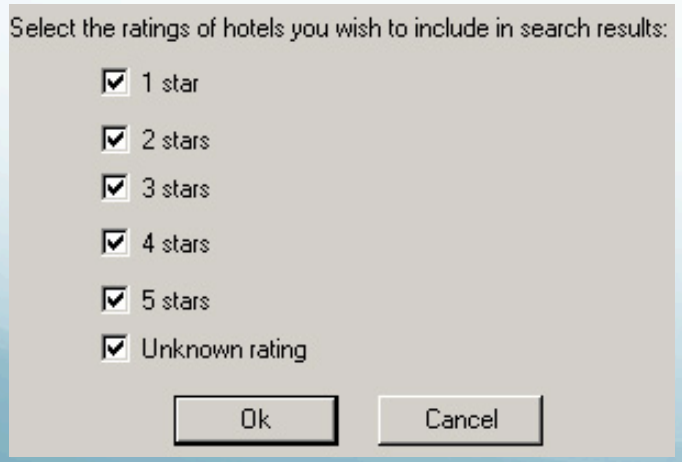
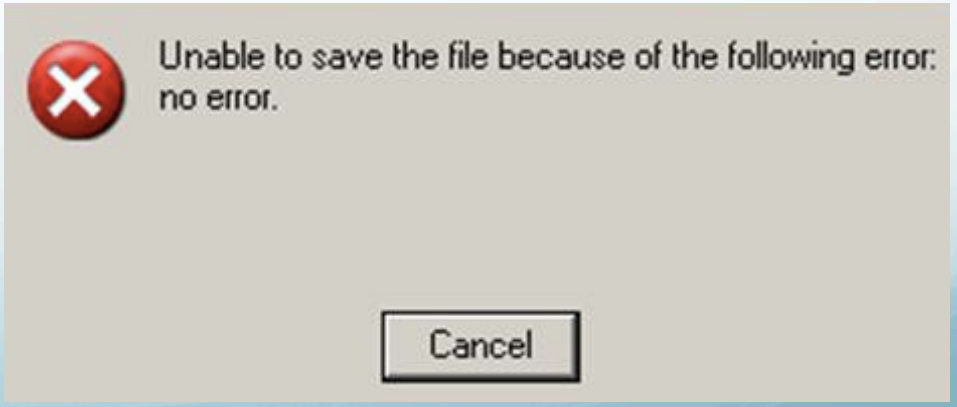
FindBugs

COMP3111/H tutorials

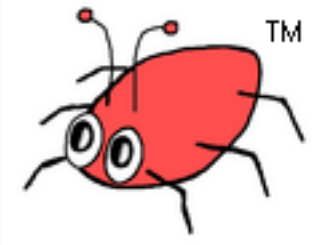
What is a bug?

- Program behavior that deviates from its specification
- A bug does not include
  - Poor performance, unless a threshold level of performance is included as part of the specification
  - An awkward or inefficient user interface





What is Findbugs?



