# Unit Testing

# Demo

# Testing Strings

# Testing Strings

This test **fails**!

- When Testing Strings:
  - **NEVER** use ==
  - This will be true for all non-primitive comparisons
- Using == checks if the two values store the same reference
- Strings can be.. weird.

```java
package week3;

public class Strings {

    public static String combineStrings(String a, String b) {
        return a + " " + b;
    }

}
```

```java
package week3;

import org.junit.Test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class StringsTests {

    @Test
    public void testStringsBadExample() {
        String a = "hello";
        String b = "goodbye";
        assertTrue(Strings.combineStrings(a, b) == "hello goodbye");
    }

}
```

# Testing Strings

- Test Strings using the equals method
  - Compares the *values* of the Strings
- This test passes 😀

```java
package week3;

public class Strings {

    public static String combineStrings(String a, String b) {
        return a + " " + b;
    }

}
```

```java
package week3;

import org.junit.Test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class StringsTests {

    @Test
    public void testStringsGoodExample() {
        String a = "hello";
        String b = "goodbye";
        assertTrue(Strings.combineStrings(a, b).equals("hello goodbye"));
    }

}
```

# Testing Strings

- We can take this a step further by using assertEquals
- assertEquals will call the equals methods on the arguments you provide
- Provides more feedback if the test fails

```java
package week3;

public class Strings {

    public static String combineStrings(String a, String b) {
        return a + " " + b;
    }

}
```

```java
package week3;

import org.junit.Test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class StringsTests {

    @Test
    public void testStringsBetterExample() {
        String a = "hello";
        String b = "goodbye";
        assertEquals(Strings.combineStrings(a, b), "hello goodbye");
    }

}
```

```java
package week2;

public class PlusMinus {
    public static String letter(int score){
        int tens=score/10;
        if (tens>=9){
            return "A";
        } else if(tens>=8){
            return "B";
        } else if(tens>=7){
            return "C";
        } else if(tens>=6){
            return "D";
        } else {
            return "F";
        }
    }
    public static String plusMinus(int score){
        int ones=score%10;
        if (ones>=7){
            return "+";
        } else if (ones>2){
            return "";
        } else {
            return "-";
        }
    }
    public static void main(String[] args) {
        System.out.println(letter(95));
        System.out.println(letter(78));
        System.out.println(letter(51));
    }

}
```

# Testing Strings

- Let's expand our letter grade example to include plusses and minuses
- The plusMinus method should return the appropriate value "+", "-", or "" for the input
  - 87-89 -> B+
  - 83-86 -> B
  - 80-82 -> B-

# Testing Strings

```java
package week2;

public class PlusMinus {

    public static String letter(int score){
        int tens=score/10;
        if (tens>=9){
            return "A";
        } else if(tens>=8){
            return "B";
        } else if(tens>=7){
            return "C";
        } else if(tens>=6){
            return "D";
        } else {
            return "F";
        }
    }

    public static String plusMinus(int score){
        int ones=score%10;
        if (ones>=7){
            return "+";
        } else if (ones>2){
            return "";
        } else {
            return "-";
        }
    }

}
```

- To test the plusMinus method, we'll write a test class
- This is a good start with 3 test cases (We would write a lot more for better testing)
- Using assertEquals to compare our Strings
- We run the test and our code passes! 😀

```java
public class PlusMinusTests {

    @Test
    public void testPlusMinus() {
        String pm = PlusMinus.plusMinus(95);
        assertEquals("95 There should be no +-, got: " + pm, pm, "");
        pm = PlusMinus.plusMinus(78);
        assertEquals("78 It should be +, got: " + pm, pm, "+");
        pm = PlusMinus.plusMinus(51);
        assertEquals("51 It should be -, got: " + pm, pm, "-");
    }
}
```

# Testing Strings

```
package week2;

public class PlusMinus {

    public static String letter(int score){
        int tens=score/10;
        if (tens>=9){
            return "A";
        } else if(tens>=8){
            return "B";
        } else if(tens>=7){
            return "C";
        } else if(tens>=6){
            return "D";
        } else {
            return "F";
        }
    }

    public static String plusMinus(int score){
        int ones=score%10;
        if (ones>=7){
            return "+";
        } else if (ones>2){
            return "";
        } else {
            return "-";
        }
    }

}
```

- Let's add one more test to be sure. We'll check the edge case of 100

- Run the test again with the new test case

```
public class PlusMinusTests {

    @Test
    public void testPlusMinus() {
        String pm = PlusMinus.plusMinus(95);
        assertEquals("95 There should be no +-, got: " + pm, pm, "");
        pm = PlusMinus.plusMinus(78);
        assertEquals("78 It should be +, got: " + pm, pm, "+");
        pm = PlusMinus.plusMinus(51);
        assertEquals("51 It should be -, got: " + pm, pm, "-");
        pm = PlusMinus.plusMinus(100);
        assertEquals("100 It should be +, got: " + pm, pm, "+");
    }
}
```

