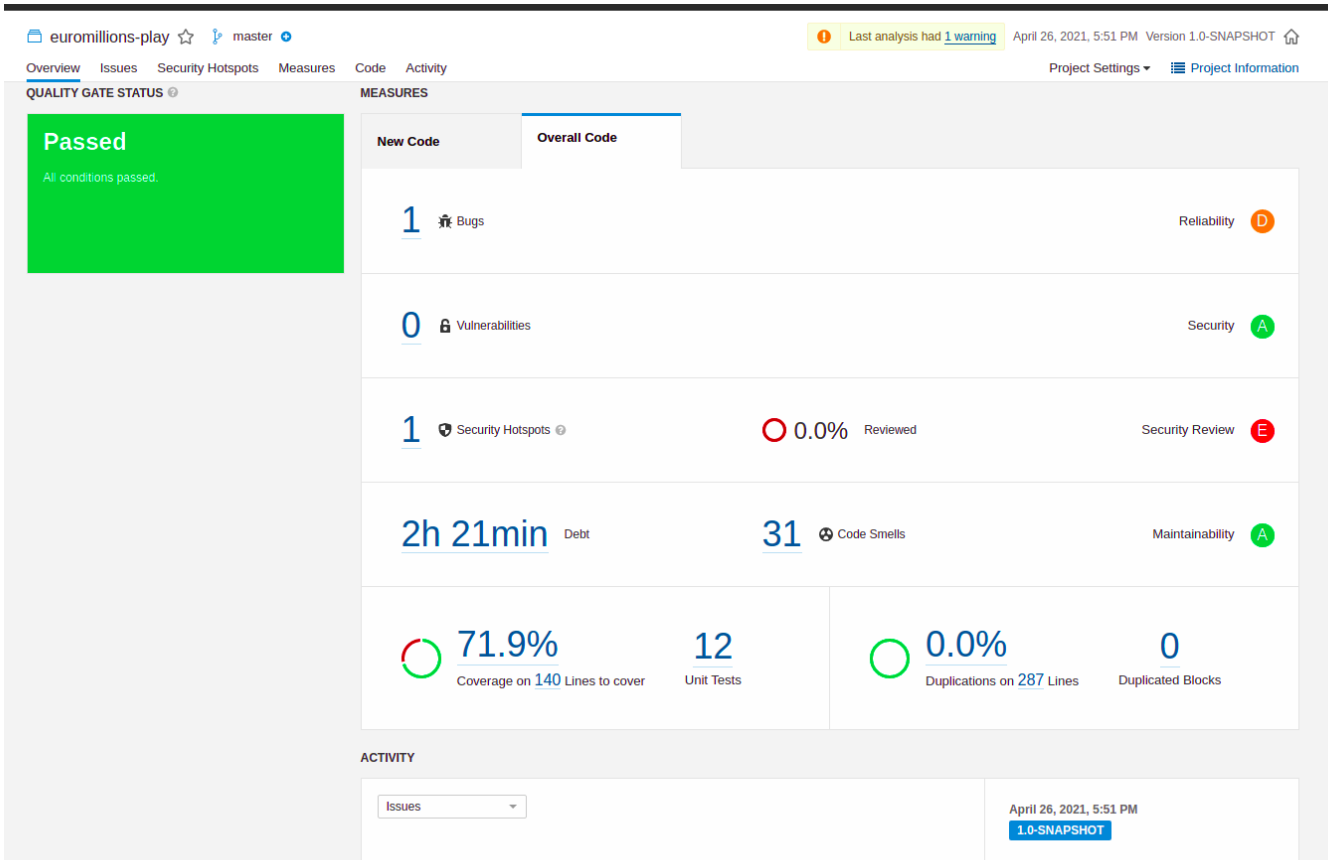
Lab6 - TQS 2021 Report

Ex.1 :

Q. Has your project passed the defined quality gate? Elaborate your answer.

A. Yes, my project passed the quality gate, however, was found 1 bug, 1 security issue and 31 code smells (from which 9 have a major severity rate).