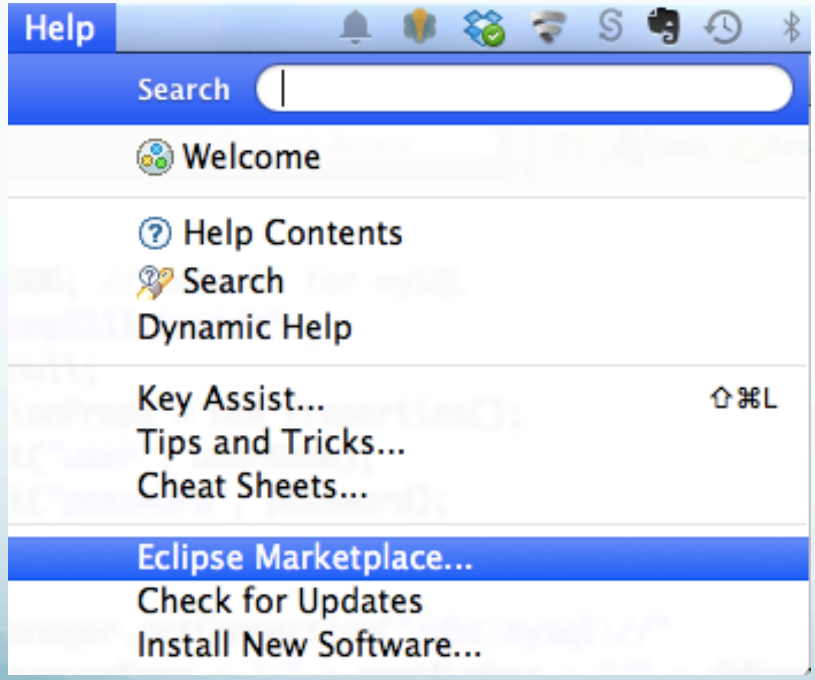
- FindBugs™ is a program to find bugs statically in Java programs
  - It works by analyzing Java bytecode (class files)
  - It can report false warnings
- Static program analysis
  - Analysis of program without actually executing it
- FindBugs fact sheet
  - http://findbugs.sourceforge.net/factSheet.html

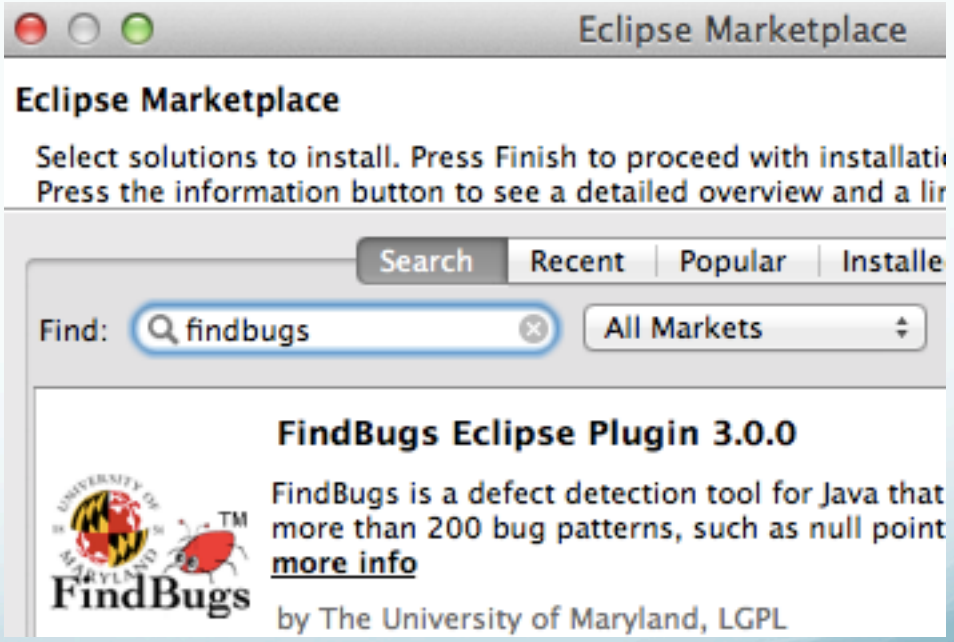
Why using Findbugs?

- Designed to work with Java
- Free and open source
- Integrate well with Eclipse
- Very easy to use
- Inspect occurrences of bug patterns
  - Bug Patterns in Java by Eric Allen Apress © 2002 (234 pages) ISBN:9781590590614

Findbugs + Eclipse

- For newer version of Eclipse, open Eclipse Marketplace and then search Findbugs

