Advanced Object-Oriented Programming

CPT204 – Lab 1

Erick Purwanto

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CPT204 Advanced Object-Oriented Programming

Lab 1

Basic Java Review, Checking 1, Testing 1

Welcome !

- Welcome to Lab 1 !
  - Please read Lecture 1 first, if you have not done so

- In this lab, we are going to guide you how to
  1. Import files into a project
  2. Submit lab exercises to Learning Mall Quiz
  3. Submit programming exercises to Learning Mall Quiz

You will need to do this step for every lab exercises and programming exercises !

- We are also going give hints to Lab Exercise 1.1

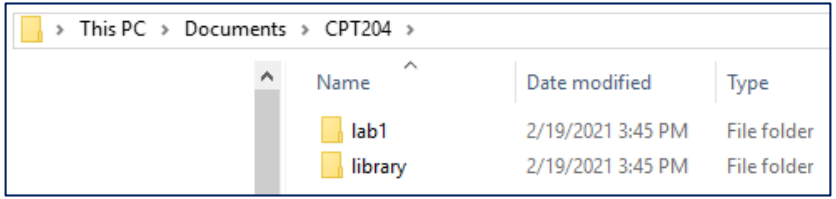
Import Lab 1 into a Project (1)

We are going to guide you to import the files of Lab 1 into an IntelliJ project (*repeat* these steps for the subsequent labs !)

1. Download from Learning Mall the zip files library and lab1 ;

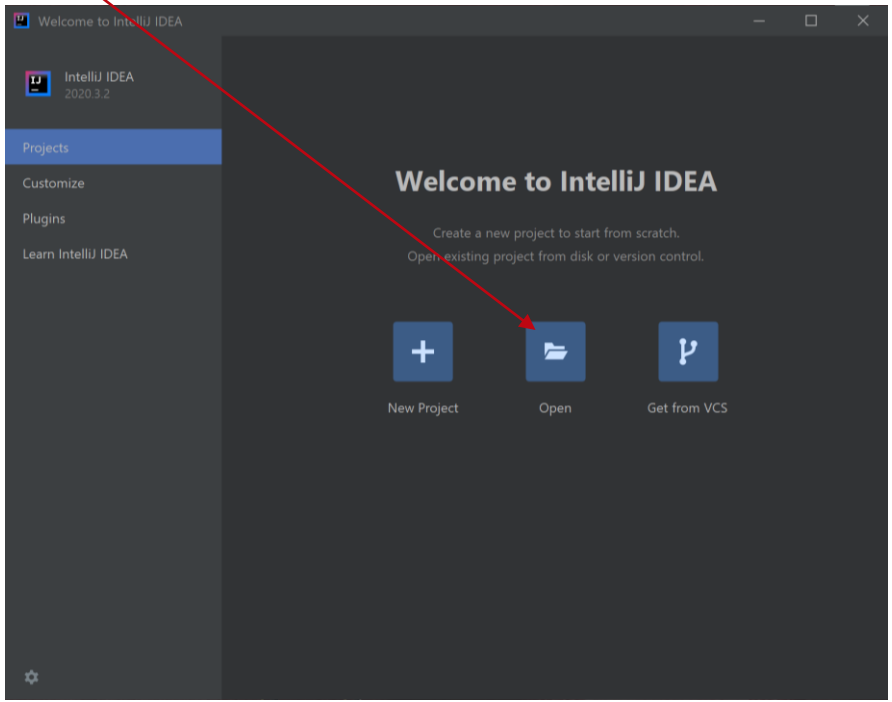
Create a folder for the CPT204 lab files, e.g. CPT204, in your computer;

Unzip to get these two folders:



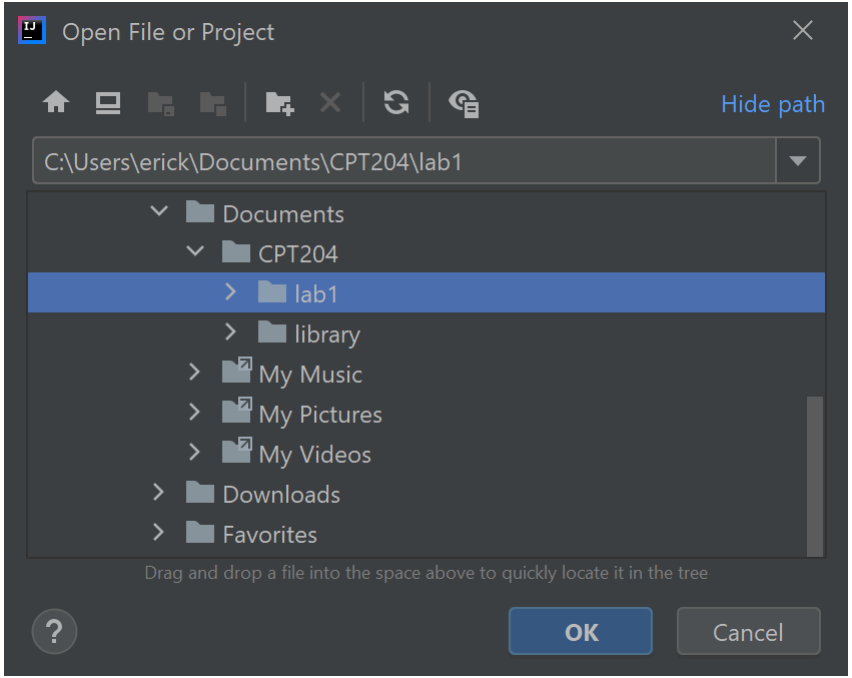
Import Lab 1 into a Project (2)