# Testing Strings

```java
package week2;

public class PlusMinus {

    public static String letter(int score){
        int tens=score/10;
        if (tens>=9){
            return "A";
        } else if(tens>=8){
            return "B";
        } else if(tens>=7){
            return "C";
        } else if(tens>=6){
            return "D";
        } else {
            return "F";
        }
    }

    public static String plusMinus(int score){
        int ones=score%10;
        if (ones>=7){
            return "+";
        } else if (ones>2){
            return "";
        } else {
            return "-";
        }
    }

}
```

- Aaaand.. the test fails! 😭
- The poor student with 100 was given an A-!! We have a bug!
- We passed 3/4 tests: but we, and the student with an A-, demand perfection

```java
public class PlusMinusTests {

    @Test
    public void testPlusMinus() {
        String pm = PlusMinus.plusMinus(95);
        assertEquals("51 It should be -, got: " + pm, pm, "-");
        pm = PlusMinus.plusMinus(100);
        assertEquals("100 It should be +, got: " + pm, pm, "+");
    }
}
```

```
org.junit.ComparisonFailure: 100 It should be +, got: -
Expected :-
Actual   :+
```

# Testing Strings

```java
package week2;

public class PlusMinus {

    public static String letter(int score){
        int tens=score/10;
        if (tens>=9){
            return "A";
        } else if(tens>=8){
            return "B";
        } else if(tens>=7){
            return "C";
        } else if(tens>=6){
            return "D";
        } else {
            return "F";
        }
    }

    public static String plusMinus(int score){
        int ones=score%10;
        if (ones>=7){
            return "+";
        } else if (ones>2){
            return "";
        } else {
            return "-";
        }
    }

}
```

- The goal of unit testing is to expose any bugs that may exist
- This unit test did a great job at exposing a bug
- Write unit tests for every possible bug you can think of
- Edit your code until it passes all your tests

```java
public class PlusMinusTests {

    @Test
    public void testPlusMinus() {
        String pm = PlusMinus.plusMinus(95);
        assertEquals("95 There should be no +-, got: " + pm, pm, "");
        pm = PlusMinus.plusMinus(78);
        assertEquals("78 It should be +, got: " + pm, pm, "+");
        pm = PlusMinus.plusMinus(51);
        assertEquals("51 It should be -, got: " + pm, pm, "-");
        pm = PlusMinus.plusMinus(100);
        assertEquals("100 It should be +, got: " + pm, pm, "+");
    }
}
```

```java
package week2;

public class PlusMinus {

    public static String letter(int score){
        int tens=score/10;
        if (tens>=9){
            return "A";
        } else if(tens>=8){
            return "B";
        } else if(tens>=7){
            return "C";
        } else if(tens>=6){
            return "D";
        } else {
            return "F";
        }
    }

    public static String plusMinus(int score){
        if(score==100){
            return "+";
        }
        int ones=score%10;
        if (ones>=7){
            return "+";
        } else if (ones>2){
            return "";
        } else {
            return "-";
        }
    }

}
```

# Testing Strings

- That's better
- Edge cases will often have special conditions in your code
- This code passes our test cases and the student gets the A+ they've earned

# Testing Doubles

# Testing Doubles

- This test fails.
- Why??

```java
package week3;

public class Doubles {

    public static double timesThree(double x) {
        return x * 3.0;
    }

}
```

```java
public class DoublesTests {

    @Test
    public void testDoublesBad() {
        double y = Doubles.timesThree(0.1);
        System.out.println(y);
        assertTrue(y == 0.3);
    }

}
```

# Testing Doubles

- If we print 0.1*3.0
- We get 0.30000000000000004
- Which is not == 0.3

```
package week3;

public class Doubles {

    public static double timesThree(double x) {
        return x * 3.0;
    }

}
```

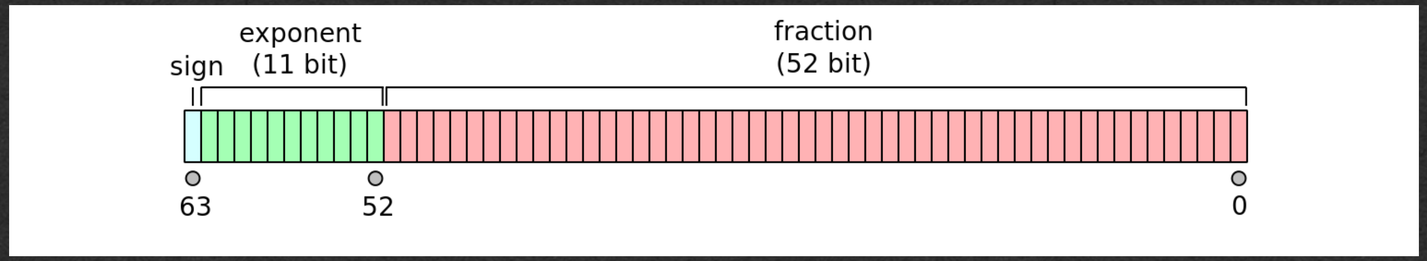
```
public class DoublesTests {

    @Test
    public void testDoublesBad() {
        double y = Doubles.timesThree(0.1);
        System.out.println(y);
        assertTrue(y == 0.3);
    }

}
```