euromillions-play ☆ master

Last analysis had 1 warning April 26, 2021, 5:51 PM Version 1.0-SNAPSHOT

Overview Issues Security Hotspots Measures Code Activity Project Settings ▾ Project Information

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code | Overall Code

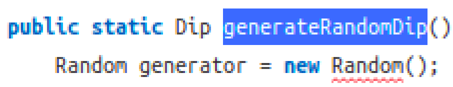
1 Bugs			Reliability D
0 Vulnerabilities			Security A
1 Security Hotspots	0.0% Reviewed		Security Review E
2h 21min Debt	31 Code Smells		Maintainability A
71.9% Coverage on 140 Lines to cover	12 Unit Tests	0.0% Duplications on 287 Lines	0 Duplicated Blocks

ACTIVITY

Issues ▾

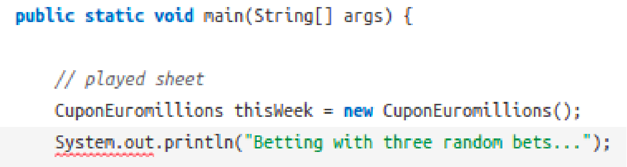
April 26, 2021, 5:51 PM
1.0-SNAPSHOT

Picture 1



```java
public static Dip generateRandomDip() {
    Random generator = new Random();
```

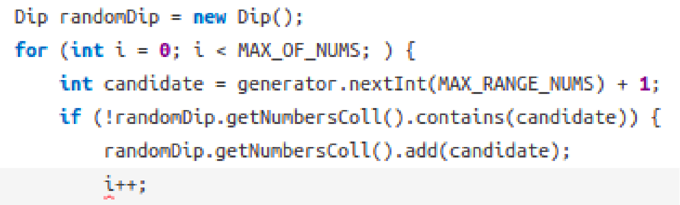
Picture 2



```java
public static void main(String[] args) {

    // played sheet
    CuponEuromillions thisWeek = new CuponEuromillions();
    System.out.println("Betting with three random bets...");
```

Picture 3



```java
Dip randomDip = new Dip();
for (int i = 0; i < MAX_OF_NUMS; ) {
    int candidate = generator.nextInt(MAX_RANGE_NUMS) + 1;
    if (!randomDip.getNumbersColl().contains(candidate)) {
        randomDip.getNumbersColl().add(candidate);
        i++;
```

Issue	Problem Description	How to solve
Bug	In picture 1 we see that every time **generateRandomDip()** is called a new random object is created. This is a problem because the object could've been the same in callback, the only thing that is needed to have a different random int is the **nextInt()** function to be called multiple times.	To solve this problem, we can simply create a private global variable of Random type to use inside **generateRandomDip()** function.
Vulnerability	-----------	-----------
Code Smell (major)	In picture 2 we see that we used **System.out** and instead of logger.log, which is not a good practice because logs assure: easiness when user wants to retrieve them; record of the data logged; security when sensitive data is logged; uniform level and message format that when well applied is easier to read.	Replace all the System.out occurrences for: **LOGGER.log(Level.INFO, "message");**
	In picture 3, the for loop construction is not the most efficient, neither the easiest to maintain and understand. This happens because a for loop stop condition should test the loop counter against an invariant value (one that is true at both the beginning and ending of every loop iteration).	To solve this, we only need to write the cycle like this: for (int i = 0; i < 10; **i++**) And remove i++ from inside the cycle.

Note: The other 7 major code smells that were detected are the same described in table but replicated in other parts of the code.

Ex2:

Q.1 Take note of the technical debt found. Explain what this value means.

A. Debt: 2h21mins
Debt Meaning: Time estimated by SonarQube to solve all the maintainability issues (code smells) found.

Q.2 How many lines are "uncovered"? And how many conditions?

A. Before the correction of severe code smells there were 42 lines uncovered and 14 uncovered conditions. After that correction there are 8 lines uncovered and 2 uncovered conditions.