2. Open IntelliJ; click Open



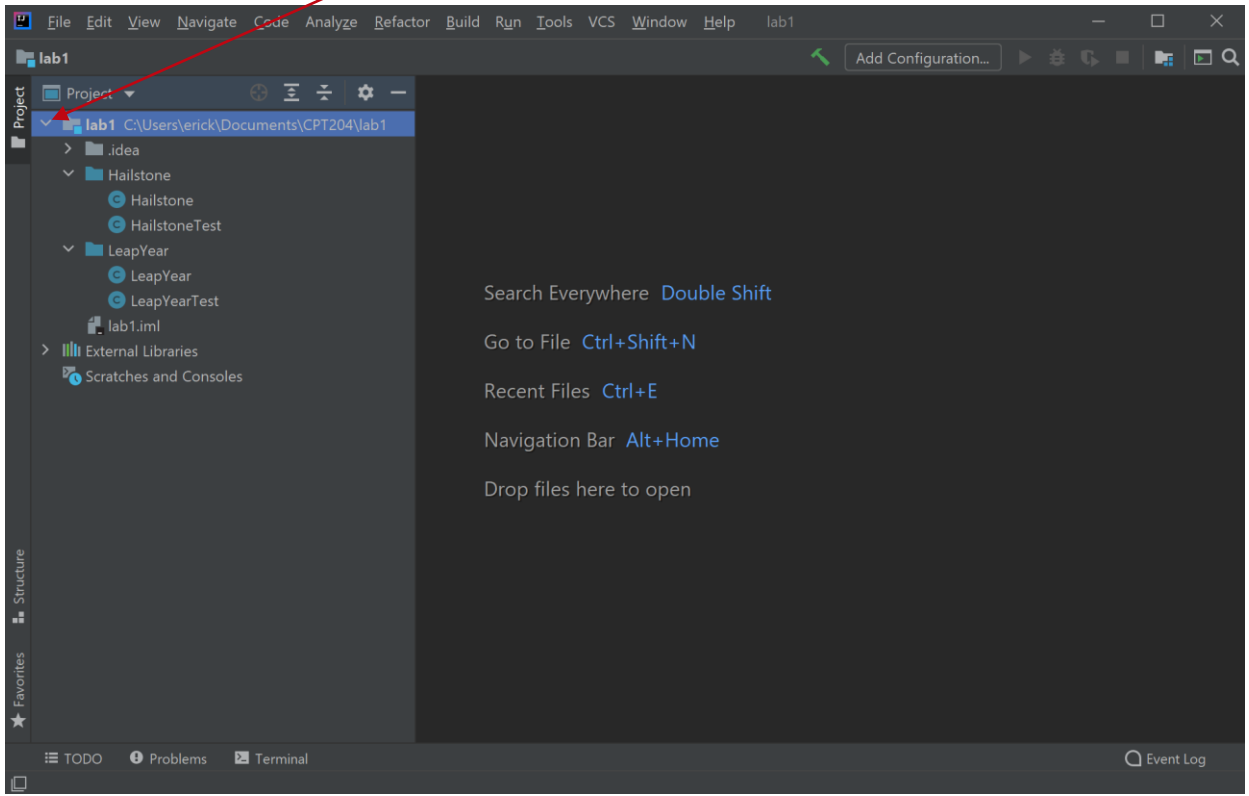
Import Lab 1 into a Project (3)

3. Find your lab1 folder, and click OK



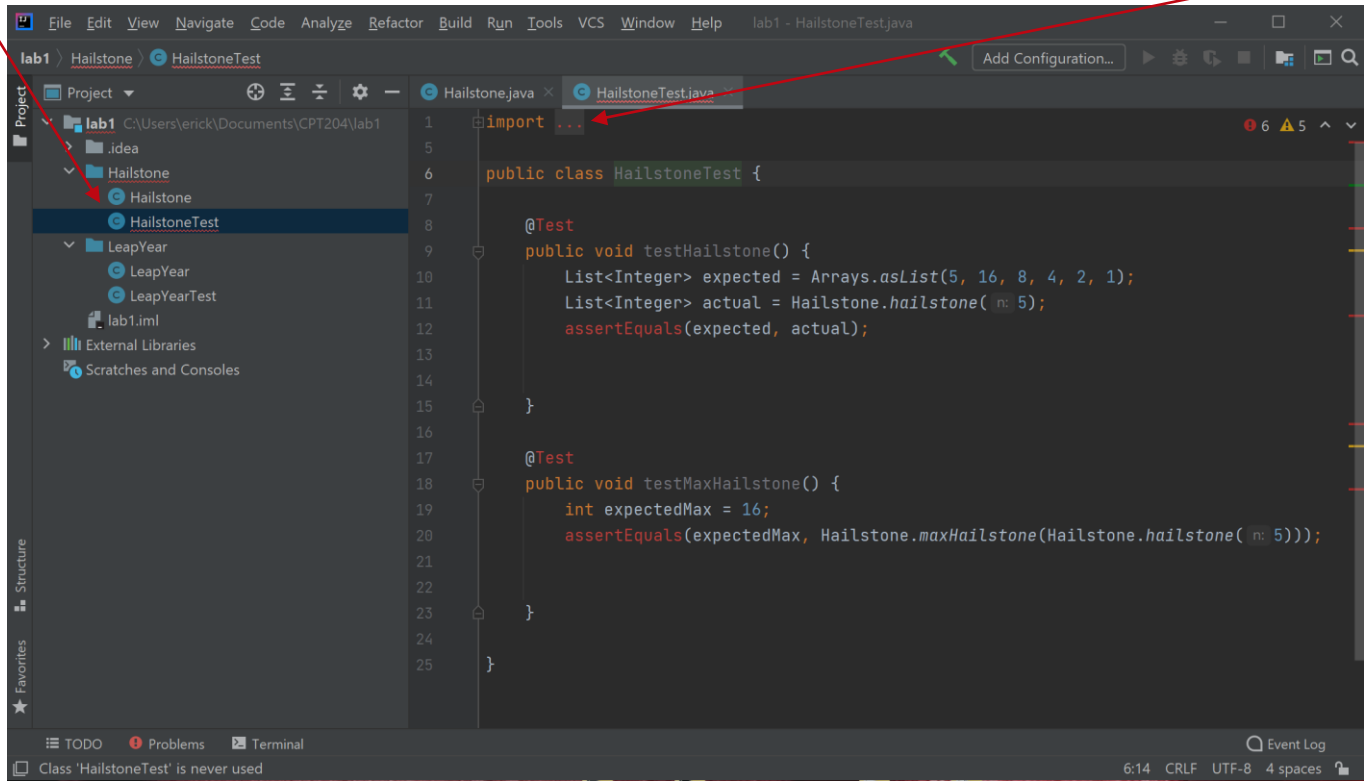
Import Lab 1 into a Project (4)

