Software Testing

JUnit Tests in NetBeans IDE

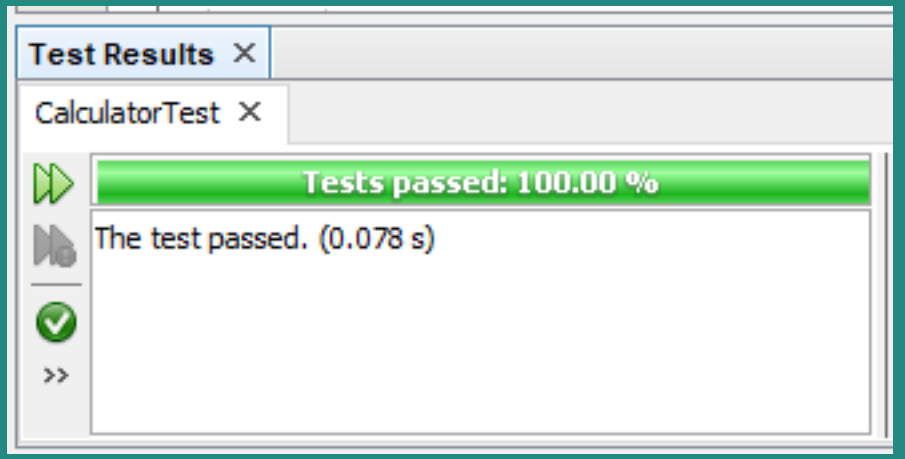
www.junit.org

What is JUnit?

- JUnit is a simple, open source framework to write and run repeatable tests.
- JUnit home page
  - http://junit.org.
- FAQs
  - https://junit.org/junit4/faq.html
- JUnit 5 User Guide
  - https://junit.org/junit5/docs/current/user-guide/

JUnit Tests Example

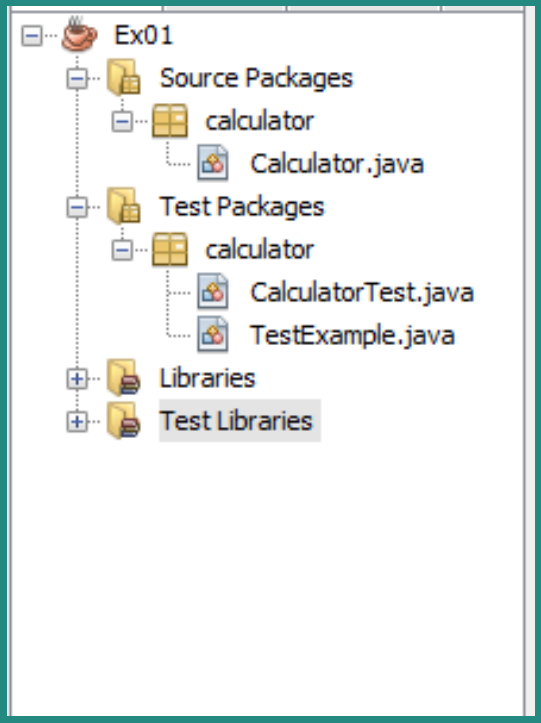
```
public class Calculator {
    public int add(int a, int b){
        return a+b;
    }

    public int max(int a, int b){
        if (a>=b) return a;
        else return b;
    }
}
```

