Floating Point Assertion in JUnit

# 1. Why Floating Point Assertion is Necessary

Floating point numbers (float, double) cannot always be compared directly using '==' or 'assertEquals(expected, actual)' because of precision errors that arise due to the binary representation of decimals. This is why JUnit provides an overloaded method:  
  
assertEquals(expected, actual, delta);  
  
Here, 'delta' is the maximum allowable difference between the expected and actual values.

actual is between expected - delta and expected + delta (inclusive).

# 2. Example: When Direct Comparison Fails

Consider this case:  
double result = 0.1 + 0.2;  
System.out.println(result); // prints: 0.30000000000000004  
  
Using 'assertEquals(expected, actual)' fails because 0.3 != 0.30000000000000004.

Correct way:  
assertEquals(0.3, 0.1 + 0.2, 1e-9); // Passes

# 3. How to Decide the Delta Parameter

Choosing the right delta depends on the context of your calculation.

* a) Understand Your Data's Precision Needs
* - High-precision (physics, cryptography): delta = 1e-9 or smaller
* - Medium-precision (financial, percentages): delta = 1e-6 to 1e-4
* - Low-precision (UI, approximate): delta = 1e-2 or larger
* b) Relative to Expected Value
* - delta = expected \* 1e-6; allows relative comparison
* c) Empirical Observation
* - Log (actual - expected) to estimate delta
* d) Avoid Zero Delta
* - delta = 0.0 behaves like '=='; avoid unless exact match is guaranteed
* E) find by subtracting expected – actual to know about delta

# 4. Recommended Delta Values (Examples)

Context | Suggested Delta

-------------------------------|------------------

0.1 + 0.2 to 0.3 | 1e-9

Interest rate calculation | 1e-6

Comparing 3.14159 to Pi | 1e-5

Game physics (position errors) | 1e-2

# 5. Summary

- Delta is the acceptable error margin for your scenario.  
- Too small: test fails due to floating point precision.  
- Too large: test may miss real bugs.  
- Start with 1e-6 if unsure, and adjust based on use-case.

Identity Assertion in Junit

Important Note: For primitive wrappers (Integer, String, etc.)

Be careful with auto-boxing and string literals, as they can sometimes be optimized by the JVM to point to the same instance for small values/interned strings.

For example, Integer i1 = 10; Integer i2 = 10; might result in i1 == i2 due to caching.

This is why assertSame is typically used for custom objects or when strict identity matters.

Here are the types of assertions in unit testing, formatted as text for your document:

**Types of Assertions in Unit Testing (Common Assertion Methods)**

Assertions are statements in unit tests that verify if a certain condition is true. If an assertion fails, it indicates a bug or an unexpected state in the code being tested. Most unit testing frameworks provide a rich set of assertion methods, typically falling into these categories:

1. **Equality Assertions:**
   * assertEquals(expected, actual): Checks if two values are equal.
   * assertNotEquals(unexpected, actual): Checks if two values are *not* equal.
2. **Boolean Assertions:**
   * assertTrue(condition): Checks if a given condition evaluates to true.
   * assertFalse(condition): Checks if a given condition evaluates to false.
3. **Nullity Assertions:**
   * assertNull(object): Checks if an object reference is null.
   * assertNotNull(object): Checks if an object reference is *not* null.
4. **Identity/Same Instance Assertions:**
   * assertSame(expected, actual): Checks if two object references point to the *exact same object* in memory.
   * assertNotSame(unexpected, actual): Checks if two object references do *not* point to the same object.
5. **Array/Collection Assertions:**
   * assertArrayEquals(expectedArray, actualArray): Checks if two arrays are equal (element by element).
   * (Many libraries also offer methods for List, Set, Map for containment, size, order, etc., e.g., contains, hasSize, isEmpty).
6. **Exception Assertions:**
   * assertThrows(expectedExceptionType, executable): Verifies that a specific piece of code throws an expected exception.
   * (Some frameworks also have assertDoesNotThrow.)

In unit testing, sometimes you **expect a method to throw an exception** under certain conditions.  
For example: dividing by zero, null input, or invalid user input.

JUnit 5 provides the method:

assertThrows(expectedException.class, () -> {

// code that should throw the exception

});

This is called an **exception assertion**. You're testing that **an exception is thrown as expected**.

1. **Floating-Point Assertions (with Delta):**
   * assertEquals(expected, actual, delta): When comparing floating-point numbers (float, double), this checks if the actual value is within a specified delta (tolerance) of the expected value, accounting for precision issues.
2. **String Assertions:**
   * (Frameworks often provide specific assertions for strings, such as startsWith, endsWith, containsString, matchesRegex, isBlank, isEmpty, etc.)
3. **Timeout Assertions:**
   * assertTimeout(duration, executable): Asserts that a given code block completes its execution within a specified time duration.
   * (assertTimeoutPreemptively is a variation that aborts execution if the timeout is exceeded.)