# Testing Doubles

- A double is stored using a 64 bit representation
- If the number doesn't fit in those 64 bits, it must be truncated
- We lose precision when 64 bits is not enough




https://en.wikipedia.org/wiki/Double-precision_floating-point_format

# Testing Doubles

- Decimal values are represented in binary as fractions of powers of 2
  - Ex. 0.11 == 1/2 + 1/4 == 3/4
- In decimal we have values that cannot be stored without truncation
  - Ex. 1/3 != 0.33333333333333333333333333333333333333333333333333
- Values such as 0.1 cannot be represented as a sum of powers of 2
  - 0.1 (base 10) != 0.0001100110011001100110011001100110011001100110011001 (base 2)
  - 64 bits is not enough to store an infinitely repeating decimal. We must truncate

# Testing Doubles

- The solution?
  - Allow for some tolerance to accept doubles that are within truncations errors of each other
- Check that the difference between the doubles is less than some small number

```java
package week3;

public class Doubles {

    public static double timesThree(double x) {
        return x * 3.0;
    }

}
```

```java
public class DoublesTests {

    private final double EPSILON = 0.001;

    public void compareDoubles(double d1, double d2) {
        assertTrue(Math.abs(d1 - d2) < EPSILON);
    }

    @Test
    public void testDoublesGood() {
        double y = Doubles.timesThree(0.1);
       compareDoubles(y, 0.3);
    }

}
```

# Testing Doubles

- We define the small number as a constant using the final keyword
  - Constants should be named with all capital letters
- This is our first private variable
  - It cannot be used outside of this class

```
package week3;

public class Doubles {

    public static double timesThree(double x) {
        return x * 3.0;
    }

}
```

```
public class DoublesTests {

    private final double EPSILON = 0.001;

    public void compareDoubles(double d1, double d2) {
        assertTrue(Math.abs(d1 - d2) < EPSILON);
    }

    @Test
    public void testDoublesGood() {
        double y = Doubles.timesThree(0.1);
       compareDoubles(y, 0.3);
    }

}
```

# Testing Doubles

- Choose a small number that is:
  - Large enough to allow for truncation errors
  - Small enough to not interfere with the test (eg. 10.0 will pass code that is off by 9.9)
  - Can be different for different applications

```
package week3;

public class Doubles {

    public static double timesThree(double x) {
        return x * 3.0;
    }

}
```

```
public class DoublesTests {

    private final double EPSILON = 0.001;

    public void compareDoubles(double d1, double d2) {
        assertTrue(Math.abs(d1 - d2) < EPSILON);
    }

    @Test
    public void testDoublesGood() {
        double y = Doubles.timesThree(0.1);
       compareDoubles(y, 0.3);
    }

}
```

# Testing Doubles

- Be sure to take the **absolute value** of the difference
  - If d1 is 5.0 and d2 is 1000000.0
  - The difference is -999995.0 which is less than 0.001!
  - The absolute value of the difference is 999995.0 which is much greater than 0.001 and correctly fails the test

```
package week3;

public class Doubles {

    public static double timesThree(double x) {
        return x * 3.0;
    }

}
```

```
public class DoublesTests {

    private final double EPSILON = 0.001;

    public void compareDoubles(double d1, double d2) {
        assertTrue(Math.abs(d1 - d2) < EPSILON);
    }

    @Test
    public void testDoublesGood() {
        double y = Doubles.timesThree(0.1);
       compareDoubles(y, 0.3);
    }

}
```

# Testing Doubles

- Unit has an assertEquals method designed specifically for doubles
- Takes a 3rd argument that is the allowed tolerance
  - Same as EPSILON in the previous example

```
package week3;

public class Doubles {

    public static double timesThree(double x) {
        return x * 3.0;
    }

}
```

- Simpler to use
- Whatever you do:
  - NEVER use == to compare doubles

```
public class DoublesTests {

    @Test
    public void testDoublesBetter() {
        double y = Doubles.timesThree(0.1);
       assertEquals(y, 0.3, 0.001);
    }

}
```