4. Your project is opened; Click the little triangle to show the source files



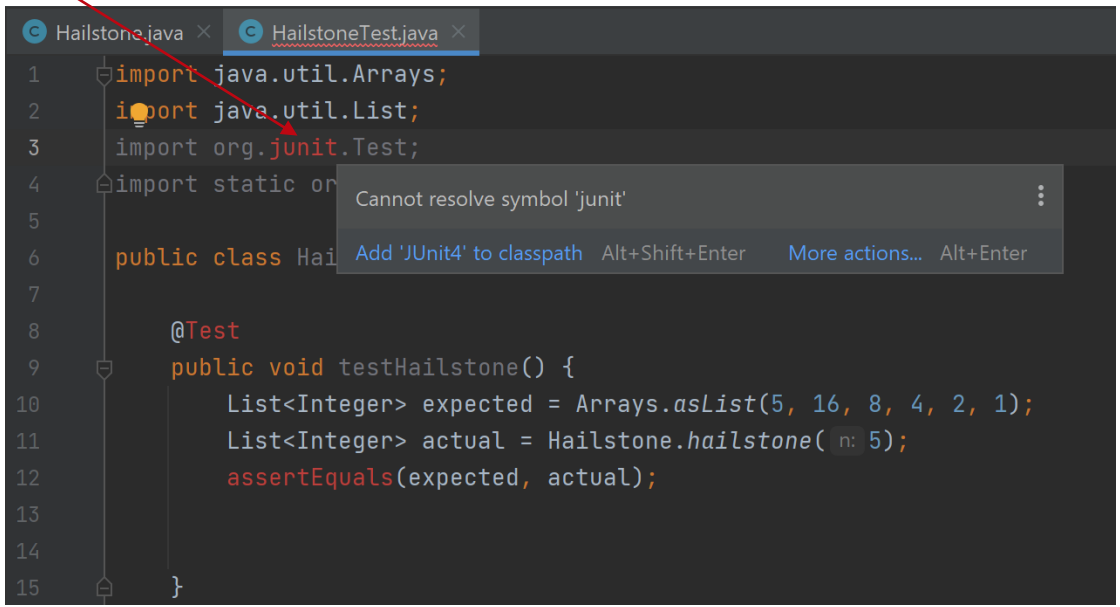
Import Lab 1 into a Project (5)

5. Click the source files, it shows errors in one of the source code; Click …



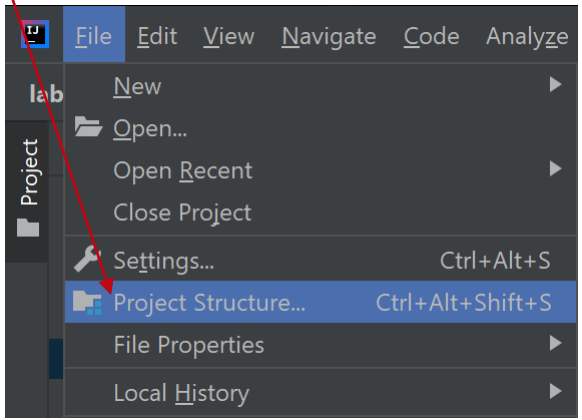
Import Lab 1 into a Project (6)

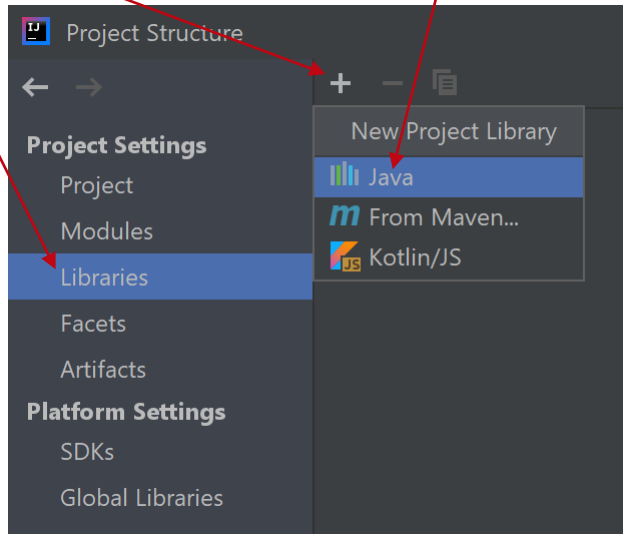
6. Hover over junit; see that Java cannot resolve it; this is because we have not added the necessary libraries, which we will be doing next



Import Lab 1 into a Project (7)

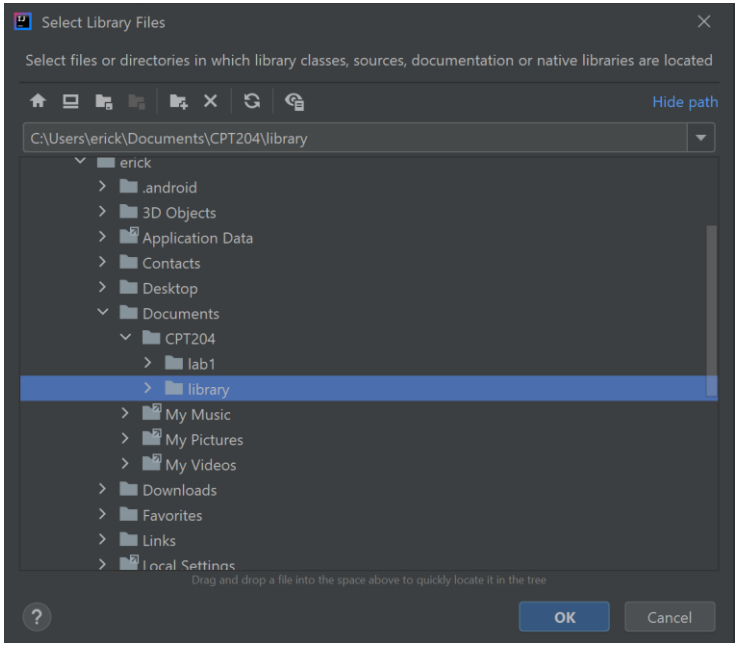
7. Click File -> Project Structure; Click Libraries -> Plus sign; Click Java

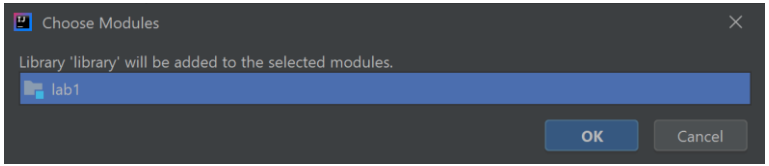




Import Lab 1 into a Project (8)

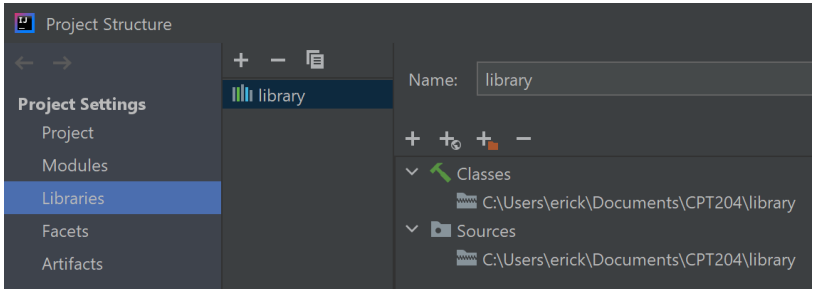
8. Find your library folder, and click OK; Click OK again



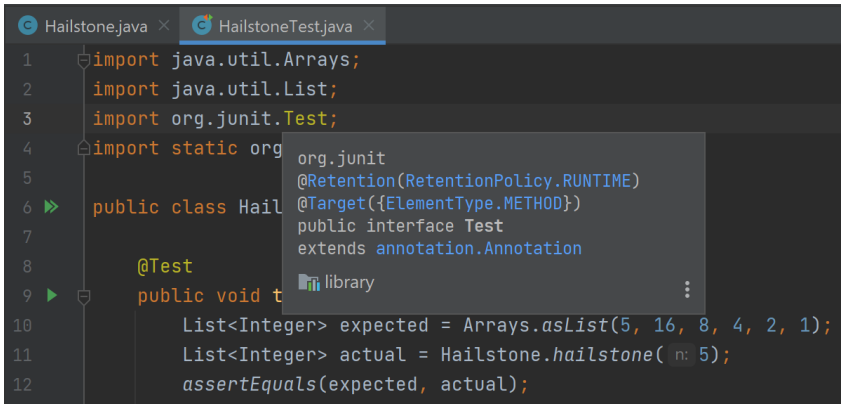


Import Lab 1 into a Project (9)

9. Finally, you should see the library in the Libraries of Project Structure ;



the errors disappear ; and we are now ready to use it in the next section!



