CS: 114 **OPPDA** **Spring, 2025**

Project4 (100 points) **Due 3/10/2025**

Submit your solutions to canvas. Do not send the entire project. All that is needed are the files ending in .java. Each class will be defined in its own separate .java file. All driver code should be put in one .java file. Please make sure your name is included at the top of each .java file. Make one zip file that includes all the .java files and submit that one zip file to canvas.

Project Goals

1. Practice with ordering objects via lambda expressions that implement the abstract method `compare`
2. Practice with `forEach` method
3. Practice with `lambda` expressions
4. Practice passing lambda expressions into a function as a parameter.

Problem 1

Examine the lecture7 java file name `FuncInterface_lambda_test`. For P1 of your driver create your own **functional interface** and create two reference variables using lambda expressions and execute those expressions. Implement your own lambda expressions for the `Consumer` interface, and the `BiConsumer` interface.

Create a static method `foo` that accepts one of your functional interface variables as a parameter and executes the function variable in foo.

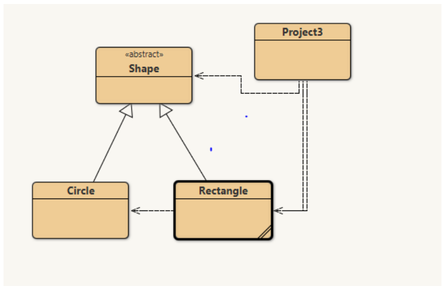
Problem 2

Refer to your solution for Project. You are to modify your code as follows:

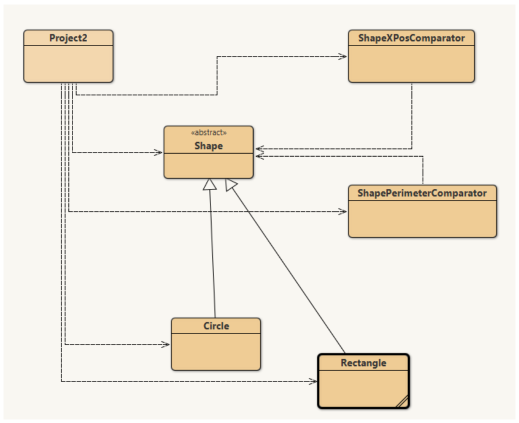
1) You will eliminate the need to implement any Comparators by instead using lambda expressions. This will affect these two calls:

```
Arrays.sort(shapesList, ~~new ShapeXPosComparator()~~);
Arrays.sort(shapesList, ~~new ShapePerimeterComparator()~~);
```

You will eliminate the need to implement any Comparators by instead using lambda expressions. Change all the enhanced for-loops to `forEach` loops. The BlueJ class diagram should look like:



as opposed to this:



Problem 3

Examine the lecture7 java file name `FuncInterface_lambda_test`. In particular look at `ex3` which calls the static method `PerformConditionally`. Supply three different lambda expressions for the predicate interface. One should filter out numbers that are not prime. The next filter should filter out numbers that are odd. The last filter should filter out numbers that are not factors of 3. The `consumer.accept` should just simply print out the numbers to the console window that successfully pass through the filters.

So, for the first time you call `PerformConditionally` it should only display the numbers that are prime numbers. The second time you call it should display the only the numbers that are even number, and the last to you call it it should display only numbers that are factors of the number 3.

You should test your code with different numbers for the `nums` array.

```
List<Integer> nums = List.of(2, 3, 1, 5, 6, 7, 8, 9, 12, 256, 300, 301,  303, 17, 120);
```

You should not modify the method `PerformConditionally`. For reference here is the method:

```
private static void PerformConditionally(List<Integer> nums, Predicate<Integer> filter,
Consumer<Integer> consumer)
    {
        for (Integer i : nums)
        {
            if (filter.test(i)) {
                consumer.accept(i);
            }
        }
    }
```