SE4950 – Fall 2019
Project 3 – Static Analysis Lab 1 – Java/SpotBugs

Background

Static code analysis is a type of program analysis that is performed without actually executing programs but rather examining the source or object code. There are many commercial and open-source tools designed to inspect code and report found issues. In this lab, students will use one of them, SpotBugs, to find bugs in a given Java project.

The objective of this lab is to get some hands-on experience in how to use a Static Analysis tool with Java and how to interpret reported issues so that it can help improve the quality of the code.

Analysis tool:

The tool we will be using in this lab is SpotBugs version 3.1.5 and since we will be working with a Java project in Eclipse, we will use its Eclipse plugin. SpotBugs is a Static code analysis tool that analyses Java bytecode and detects a wide range of problems based on the concept of bug patterns. We will be running SpotBugs with Java 8.

System under test (SUT):

The project that we will be testing in this lab is HospitalSystem 1.0 which is a simple Java project for checking in and out patients and doctors and assigning patients to the right doctors.

More information from the document "HospitalSystem Overview and Tool Setup"

Credits: Lab adapted from lab materials from University of Tartu Software Testing Course

HospitalSystem Overview

HospitalSystem is a simple Java application for managing a hospital by checking in and out patients and doctors. This project was created as a sample project for this lab, therefore it should not be looked at as a complete system.
HospitalSystem allows you to:

- Add a doctor who is then automatically assigned to his favourite room. If the room is occupied, he/she is assigned to the first room available.
- Add a patient who is then automatically assigned to a doctor's waiting list with the desired specialty. If no such doctor is checked in, the patient is assigned to a general practitioner or, if no general practitioner is checked in, to the doctor with the shortest waiting list.
- Check out a doctor whose room is then freed and all patients added to the waiting list of the doctor with the shortest queue.
- Check out a patient.
- All checked in patients and doctors can be seen in the Overview tab.
- An overview of the rooms in the hospital can be seen in the Running rooms tab.

Task 1 – VM Installation and Bug Display

Install the Lab VM on your machines, it is provided on Google Drive and is an Open Virtualization Archive (.ova). It can be imported into either Virtual Box or VMWare. Note, if you encounter errors importing it into VMWare, click the Retry button, and it should work fine.

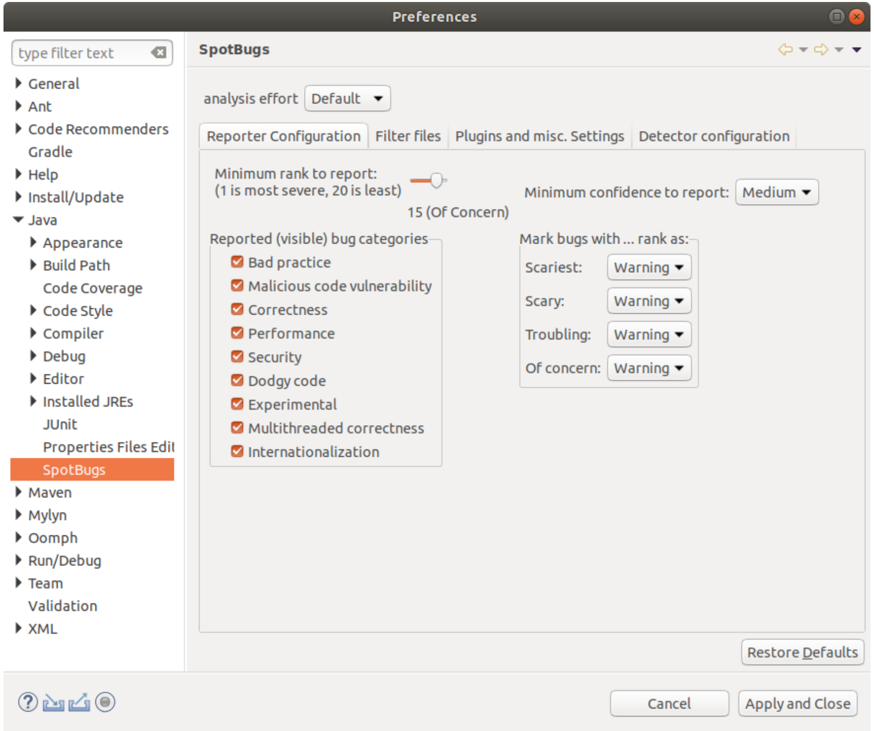
https://drive.google.com/file/d/1SMBGdAc4IhZz1svL70KZ8gl5dW229XqQ/view

Once the VM has been imported into the Virtualization software, startup the VM.