The First Lab Exercise

- We now continue with the Lab Exercise
- We will first use IntelliJ to write and test our codes
  - and then you need to submit your code to Learning Mall

- In the next slide, you will see Lab Exercise 1.1
- Lab Exercise slides will consist of
  - Description of the problem
  - One or more test cases
  - Hints and skeleton code (optional)

- After that, we will guide you to solve and submit the Lab Exercise 1.1

Lab Exercise 1.1 Max Hailstone

- Complete the code to compute the maximum element of a hailstone sequence
- The input is an integer n > 0 ,
  and the output is the maximum element in that sequence
- Test case:
  - `maxHailstone(5) → 16`

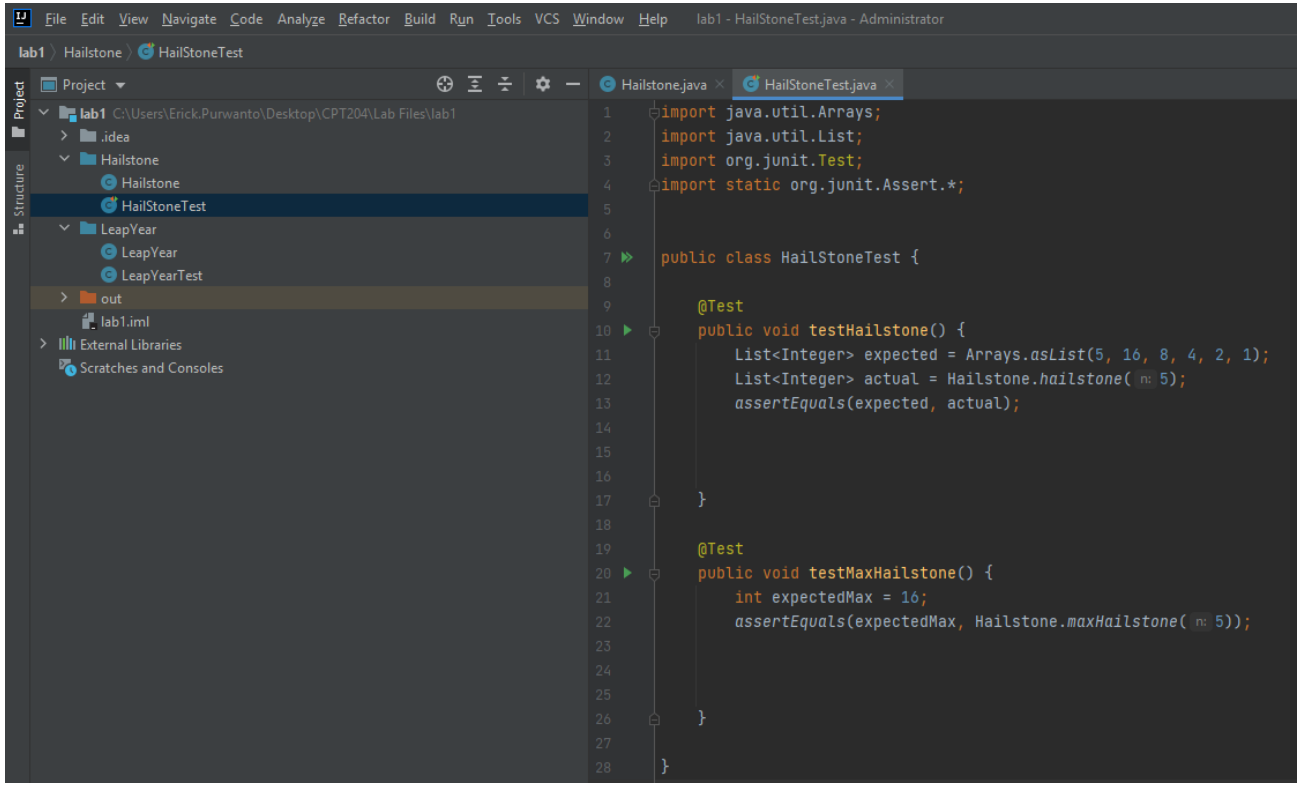
Lab Exercise 1.1 Max Hailstone

- Skeleton code:

```
/**
  * Compute the largest element in a hailstone sequence.
  * For example, maxHailstone(5) = 16.
  * @param n starting number of the sequence. Assume n > 0.
  * @return the largest element the sequence.
  */
public static int maxHailstone(int n) {



}
```

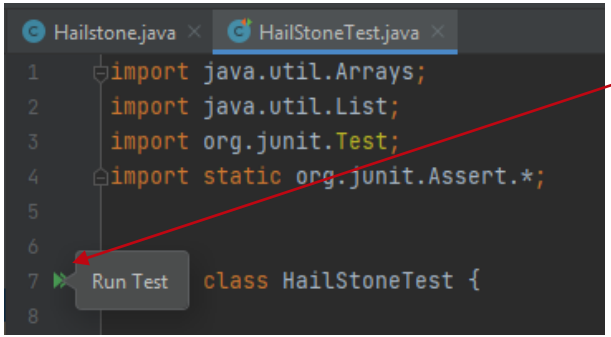
Our First Lab Exercise (1)

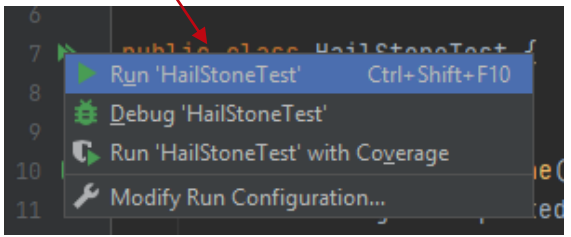
1. Finish the previous step Import Lab1 into a Project; Open HailstoneTest.java