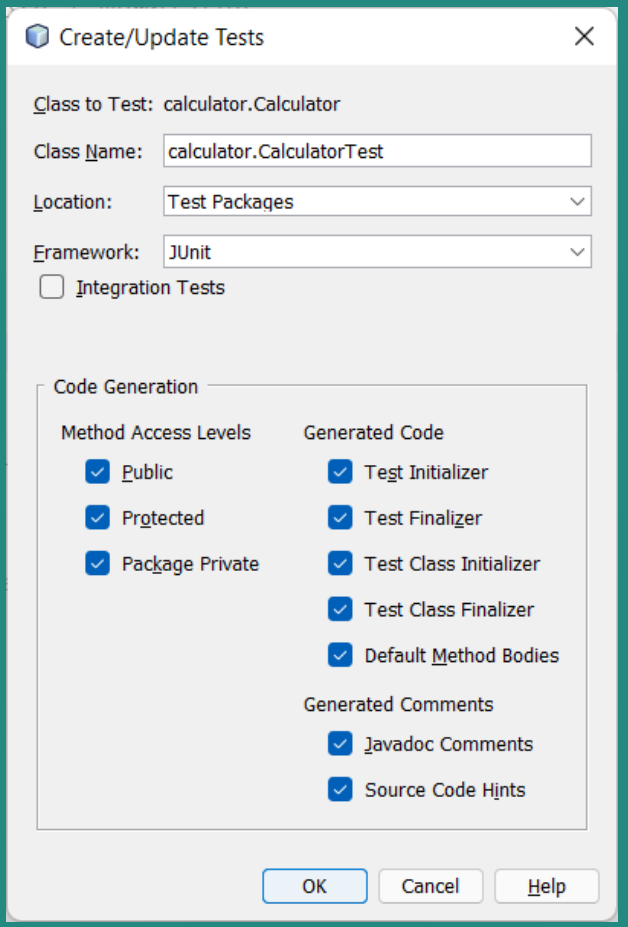



```
@Test
public void testAdd() {
    Calculator cal = new Calculator();
    int a=1, b=2; // input
    int expResult = 3; // expected output
    int actResult = cal.add(a, b); // actual output
    assertEquals(expResult, actResult); // result: pass/fail
}
```

Creating the Project (1)

- Creating the Java Class Library Project
- Creating a Test Class Xxx.java
  - **Right-click** Xxx.java → Tools > Create Tests.





Creating the Project (2)

```
19  public class CalculatorTest {
20
21      public CalculatorTest() {
22      }
23
24      @BeforeClass
25      public static void setUpClass() {
26      }
27
28      @AfterClass
29      public static void tearDownClass() {
30      }
31
32      @Before
33      public void setUp() {
34      }
35
36      @After
37      public void tearDown() {
38      }
39
40      @Test
41      public void testAdd() {
42      }
43  }
```

public void method can be executed as a test case.

JUnit 4 Annotations

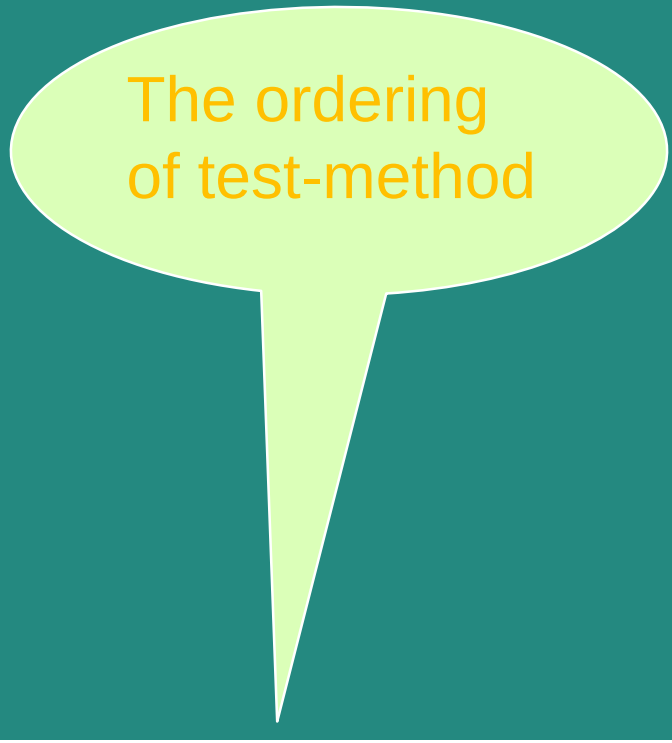
- @Test
  - ***public void method*** can be executed as a test case.
- @Test(timeout)
  - enforce timeout in JUnit4 test case
- @Test(expected)
  - check for specified exception during its execution
- @Before @BeforeClass
- @After @AfterClass
- @Ignore

setUp and tearDown method

- **setUp method**
  - Using @Before, executed before each test cases
  - Using @BeforeClass , executed before all test cases
    - and prior to any @Before methods
- **tearDown method**
  - Using @After, executed after each test cases
  - Using @AfterClass, executed after all test cases