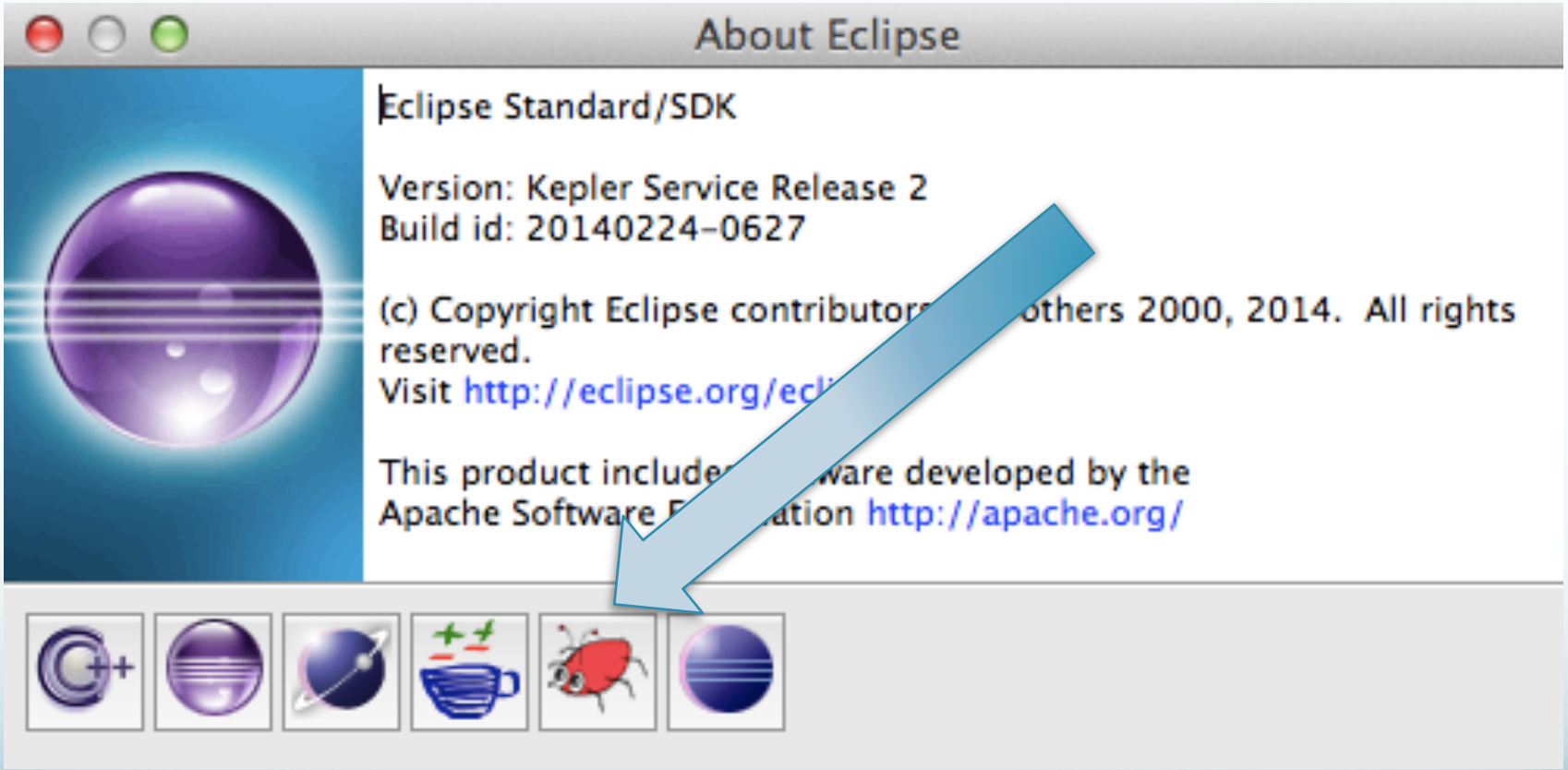




Findbugs + Eclipse

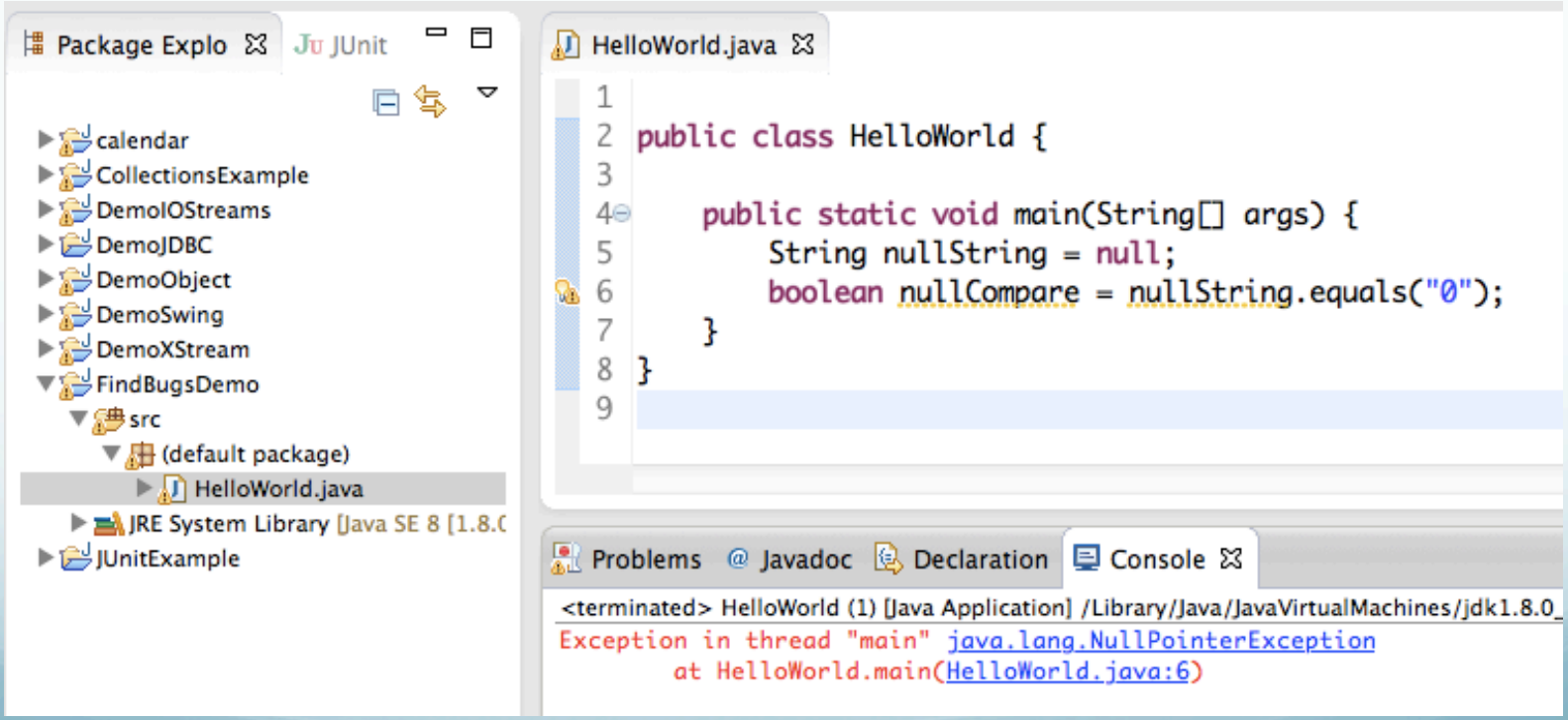
- For older versions of Eclipse
  - Help -> Software Update -> Find and Install...
  - Choose the Search for new features to install option, and click Next
  - Click New Remote Site.
  - Enter the following:
    - Name: FindBugs update site
    - URL: http://findbugs.cs.umd.edu/eclipse
  - Reference:
    - http://findbugs.sourceforge.net/manual/eclipse.html

