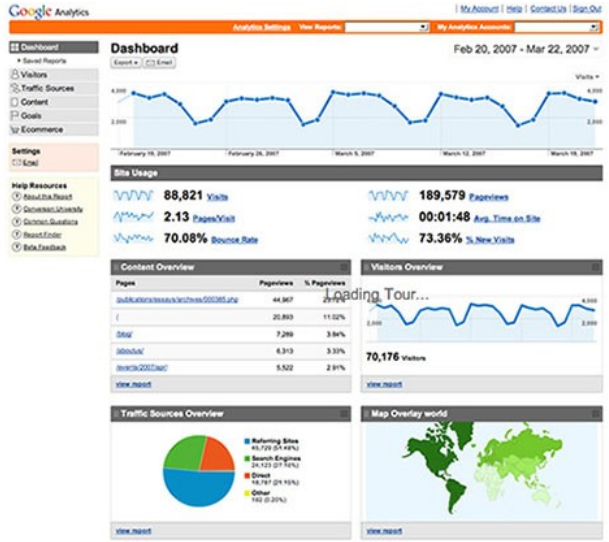
You work for a company that sells a desktop app that millions of customers use. 10% of your customer base complains that the app crashes every Tuesday morning. How do you start your investigation?

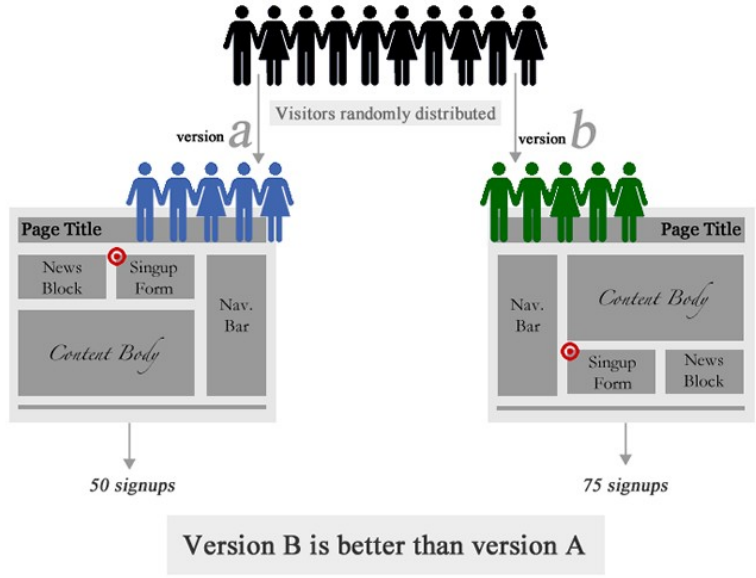
Concept questions

You are writing a java program and notice a bug. You put a print statement in your code to diagnose the bug. The bug disappeared. What could have happened

Alternative Testing Methods







Alternative Testing Methods