A test class example (1)

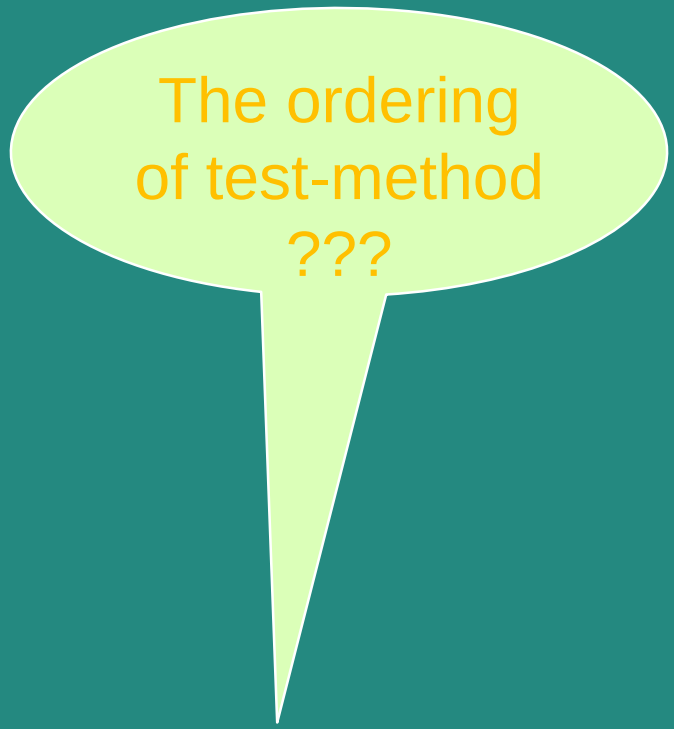
```
7. public class SimpleTest {
8.
9.     private Collection<Object> collection;
10.
11.    @Before
12.    public void setUp() {
13.        collection = new ArrayList<Object>();
14.    }
15.
16.    @Test
17.    public void testEmptyCollection() {
18.        assertTrue(collection.isEmpty());
19.    }
20.
21.
22.    @Test
23.    public void testOneItemCollection() {
24.        collection.add("itemA");
25.        assertEquals(1, collection.size());
26.    }
27. }
```




```
1. setUp()
2. testEmptyCollection()
3. setUp()
4. testOneItemCollection()
```

A test class example (2)

```
 7. public class OutputTest {
 8.
 9.     private File output;
10.
11.     @Before
12.     public void createOutputFile() {
13.         output = new File(...);
14.     }
15.
16.     @After
17.     public void deleteOutputFile() {
18.         output.delete();
19.     }
20.
21.     @Test
22.     public void testSomethingWithFile() {
23.         ...
24.     }
25. }
```




```
1. createOutputFile()
2. testSomethingWithFile()
3. deleteOutputFile()
```