Findbugs is installed



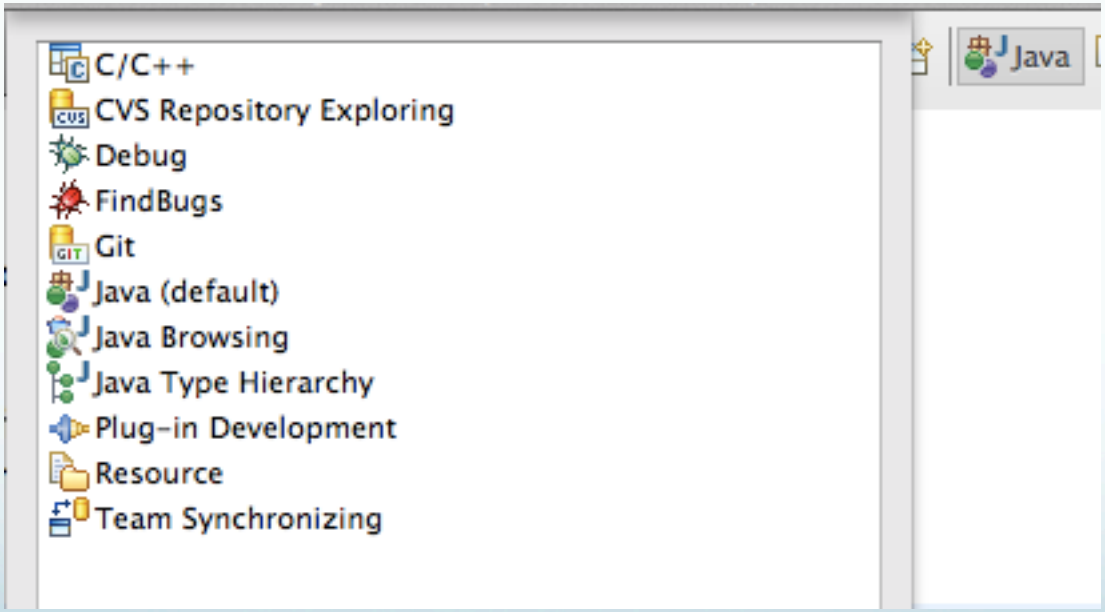
HelloWorld in Findbugs

- Example: The following program will generate a null pointer exception. This example is used to illustrate how to use Findbugs



Findbugs Perspective

- To use Findbugs, switch to Findbugs Perspective
  - By default, Eclipse is configured to run on Java Perspective

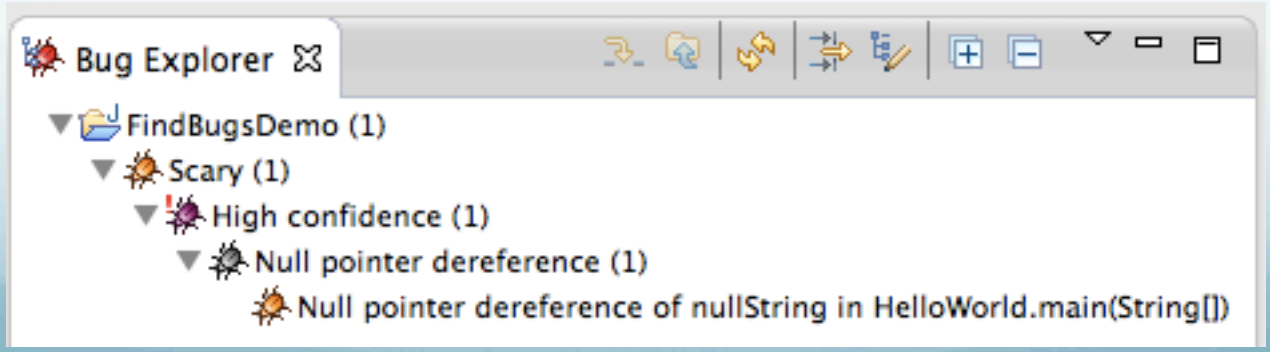


Using FindBugs

- In an Editor window, right-click “HelloWorld.java”
- Select “Find Bugs > Find Bugs”