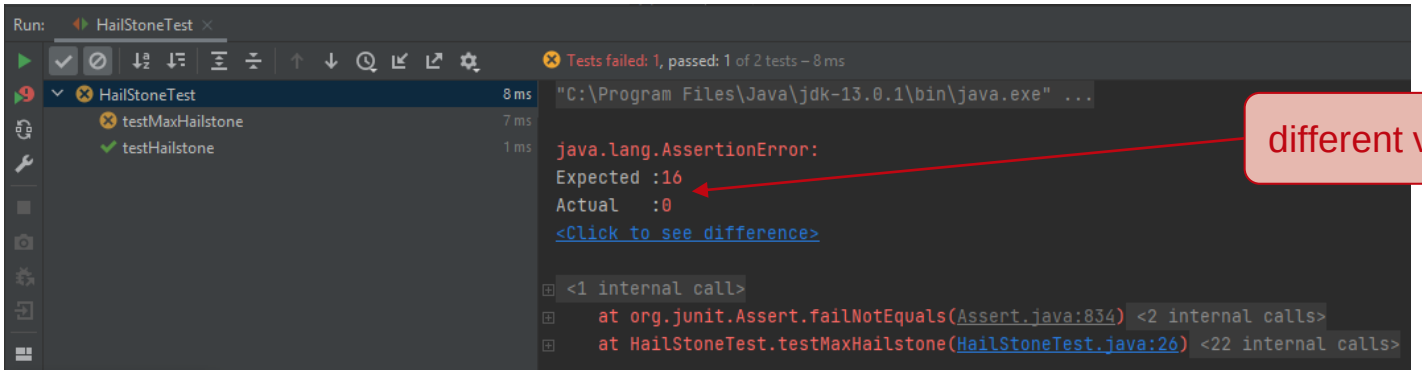
Our First Lab Exercise (2)

2. Let's first run all the already written tests by clicking Green Arrow and Run





testMaxHailstone failed because we have *not* complete the code



Our First Lab Exercise (3)

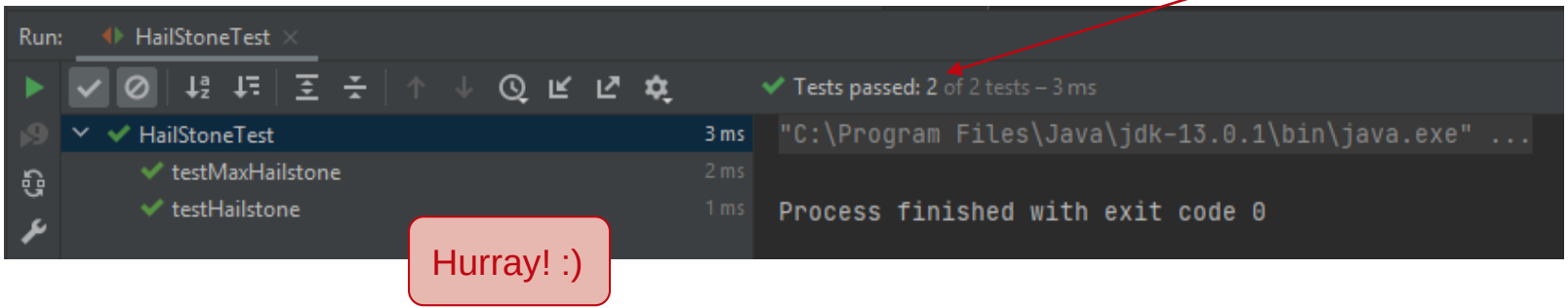
3. Add ***more*** test cases, as discussed in Lecture 1
   For example, hailstone(3) = [3 10 5 16 8 4 2 1] and its max element = 16

```java
@Test
public void testHailstone() {
    List<Integer> expected = Arrays.asList(5, 16, 8, 4, 2, 1);
    List<Integer> actual = Hailstone.hailstone( n: 5);
    assertEquals(expected, actual);

    expected = Arrays.asList(3, 10, 5, 16, 8, 4, 2, 1);
    actual = Hailstone.hailstone( n: 3);
    assertEquals(expected, actual);

}

@Test
public void testMaxHailstone() {
    int expectedMax = 16;
    assertEquals(expectedMax, Hailstone.maxHailstone( n: 5));

    expectedMax = 16;
    assertEquals(expectedMax, Hailstone.maxHailstone( n: 3));

}
```

add your own test cases !

Our First Lab Exercise (4)

4. Open Hailstone.java
   The method hailstone has been completed for you
   Now complete the method maxHailstone,
   and then check againsts your own set of test cases until all tests are passed



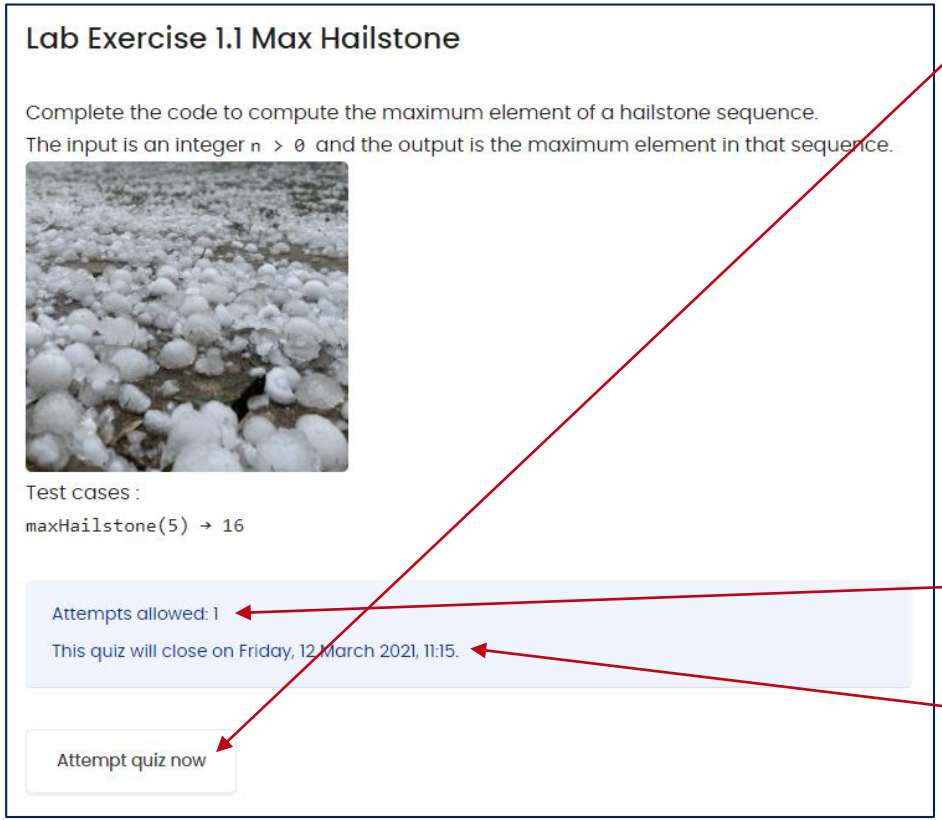
Our First Lab Exercise (5)