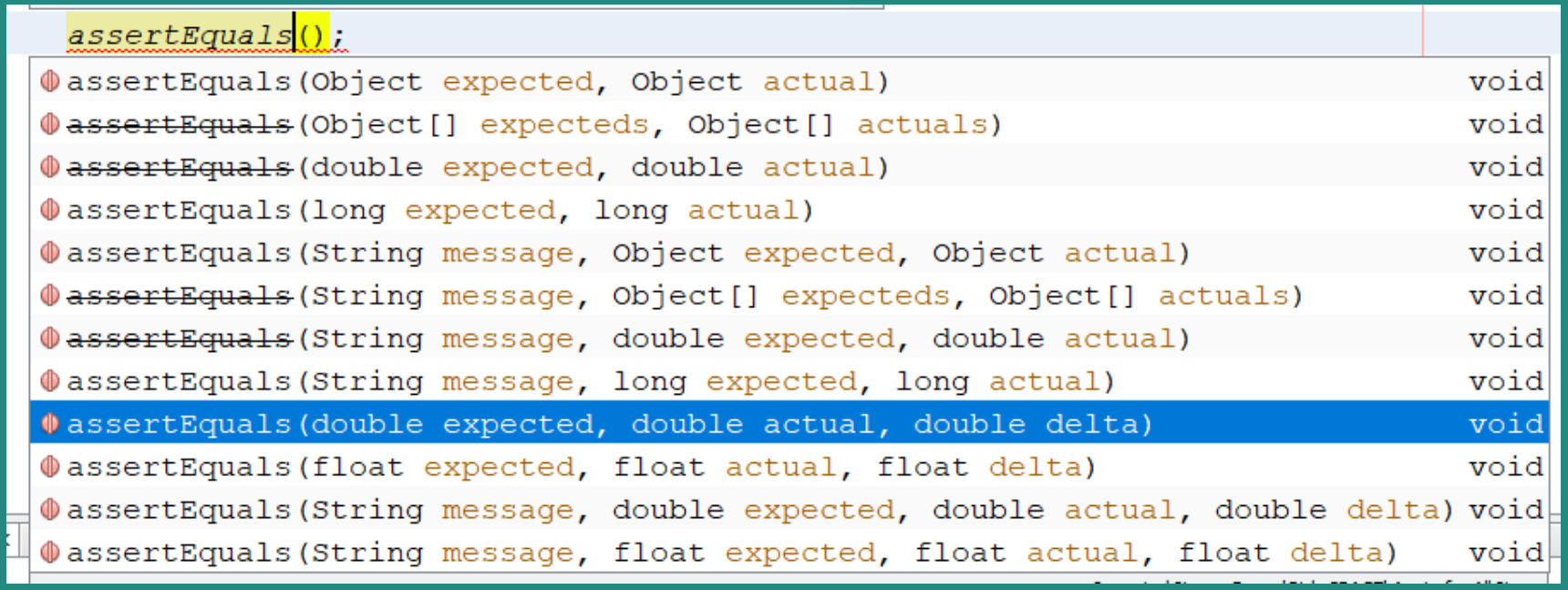
assertEquals

```
@Test
public void assertEqualsTest1() {
    String expected = "Thanh";
    String actual = "ThanhNT";
    assertEquals(expected, actual);
}

@Test
public void assertEqualsTest2() {
    String expected = "Thanh";
    String actual = "ThanhNT";
    // display a message when the assertion fails
    assertEquals("failure - strings are not equal", expected, actual);
}
```

- **Floating point assertions**

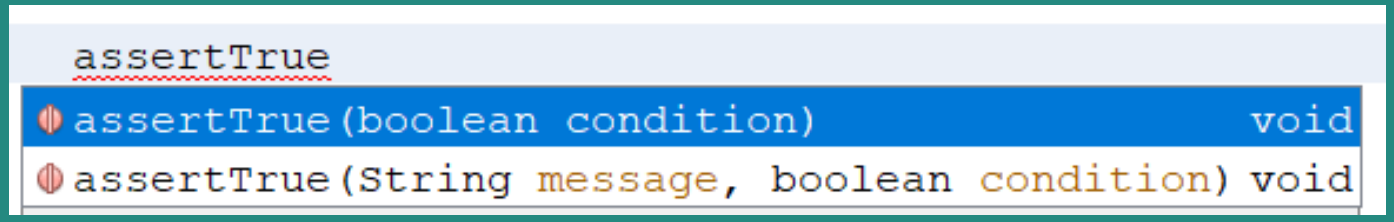
  assertEquals(aDoubleValue, anotherDoubleValue, 0.001)

assertEquals



assertTrue and assertFalse

```java
@Test
public void assertTrueTest1() {
    assertTrue("5 is greater then 4", 5 > 4); // test passed
}

@Test
public void assertTrueTest2() {
    assertTrue("5 is equal to 4", 5 == 4); // test failed
}

@Test
public void assertFalseTest() {
    assertFalse("5 is not greater then 6", 5 > 6); // test passed
}
```



