See the API attached at the end of this worksheet.

1. **Organization of a Java program**

   (a) What is the package name of the provided API?

   > **Solution:** `eecs2030.test2` (found at the beginning of the API before the class name)

   (b) What is the class name of the provided API?

   > **Solution:** The class name is `Test2E`. The fully qualified class name would also include the package name and is `eecs2030.test2.Test2E`.

   (c) How many methods appear in the API?

   > **Solution:** 13 (4 are listed in the Method Summary section, and 9 are listed in the Methods inherited from class java.lang.Object section).

   (d) Can you guess what classes might need to be imported when implementing the class described by the API?

   > **Solution:** Three of the methods have a `List` as a parameter or return a `List`; therefore, the class `java.util.List` needs to be imported.

2. **Methods: Basics**

   (a) All of the methods in the API have the same modifiers. What are the modifiers for these methods?

   > **Solution:** `public static`; you have to consult the Method Detail section to find the modifier `public` in the method header.

   (b) State the signature for each method in the API.

   signature of `avg`

   signature of `swap2`

   signature of `allGreaterThan`

   signature of `toInt`

   > **Solution:**
   > ```
   > avg(int, int, int)
   > swap2(List<Integer>)
   > allGreaterThan(List<Integer>, int)
   > toInt(List<Integer>)
   > ```

(c) State the return value type for each method in the API.

return type of `avg`

return type of `swap2`

return type of `allGreaterThan`

return type of `toInt`

> **Solution:** `double`
> `void`
> `List<Integer>`
> `int`

(d) All of the following groups of Java statements that are written by a client of the Test2E class contain an error; circle the error and explain what the error is.

i.
```
double avg = Test2E.avg(1.0, 2.0, 3.0);
```

> **Solution:** `avg` has 3 `int` parameters but the client has used 3 `double` arguments.

ii.
```
List<Integer> t = new ArrayList<Integer>();
t.add(5);
t.add(6);
List<Integer> u = Test2E.swap2(t);
```

> **Solution:** `swap2` does not return value.

iii.
```
List<Integer> t = new ArrayList<Integer>();
t.add(5);
t.add(6);
List<Integer> u = Test2E.allGreaterThan(t);
```

> **Solution:** `allGreaterThan` has a `List<Integer>` parameter followed by an `int` parameter but the client has used only a `List<Integer>` argument.

iv.
```
ArrayList<Integer> t = new ArrayList<Integer>();
t.add(-1);
t.add(0);
double value = toInt(t);
```

> **Solution:** When using a `static` method the client (usually) needs to use the class name before the method name; i.e., the client should have written:
>
> ```
> double value = Test2E.toInt(t);
> ```

Note that even though `Test2E.toInt` returns an `int` it is not an error to assign the returned value to a `double` variable.

If you know what a `static import` statement does then you could argue that there is no error in the client's code; feel free to google the phrase "java static import" to find out more.

3. **Methods: Preconditions and postconditions**

   (a) Inspect the API for the method named avg. What are its preconditions? What are its postconditions?

   Solution: There are no preconditions. The postcondition is that the average of `a`, `b`, and `c` is returned.

   (b) Inspect the API for the method named swap. What are its preconditions? What are its postconditions?

   Solution: The preconditions are that `t` is not `null` and contains exactly two integers. The postcondition is that the order of the two elements in the list are swapped.

   (c) Inspect the API for the method named allGreaterThan. Is "the elements of the list t must be integers" a precondition? Explain why or why not.

   Solution: This is not a precondition because the client's code will not compile if the client does not provide a list of integers.*

   The answer above is not completely accurate. It is possible for the client to write code that calls `allGreaterThan` using a list that does not contain integers:

```
List t = new ArrayList();   // old style List usage; do not do this
t.add("hello");             // t is a list of strings
Test2E.allGreaterThan(t);   // compiles
```

   This code compiles, but the method fails when the program is run. This happens because generic types were introduced in version 5 of the Java language; prior to version 5, clients had to remember what types were held by a list, or had to write code to determine what types were held by a list. If you write code like that shown above, a modern compiler will warn that the type held by the list is missing and is unsafe.

4. **Methods: Implementation**

   (a) Implement the method named `avg`.

   (b) Implement the method named `swap`.

   (c) Implement the method named `allGreaterThan`.

   Solution: See Test2E.java on the course moodle in the Lectures section for Section E.

5. **Methods: Pass-by-value** Consider the following class having a single method:

```
class Swapper {

    // Swaps the values of a and b
    public static void swap(int a, int b) {
        int tmp = a;
        a = b;
        b = tmp;
    }

}
```

Now consider a client program that tries to use `Swapper`:

```
class Swapper {

    public static void main(String[] args) {
        int x = 99;
        int y = 100;
        Swapper.swap(x, y);
        System.out.println("x = " + x + ", y = " + y;
    }

}
```

(a) What does the program print?

Solution: `x = 99, y = 100`

(b) Draw a memory diagram for the client program (ignoring the `println` statement).

Solution: Immediately when the `swap` method is called, the values of `x` and `y` are passed by value to the `swap` method:

	100	main method
x		99
y		100
	200	swap method
a		99
b		100

When the `swap` method is finished running, the values of `a` and `b` are swapped, but the values of `x` and `y` remain unchanged:

	100	main method
x		99
y		100
	200	swap method
a		100
b		99
tmp		99

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

(a)
```
/**
 *
 *
 * @param a
 *
 * @param b
 *
 * @param c
 *
 * @return
 */
public static double avg(int a, int b, int c)
```

(b)
```
/**
 * Given a list containing exactly 2 integers, swaps the positions
 * of the integers in the list. For example, given a list
 *
 * <p>
 * <code>[-5, 9]</code>
 *
 * <p>
 * <code>swap2</code> modifies the list so that it becomes
 *
 * <p>
 * <code>[9, -5]</code>
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
```

```
     */
    public static void swap2(List<Integer> t)
```

Solution: See Test2E.java on the course moodle in the Lectures section for Section E.

7. **Utility classes**

   Create a utility class with the following features:

   1. it is located in the package named `eecs2030.test1`
   2. its name is `CircleUtil`
   3. it has a `public` constant named `TWO_PI` whose value is 2π
   4. it has a method named `circumference` that has one parameter of type `double` named `radius` and returns a `double` value
   5. the method named `circumference` returns the circumference of the circle having the given radius

   Think about what preconditions the method might have.

Solution:

```
package eecs2030.test1;

public class CircleUtil {

    public static final double TWO_PI = 2 * Math.PI;

    private CircleUtil() {
        // to prevent object creation
    }

    public static double circumference(double radius) {
        return CircleUtil.TWO_PI * radius;
    }
}
```

A possible precondition for the method is that the radius should be greater than or equal to zero.