To log into the student account on the VM, the password is: letmein

Startup Eclispe, which is in the Favorites bar on the left side of the screen and run SpotBugs on the HospitalSystem project.

SpotBugs is configured to find all available bug categories:



SpotBugs

analysis effort: Default

Minimum rank to report:
(1 is most severe, 20 is least) 15 (Of Concern)

Minimum confidence to report: Medium

Reported (visible) bug categories

- ☑ Bad practice
- ☑ Malicious code vulnerability
- ☑ Correctness
- ☑ Performance
- ☑ Security
- ☑ Dodgy code
- ☑ Experimental
- ☑ Multithreaded correctness
- ☑ Internationalization

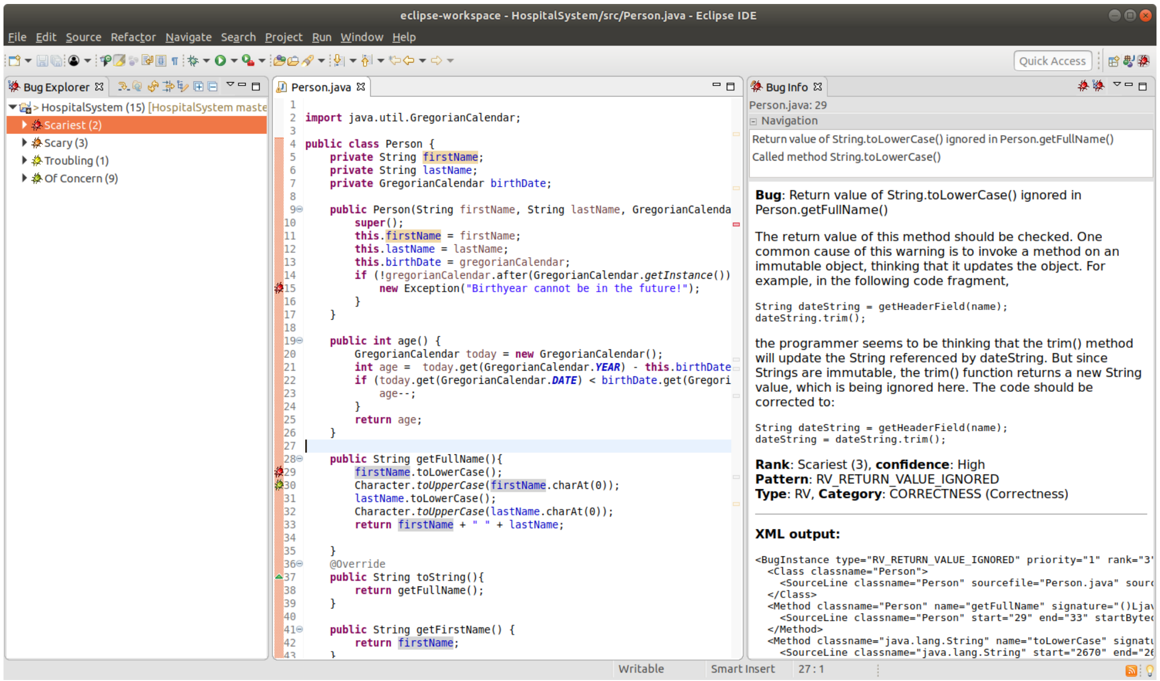
Mark bugs with ... rank as:

Scariest:	Warning
Scary:	Warning
Troubling:	Warning
Of concern:	Warning

To run SpotBugs on the project:

1) Right click on the “HospitalSystem” project in the package explorer.
2) Select “SpotBugs > Find Bugs” from the context menu.
3) Wait for SpotBugs to finish.
4) In the top right corner select the SpotBugs perspective (Bug Icon)

You should now see the SpotBugs perspective.


eclipse-workspace - HospitalSystem/src/Person.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Bug Explorer
Person.java
Bug Info
Writable
Smart Insert
27 : 1