assertNull and assertNotNull

```java
@Test
public void assertNullTest() {
    Object obj = null;

    assertNull("The object should be null", obj);
}

@Test
public void assertNotNullTest() {
    Object obj = new Object();

    assertNotNull("The object should be not null", obj);
}
```

assertArrayEquals

```
@Test
public void assertArrayEqualsTest() {
    char[] expected = {'J','u','n','i','t','4'};
    char[] actual = "Junit4".toCharArray();

    assertArrayEquals(expected, actual);
}
```

assertSame and assertNotSame

- To compare two references to the same java object

```
@Test
public void assertNotSameTest() {
    Object obj1 = new Object();
    Object obj2 = new Object();

    assertNotSame(obj1, obj2);
}

@Test
public void assertSameTest() {
    Object obj1 = new Object();
    Object obj2 = obj1;

    assertSame(obj1, obj2);
}
```

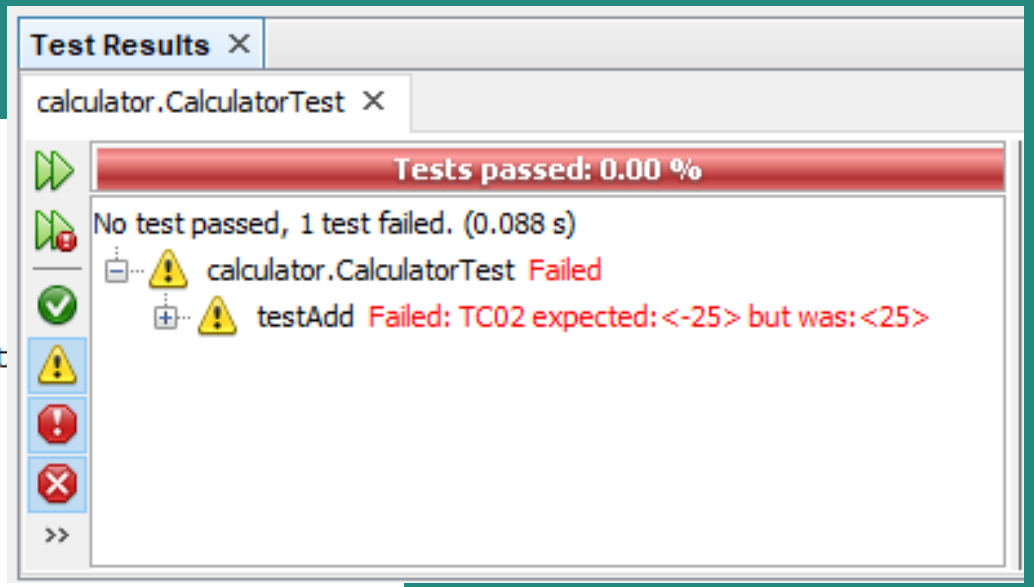
fail

- to make a test failing

```
@Test
public void failingTest() {
    fail("a failing test");
}

@Test
public void abortedTest() {
    assertTrue("abc".contains("Z"));
    fail("test should have been aborted");
}
```

JUnit Parameterized Test (1)

```
16  public class CalculatorTest {
17      private Calculator cal;
18
19      @Before
20      public void setUp() throws Except
21          cal = new Calculator();
22      }
23
24      @Test
25      public void testAdd() {
26          assertEquals("TC01", 30, cal.add(10, 20));
27          assertEquals("TC02", -25, cal.add(0, 25));
28          assertEquals("TC03", -100, cal.add(-40, -60));
29          assertEquals("TC04", 10, cal.add(-40, 50));
30      }
31  }
```

