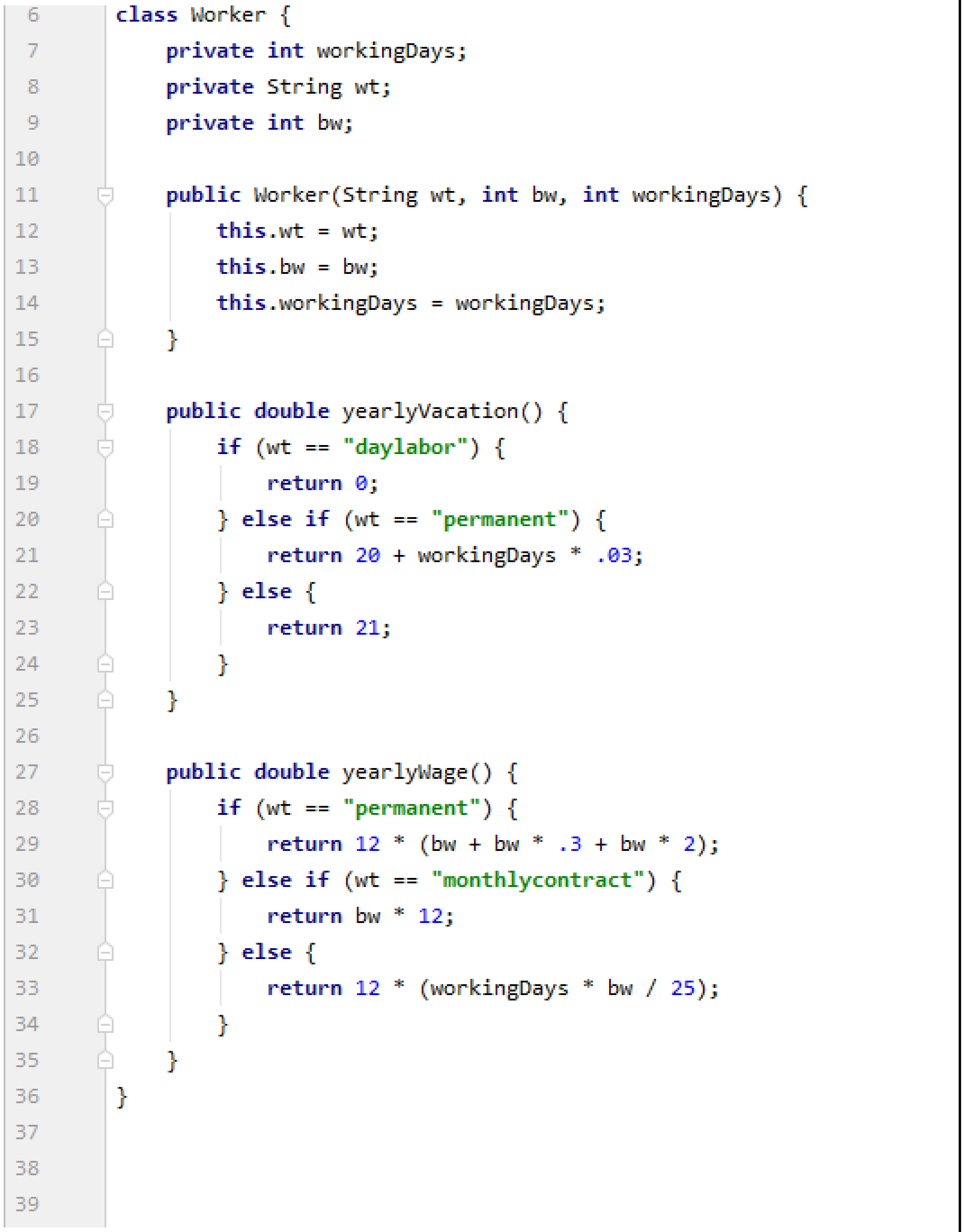
Lab 6: Generics

1.	**GenericMaxStack** **Time**: 30 minutes **Problem Description** You need to create a class named GenericMaxStack that represents a last-in-first-out (LIFO) data structure with the following properties: 1. It has push(int) and pop() operations that work the same way as a normal stack 2. In addition, it has a max() operation that returns the maximum value in the current stack. 3. You have to ensure that your code is working for Integer, Double, and String data types. **Constraints** The max() operation should operate at constant complexity, O(1). This means you cannot use a loop or recursion to find the minimum value. **Test cases** 1. Push 3, 5, 2. Assert max = 5. 2. Push 2, 1, 2, 5. Pop the last element. Assert max = 2. Pop again. Assert max = 2. 3. Push 49.75, 23.54, 100.0. Assert max 100. Pop the last element. Assert max 49.75. 4. Push “OOC is bad”, “Nothing to understand”, and “Try hard”. Assert max “Try hard”. Pop the last element. Assert max “OOC is bad”. **Hint** 1. You can use the built-in Stack class if necessary. 2. You can keep up to the max in the stack each time you insert an element in the stack.	5
2.	**GenericCount** **Time**: 40 minutes **Problem Description** Write a generic method to count the number of elements in a list that have a specific property like divisible by 3. Keep in mind that this property could be changed into odd numbers, or each element length is greater than 5 for a string	5

	type element. So, you should write your code in a way that is open to future changes. 1. Write a class Algorithm that has a generic method **countIf**. This method receives a list of integers or strings and another parameter to know whether you count divisible by 3, an odd number, or the length is greater than 4. **Test cases** 1. Call the countIf method with a list of numbers 2, 3, 5, and 6. Assert 2 numbers are divisible by 3. 2. Call the countIf method with a list of numbers 2, 3, 16, 6. Assert 1 odd number. 3. Call the countIf method with a list of strings “Alice”, “Bob”, and “Beautiful”. Assert 2. **Hint** 1. You have to think about the interface to solve it.	
3	**Refactoring** Time: 40 minutes Refactor the following code: **Test Cases** 1. Write 3 test cases for each of the employee types to check their yearly salary and yearly leaves. 2. Write 1 test case to check the type of an object using assertInstanceOf method.	5



```java
class Worker {
    private int workingDays;
    private String wt;
    private int bw;

    public Worker(String wt, int bw, int workingDays) {
        this.wt = wt;
        this.bw = bw;
        this.workingDays = workingDays;
    }

    public double yearlyVacation() {
        if (wt == "daylabor") {
            return 0;
        } else if (wt == "permanent") {
            return 20 + workingDays * .03;
        } else {
            return 21;
        }
    }

    public double yearlyWage() {
        if (wt == "permanent") {
            return 12 * (bw + bw * .3 + bw * 2);
        } else if (wt == "monthlycontract") {
            return bw * 12;
        } else {
            return 12 * (workingDays * bw / 25);
        }
    }
}
```