HospitalSystem (15) [HospitalSystem maste
- Scariest (2)
- Scary (3)
- Troubling (1)
- Of Concern (9)

```java

import java.util.GregorianCalendar;

public class Person {
    private String firstName;
    private String lastName;
    private GregorianCalendar birthDate;

    public Person(String firstName, String lastName, GregorianCalenda
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = gregorianCalendar;
        if (!gregorianCalendar.after(GregorianCalendar.getInstance())
            new Exception("Birthyear cannot be in the future!");
        }
    }

    public int age() {
        GregorianCalendar today = new GregorianCalendar();
        int age =  today.get(GregorianCalendar.YEAR) - this.birthDate
        if (today.get(GregorianCalendar.DATE) < birthDate.get(Gregori
            age--;
        }
        return age;
    }

    public String getFullName(){
        firstName.toLowerCase();
        Character.toUpperCase(firstName.charAt(0));
        lastName.toLowerCase();
        Character.toUpperCase(lastName.charAt(0));
        return firstName + " " + lastName;

    }
    @Override
    public String toString(){
        return getFullName();
    }

    public String getFirstName() {
        return firstName;
    }
```

Person.java: 29

Navigation

Return value of String.toLowerCase() ignored in Person.getFullName()

Called method String.toLowerCase()

Bug: Return value of String.toLowerCase() ignored in Person.getFullName()

The return value of this method should be checked. One common cause of this warning is to invoke a method on an immutable object, thinking that it updates the object. For example, in the following code fragment,

```
String dateString = getHeaderField(name);
dateString.trim();
```

the programmer seems to be thinking that the trim() method will update the String referenced by dateString. But since Strings are immutable, the trim() function returns a new String value, which is being ignored here. The code should be corrected to:

```
String dateString = getHeaderField(name);
dateString = dateString.trim();
```

Rank: Scariest (3), **confidence**: High
Pattern: RV_RETURN_VALUE_IGNORED
Type: RV, **Category**: CORRECTNESS (Correctness)

XML output:

```
<BugInstance type="RV_RETURN_VALUE_IGNORED" priority="1" rank="3"
  <Class classname="Person">
    <SourceLine classname="Person" sourcefile="Person.java" sourc
  </Class>
  <Method classname="Person" name="getFullName" signature="()Ljav
    <SourceLine classname="Person" start="29" end="33" startBytec
  </Method>
  <Method classname="java.lang.String" name="toLowerCase" signatu
    <SourceLine classname="java.lang.String" start="2670" end="26
```

Explore Bugs

You can see all the information about bugs that have been found by SpotBugs in the project code in the SpotBugs perspective. To see a list of the problems, click on the small arrow in front of “HospitalSystem” in the “Bug Explorer” pane.

SpotBugs categorizes bugs in three main ways.

1. **Rank** - a bug’s rank measures the severity of the problem. It varies from 20 (lowest) to 1 (highest) and the values are grouped into four categories: Of Concern, Troubling, Scary and Scariest.
2. **Confidence** - a bug’s confidence measures the likelihood that a real bug was flagged rather than a false-positive.
3. **Category** - a bug’s category notes the bug type. There are nine categories in total, some examples being Correctness, Bad Practice, Performance etc. In addition, SpotBugs matches every bug with a pattern. If interested, you can read more about patterns here: https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html

In Eclipse you can find individual bugs by navigating through the hierarchy of found problems.

After clicking on the bug description once, SpotBugs navigates you to the location of the problem in code and further information about it is located in the “Bug Info” tab

Task 2 – Analyze Bugs

When analyzing an issue reported by SpotBugs, the most important thing is to figure out whether an actual fault or a false positive was found.

Your task is to analyze 5 bugs of your choice found by the SpotBugs tool in the HospitalSystem project. Choose the bugs from all different ranks (Of Concern, Troubling, Scary, Scariest) and confidences (Low, Normal, High) and try to decide whether the given bug is an actual fault or a false-positive. In case of an actual fault, try to think of a solution to the problem.
For each bug fill out the following:

- Bug description and location
- What seems to be the problem?
- Is this a false or a true positive? Should this bug be fixed? If yes, then how, otherwise why not?

Please ask questions about bugs if you don’t know the details about a particular type of Java error, and we will discuss it in class.

Grading

The project will be worth 50 points.