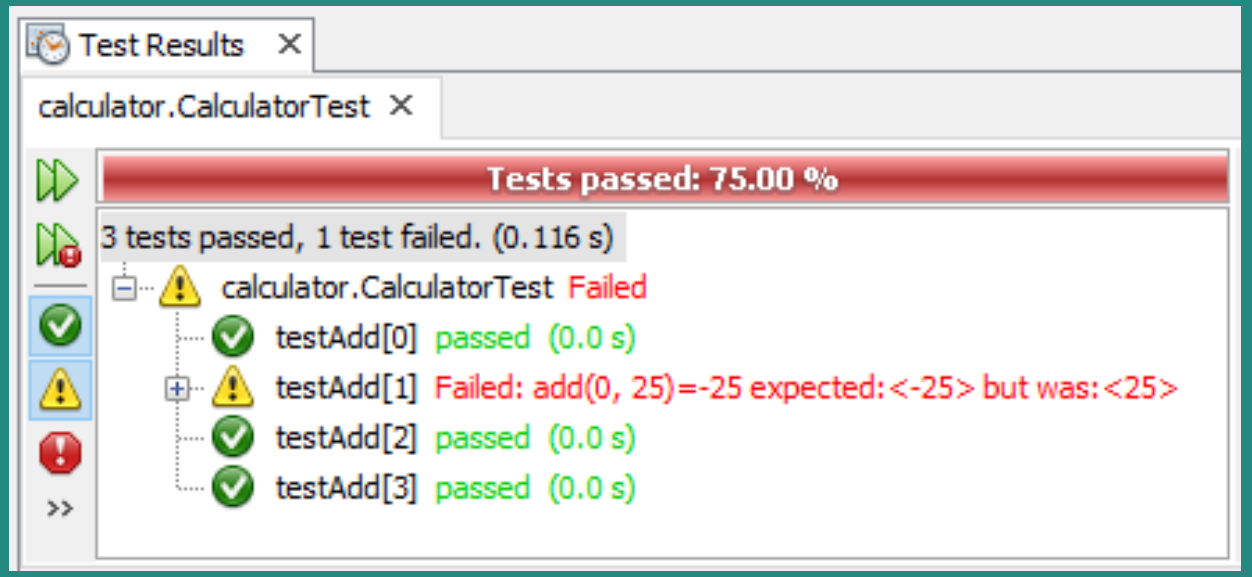


JUnit Parameterized Test (2)

```
@RunWith(Parameterized.class)
public class CalculatorTest {
    private Calculator cal;

    @Parameter(0)
    public int a; // NOT private

    @Parameter(1)
    public int b; // NOT private

    @Parameter(2)
    public int expResult; // NOT private

    @Parameters
    public static Collection setParameters(){
        return Arrays.asList(new Object[][]{
            {10, 20, 30}, {0, 25, -25}, {-40, -60, -100}, {-40, 50, 10}});
    }
```

JUnit Parameterized Test (3)

```
    @Test
    public void testAdd() {
        String msg = "add("+a+", "+b+")="+expResult;
        assertEquals(msg, expResult, cal.add(a, b));
//          assertEquals("TC01", 30, cal.add(10, 20));
//          assertEquals("TC02", -25, cal.add(0, 25));
//          assertEquals("TC03", -100, cal.add(-40, -60));
//          assertEquals("TC04", 10, cal.add(-40, 50));
    }
```



JUnit Parameterized Test (4)

```
@RunWith(Parameterized.class)
public class CalculatorTest {
    private Calculator cal;
    private int a;
    private int b;
    private int expResult;

    public CalculatorTest(int a, int b, int expResult) {
        this.a = a;
        this.b = b;
        this.expResult = expResult;
    }
    @Parameterized.Parameters
    public static Collection setParameters(){
        return Arrays.asList(new Object[][]{
            {10, 20, 30}, {0, 25, -25}, {-40, -60, -100}, {-40, 50, 10}});
    }
```

Constructor

Bài tập thực hành 01

```
public static String getGreeting(int hour) {
    if (hour < 0 || hour > 23) {
        return "Invalid hour";
    } else if (hour < 12) {
        return "Good morning";
    } else if (hour < 18) {
        return "Good afternoon";
    } else {
        return "Good evening";
    }
}
```

Bài tập thực hành 02

- **Viết hàm tính tổng tiền mua hàng biết số lượng sản phẩm và đơn giá, chương trình khuyến mãi được áp dụng như sau**
  - **Mua 4 chỉ tính tiền 3**
  - **Số lượng còn lại không tròn 4 sản phẩm được tính**
    - Nếu còn lẻ 1 sp: giảm giá 10% cho 1 sản phẩm này
    - Nếu còn lẻ 2 sp: giảm giá 15% cho 2 sản phẩm này
    - Nếu còn lẻ 3 sp: giảm giá 20% cho 3 sản phẩm này
- **Hàm có 2 tham số int và return double (tổng tiền phải trả)**