5. Continue to test and debug your code using IntelliJ & JUnit until all of your own test cases are passed
   (create more test cases if necessary)

6. After you are satisfied with your code, submit into Learning Mall, click the quiz Lab Exercise 1.1

Our First Lab Exercise (6)

7. You will see the following screen, click Attempt quiz now

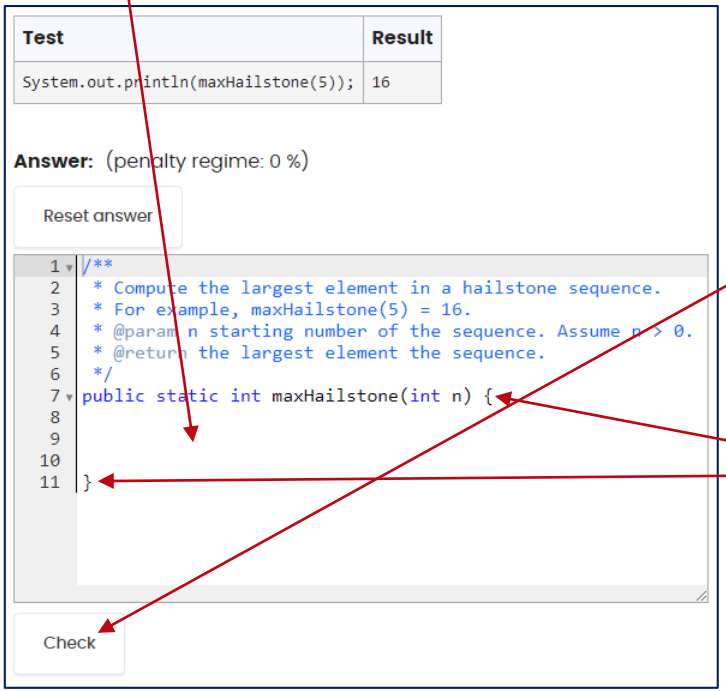


note that only one attempt is allowed for a lab exercise

and it has a deadline !

Our First Lab Exercise (7)

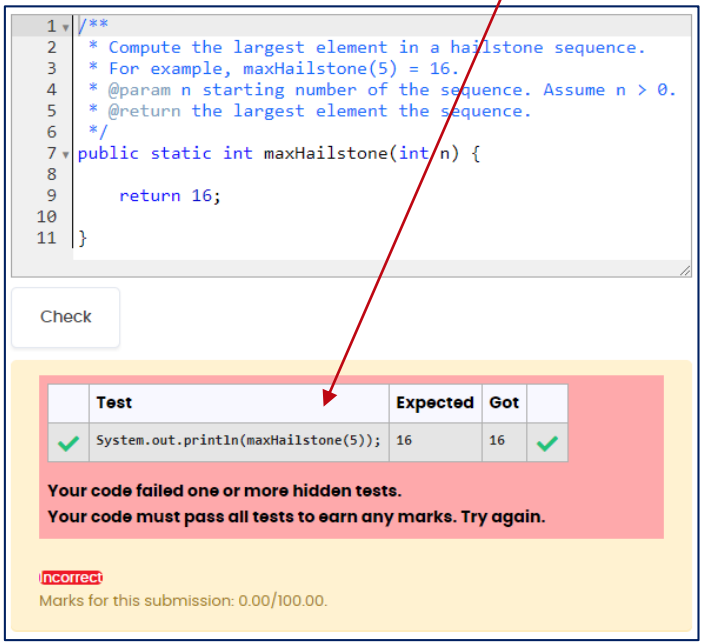
8. Copy paste your code from IntelliJ into the box, and then click Check



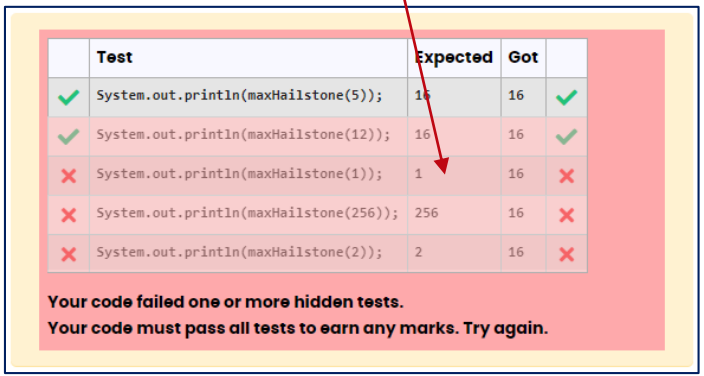
pay attention to the curly braces { }
missing or extra curly braces are common errors!

Our First Lab Exercise (8)

9. You have to pass **all** test cases: given and hidden ones, to earn marks

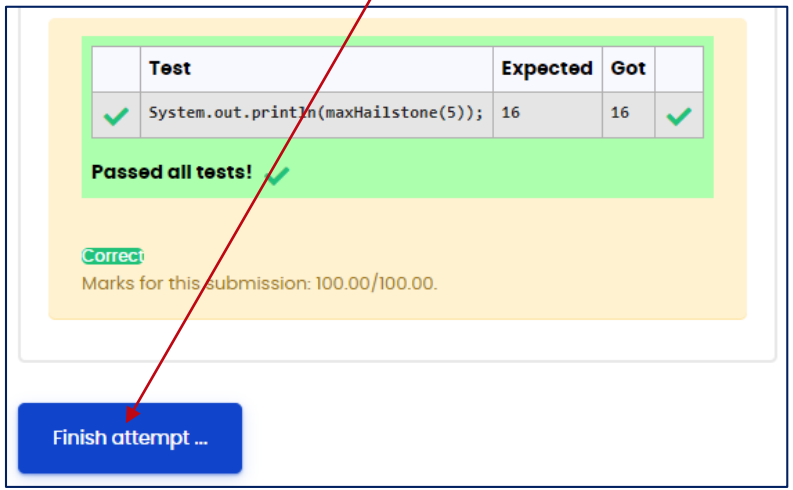


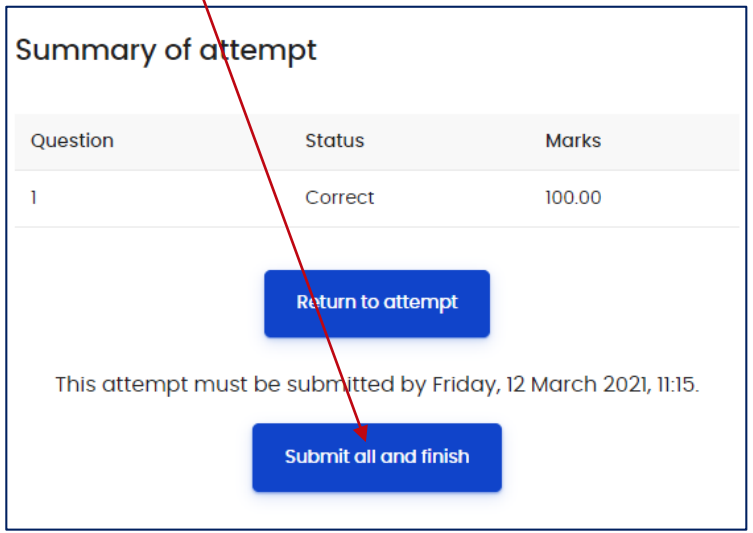
you cannot see this,
but the code gave incorrect
output to some hidden test cases



