See the 2 APIs attached at the end of this worksheet.

1. **Methods: Javadoc** Complete the Javadoc comments for the following two methods from the API:

(a)
```
   /**
    *
    *
    *
    *
    * @param
    *
    * @param
    *
    * @param
    *
    * @return
    *
    * @pre.
    */
   public static boolean isOutside(int min, int max, int value)
```

Solution:

```
   /**
    * Returns true if value is strictly less than min
    * or strictly greater than max, and false otherwise.
    *
    * @param min
    *            a minimum value
    * @param max
    *            a maximum value
    * @param value
    *            a value to check
    * @return true if value is strictly less than min
    *         or strictly greater than max, and false otherwise
    * @pre. min is greater than or equal to max
    */
   public static boolean isOutside(int min, int max, int value)
```

(b)
```
  /**
    * Given a list containing exactly 2 integers, negates the values
    * of the integers in the list. The list t is not modified by this
    * method. For example, given a list
    *
    * <p>
    * <code>[-5, 9]</code>
    *
    * <p>
    * <code>negate2</code> modifies the list so that it becomes
    *
    * <p>
    * <code>[5, -9]</code>
    *
```

```
     *
     *
     *
     *
     *
     *
     *
     *
     *
     *
     *
     *
     *
     *
     */
    public static void negate2(List<Integer> t)
```

Solution:

```
    /**
     * Given a list containing exactly 2 integers, negates the
     * values of the integers in the list. The list t is not
     * modified by this method. For example, given a list
     *
     * <p>
     * <code>[-5, 9]</code>
     *
     * <p>
     * <code>negate2</code> modifies the list so that it becomes
     *
     * <p>
     * <code>[5, -9]</code>
     *
     * @pre t is not null
     *
     * @param t
     *            a list containing exactly 2 integers
     * @throws IllegalArgumentException
     *             if the list does not contain exactly 2 integers
     */
    public static void negate2(List<Integer> t)
```

2. **Utility classes**

   Create a utility class with the following features:

   1. it is located in the package named `eecs2030.test1`
   2. its name is `Pizza`
   3. it has a `public` constant named `SLICES_LARGE` whose value is 10
   4. it has a method named `numberOfSlices` that has one parameter of type `int` named `n` and returns an `int` value
   5. the method named `numberOfSlices` returns the total number of slices in the given number of large pizzas.

   Think about what preconditions the method might have.

Solution:

```
package eecs2030.test1;

public class Pizza {

    /**
     * The number of slices in a large pizza.
     */
    public static final int SLICES_LARGE = 10;

    private Pizza() {
        // TO PREVENT INSTANTIATION
    }

    /**
     * Returns the total number of slices in the given number
     * of large pizzas.
     *
     * @param n the number of pizzas
     *
     * @return the total number of slices in the given number
     * of large pizzas
     *
     * @pre. n >= 0
     */
    public static int numberOfSlices(int n) {
        return n * Pizza.SLICES_LARGE;
    }
}
```

3. **Utility classes**

   Find all of the errors in the following utility class; note that the class does compile:

```
import java.util.Date;

public class DateUtil {

    /**
     * Today's date.
     */
    public static final Date TODAY = new Date();

    /**
     * Returns the year of today's date.
     *
     * @return the year of today's date
     */
    public int getYear() {
        // Date is weird; it represents the year as the number of
        // years after 1900
        return DateUtil.TODAY.getYear() + 1900;
    }

}
```

Solution:

1. `TODAY` looks like a constant but `Date` is mutable (so anyone can change the state of `TODAY` so that it is a different date)
2. a `private` constructor is missing
3. `getYear` should be a `static` method

4. **Write a test using a main method**

   Write a main method that tests the method `Test2G.interval(int n)`.

Solution: There are many possible answers; the solution below tests a typical case.

```
    public static void main(String[] args) {
        int n = 3;
        List<Integer> got = Test2G.interval(n);
        System.out.println(
            String.format("interval(%d) : %s", n, got));
    }
```

5. **JUnit 1**

   Write a method that uses JUnit to test the method `Test2G.isOutside(int min, int max, int value)`.

   Solution: There are many possible solutions; the solution below tests a boundary case.

```
    @Test
    public void test_isOutside() {
        int min = -5;
        int max = 3;
        int value = 3;
        boolean exp = false;
        boolean got = Test2G.isOutside(min, max, value);
        assertEquals(exp, got);

        // could also use:
        // assertFalse(got);
    }
```

6. **JUnit 2**

   Convert your answer for Question 4 into a method that uses JUnit to test the method `Test2G.interval(int n)`.

   Solution:

```
    @Test
    public void test_isOutside() {
        int n = 3;
        List<Integer> exp = new ArrayList<>();
        exp.add(0);
        exp.add(1);
        exp.add(2);
        exp.add(3);

        List<Integer> got = Test2G.interval(n);
        assertEquals(exp, got);
    }
```

7. **Test cases**

   Recall that a test case is a specific set of arguments to pass to a method, and the expected return value (if any) and the expected results when the method is called with the specified arguments.