Bài tập thực hành 02

```
public static double getTotal(int quantity, double price) {
    int fullSets = quantity / 4;
    int remainingItems = quantity % 4;
    double total = 0.0;

    total += fullSets * 3 * price;
    if (remainingItems == 1) {
        total += remainingItems * 0.9 * price;
    } else if (remainingItems == 2) {
        total += remainingItems * 0.85 * price;
    } else if (remainingItems == 3) {
        total += remainingItems * 0.8 * price;
    }
    return total;
}
```

Bài tập thực hành 03

- Viết hàm tính tiền nước biết lượng tiêu thụ và đơn giá được tính *tích lũy* như sau:
  - $20m^3$ đầu tiên: theo đơn giá ban đầu
  - $10m^3$ tiếp theo: đơn giá tăng 50% giá ban đầu
  - $5m^3$ tiếp theo: đơn giá tăng 60% giá ban đầu
  - Số còn lại: đơn giá tăng 70% giá ban đầu
- Hàm có 2 tham số int và return double (tổng tiền phải trả)

Bài tập thực hành 03

```java
public static double getWaterBill(int quantity, double initPrice) {
    double total;

    if (quantity <= 20) {
        total = quantity * initPrice;
    } else if (quantity <= 30) {
        total = 20 * initPrice + (quantity - 20) * 1.5 * initPrice;
    } else if (quantity <= 35) {
        total = 20 * initPrice + 10 * 1.5 * initPrice + (quantity - 30) * 1.6 * initPrice;
    } else {
        total = 20 * initPrice + 10 * 1.5 * initPrice + 5 * 1.6 * initPrice + (quantity - 35) * 1.7 *
    }
    return total;
}
```

Bài tập thực hành 04

- Hàm xếp loại tam giác có 3 cạnh a<=b<=c
  1. a, b, c không tạo thành tam giác
  2. a=b=c: tam giác đều
  3. a=b và $c^2=a^2+b^2$: vuông cân tại C
  4. $c^2=a^2+b^2$: vuông tại C
  5. cân (cân tại A hoặc cân tại B hoặc cân tại C)
  6. tam giác thường
- Hàm có 3 tham số a, b, c kiểu nguyên
- Hàm trả về -1, 0, 1, 2, 3, 4 tương ứng với các trường hợp 1..6

Bài tập thực hành 04

```
public static int getTriangleType(int a, int b, int c) {
    if (a + b <= c) {
        return -1; // Không phải tam giác
    } else if (a == b && b == c) {
        return 0; // Tam giác đều
    } else if ((c * c == a * a + b * b) && (a == b)) {
        return 2; // Tam giác vuông cân
    } else if (a == b) {
        return 3; // Tam giác cân
    } else if (c * c == a * a + b * b) {
        return 1; // Tam giác vuông
    } else {
        return 4; // Tam giác thường
    }
}
```

Mẫu White Box Testcases

No	Description	Test data	Expected Output	Note
Statements				
1	Bao phủ tất cả các dòng code ngoại trừ dòng 5 và 12	a=?, b=?	TRUE	
2	Bao phủ dòng 5	a=?, b=?	FALSE	
3	Bao phủ dòng 12	a=?, b=?	TRUE	
Paths				
4	a%2==0 && b>0: true	a=?, b=?	FALSE	Trường hợp trùng với TCs của statements thì chọn test data khác nhau
5	a%2==0 && b>0: false	a=?, b=?	TRUE	
Conditions				
6	a%2==0 && b>0: F, -	a=?, b=?	TRUE	- Chỉ viết TCs trong trường hợp biểu thức điều kiện có từ 2 toán hạng trở lên - Lập bảng quyết định
7	a%2==0 && b>0: T, F	a=?, b=?	FALSE	
8	a%2==0 && b>0: T, T	a=?, b=?	FALSE	