Our First Lab Exercise (9)

10. After passing, click Finish attempt , and then click Submit all and finish





WARNING: Hint to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 1.1 Max Hailstone Hints

- You can simply reuse the code to generate the list of hailstone numbers, and find its maximum number
- Or, more efficiently, you can just store the current max while iterating
  - make sure you initialize the variable storing max correctly

The First Programming Exercise

- We now continue with the Programming Exercise
- Just like the Lab Exercise, use IntelliJ to write and test your code first
- After it is well-tested, submit your code to Learning Mall

- In the next slide, you will see the Exercise 1.1
- Programming Exercise slides will consist of
  - Description of the problem
  - One or more test cases
  - Hints and skeleton code (optional)

- After that, we will guide you to submit the Exercise 1.1
  - You will have to solve it yourself

Exercise 1.1 Leap Year

- Leap years are years where an extra day is added to the end of the shortest month, February.
- Write a Java method with signature `boolean isLeapYear(int year)` that returns `true` if and only if `year` is a leap year.

- Test cases:
  - `isLeapYear(2020)` → `true`
  - `isLeapYear(2019)` → `false`

Exercise 1.1 Leap Year

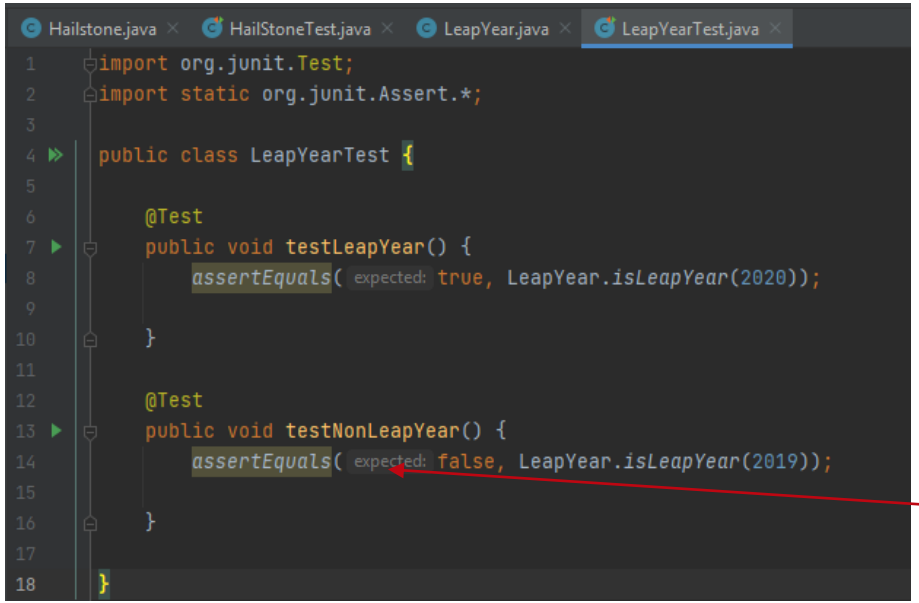
- Skeleton code:

```
/**
 * Checks if an input year is a leap year.
 * @param year is the input year
 * Requires year to be a valid year
 * @return true iff year is a leap year
 */
public static boolean isLeapYear(int year) {



    return true;
}
```

Our First Programming Exercise (1)

1. It is recommended that you finish Lab Exercise 1.1 first

   Open LeapYearTest. java, write the JUnit import statements, and write the test cases, for example:

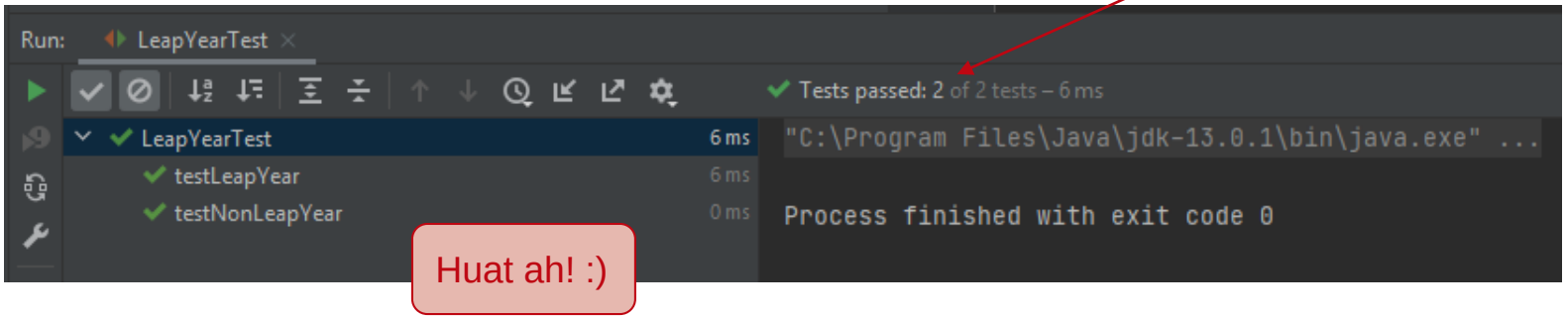


text automatically added by IntelliJ, to remind you what argument it is, do not type in "expected:"

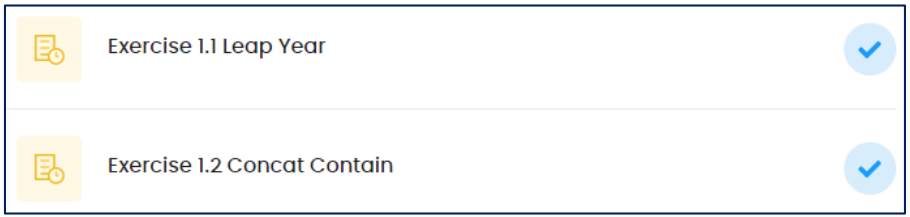
Test cases above are **not enough**, find other cases of leap/nonleap year !!