   Provide two test cases for the method `negate2(List<Integer> t)`. One test case should test that an exception is thrown, and the other test case should test that the values in the list are negated.

   Solution: Your test cases should show the arguments to the method being tested and the corresponding expected result of running the method using those inputs. 5 possible test cases are shown below:

arguments t	expected result t	explanation
$[\,]$	IllegalArgumentException	list of size 0 should cause an exception
$[100]$	IllegalArgumentException	list of size 1 should cause an exception
$[100, 200, 300]$	IllegalArgumentException	list of size 3 should cause an exception
$[5, 7]$	$[-5, -7]$	positive values in t are negated
$[-5, -7]$	$[5, 7]$	negative values in t are negated

8. **Test cases 2**

Consider the method `isInside(double x, double y)`:

```
/**
 * Determine if the point (x, y) is strictly inside the circle with center
 * (0, 0) and having radius equal to 1. A point on the perimeter of
 * the circle is considered outside of the circle.
 *
 * @param x the x-coordinate of the point
 * @param y the y-coordinate of the point
 * @return true if (x, y) is inside the unit circle, and false otherwise
 */
public static boolean isInside(double x, double y)
```

Provide 5 test cases for the method `isInside(double x, double y)`. Make sure to provide test cases for typical argument values that produce return values of true and false, and test cases that test boundary cases.

Solution: Again, you should list the arguments to the method and the expected result of running the method with the arguments.

arguments x	 y	expected result return value
0.5	0.5	true
0.9999	0	true
0	1.0001	false
-1.2	0.3	false
1.0	0.0	false (boundary case where (x, y) is exactly on the circle)
0.0	-1.0	false (boundary case where (x, y) is exactly on the circle)

eecs2030.test2

Class Test2G

java.lang.Object
    eecs2030.test2.Test2G

```
public class Test2G
extends Object
```

Field Summary

Fields

Modifier and Type	Field and Description
`static int`	`NEGATE2_LIST_SIZE` The size of the list required by the method Test2G.negate2

Method Summary

All Methods **Static Methods** **Concrete Methods**

Modifier and Type	Method and Description
`static List<Integer>`	`encode(List<Integer> t)` Returns a run length encoded list representing the numbers in the the list `t`.
`static List<Integer>`	`interval(int n)` Returns a new list containing the values 0, 1, 2, ..., n.
`static boolean`	`isOutside(int min, int max, int value)` Returns true if `value` is strictly less than `min` or strictly greater than `max`, and false otherwise.
`static void`	`negate2(List<Integer> t)` Given a list containing exactly 2 integers, negates the values of the integers in the list.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

NEGATE2_LIST_SIZE

```
public static final int NEGATE2_LIST_SIZE
```

The size of the list required by the method Test2G.negate2

See Also:

`Constant Field Values`

Method Detail

isOutside

```
public static boolean isOutside(int min,
                                int max,
                                int value)
```

Returns true if `value` is strictly less than `min` or strictly greater than `max`, and false otherwise.

Parameters:

`min` - a minimum value

`max` - a maximum value

`value` - a value to check

Returns:

true if value is strictly less than min or strictly greater than max, and false otherwise

Precondition:

min is greater than or equal to max

negate2

```
public static void negate2(List<Integer> t)
```

Given a list containing exactly 2 integers, negates the values of the integers in the list. The list `t` is not modified by this method. For example, given a list

```
[-5, 9]
```

`negate2` modifies the list so that it becomes

```
[5, -9]
```

Parameters:

`t` - a list containing exactly 2 integers

Throws:

`IllegalArgumentException` - if the list does not contain exactly 2 integers

Precondition:

t is not null

interval

```
public static List<Integer> interval(int n)
```

Returns a new list containing the values 0, 1, 2, ..., n. Returns an empty list if n is less than zero.

Parameters:

`n` - the upper limit of the interval

Returns:

a new list containing the values 0, 1, 2, ..., n or the empty list if n is less than zero

encode

```
public static List<Integer> encode(List<Integer> t)
```

Returns a run length encoded list representing the numbers in the the list `t`. The list `t` is not modified by this method.

Consider the list containing twelve ones:

```
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

A run length encoded version of the list would be:

```
 [12, 1]         (twelve 1s)
```

Similarly, consider the list containing four tens and three fives:

```
 [10, 10, 10, 10, 5, 5, 5]
```

A run length encoded version of the list would be:

```
 [4, 10, 3, 5]         (four 10s, three 5s)
```

If the list contains few or no repeated values, then the run length encoded version of the list is longer than the original list. For example, the run length encoded version of the list:

```
 [1, 2, 3, 4]
```

is:

```
 [1, 1, 1, 2, 1, 3, 1, 4]    (one 1, one 2, one 3, one 4)
```

Parameters:

`t - a list of integer values`

Returns:

`a new list equal to the run length encoded version of t`

Precondition:

`t is not null, t is not empty`