Our First Programming Exercise (2)

2. Open LeapYear.java, write your code
   Continue to write, test, debug your code in IntelliJ until it passes your good set of test cases

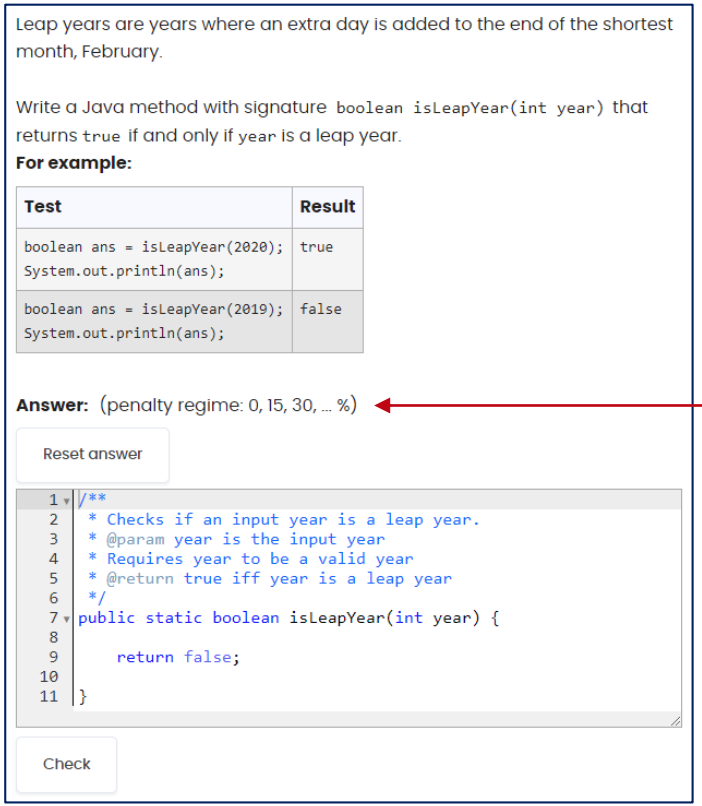


3. After you are satisfied with your code, submit it to Learning Mall the same way as submitting Lab Exercise 1.1 before



Our First Programming Exercise (3)

4. Note that programming exercises carry penalty for repeated mistakes



Leap years are years where an extra day is added to the end of the shortest month, February.

Write a Java method with signature `boolean isLeapYear(int year)` that returns `true` if and only if `year` is a leap year.

For example:

Test	Result
`boolean ans = isLeapYear(2020);` `System.out.println(ans);`	`true`
`boolean ans = isLeapYear(2019);` `System.out.println(ans);`	`false`

Answer: (penalty regime: 0, 15, 30, ... %)

Reset answer

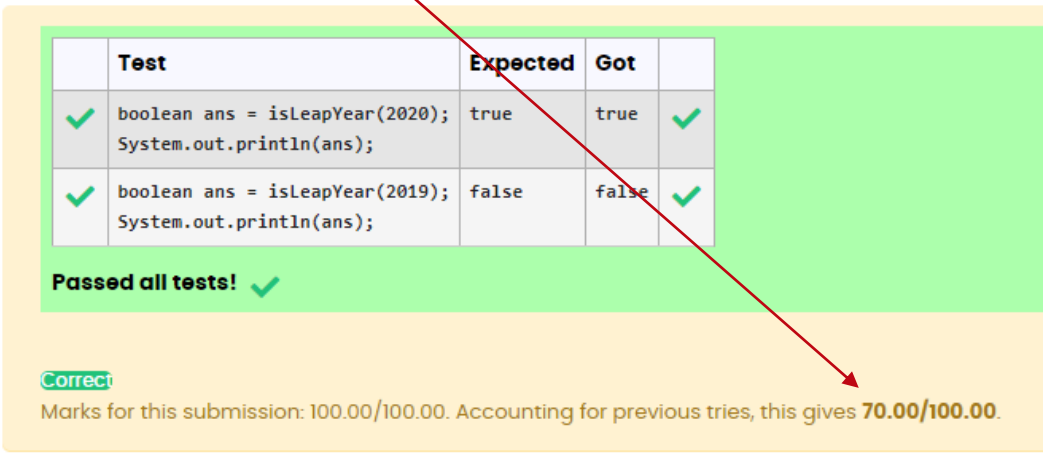
```
1  /**
2   * Checks if an input year is a leap year.
3   * @param year is the input year
4   * Requires year to be a valid year
5   * @return true iff year is a leap year
6   */
7  public static boolean isLeapYear(int year) {
8
9      return false;
10
11 }
```

Check

note that the ***first*** *Check* has no penalty if your code is incorrect, but the ***second*** *Check* and subsequent checks will be penalized by accumulative 15% thus, do ***not*** test your code here, but **test in IntelliJ+JUnit** !

Our First Programming Exercise (4)

For example, if your 1st, 2nd and 3rd checks are incorrect,
but your 4th check is correct,
you will be penalized by 15%+15%



	Test	Expected	Got	
✓	boolean ans = isLeapYear(2020); System.out.println(ans);	true	true	✓
✓	boolean ans = isLeapYear(2019); System.out.println(ans);	false	false	✓

Passed all tests! ✓

Correct

Marks for this submission: 100.00/100.00. Accounting for previous tries, this gives **70.00/100.00**.

the same setting is used for the **CW / Lab Exam**!

don't test your code here,
test your code in IntelliJ !

Exercise 1.2 Concat Contain

- Complete the method int concatContain(String source, String target).
- Given two *non-empty* strings source and target, it could be possible to **concatenate** the string source **with itself a number of times**, so that the string target can be **contained** in it.
- For example, source "ab" concatenated 2 times "ab"+"ab"+"ab" into "ababab" contains target "baba".
- Return the smallest number of times you concatenate source so that it contains target; and if it is not possible for target to be contained in concatenated source strings, return -1.
- You must **not** use StringBuilder or Regular Expression methods: append, matches, replaceAll.
- Test cases:
  - `concatContain("ab", "baba")  → 2`
  - `concatContain("ab", "abcde") → -1`

Exercise 1.2 Concat Contain

- Skeleton code:

```
/**
 * Compute the smallest number of times source is concatenated with itself
 * so that the resulting string contains target.
 * For example, For example, source "ab" concatenated 2 times "ab"+"ab"+"ab" into "ababab"
 * contains target "baba".
 * @param source a non-empty string to be concatenated.
 * @param target a non-empty string that can be contained in repeatedly concatenated source.
 * @return the smallest number of times of the concatenation.
 */
public static int concatContain(String source, String target) {



    return 0;
}
```

Thank you for your attention !

- In this lab, you have learned:
  - Creating test cases
  - Test-driven Programming
  - Reviewing basic Java and Lists
  - Using IntelliJ and JUnit
  - Submitting your code to Learning Mall