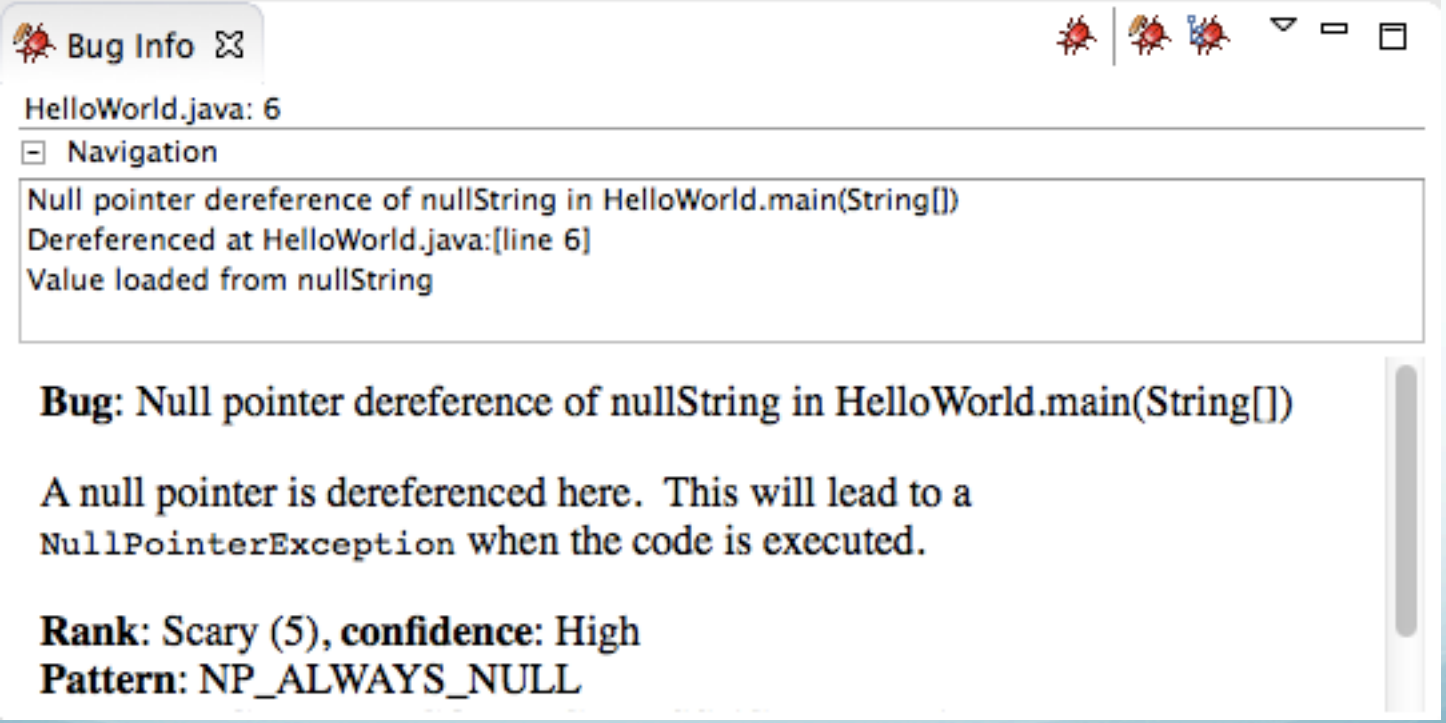
Bug Explorer

- A bug report is generated on the bug explorer window to summarize bugs identified by Findbugs
- In this example, a null pointer deference of “nullString” is detected, with a high level of confidenence
- Double-clicking the bug icon moves to the buggy line of code



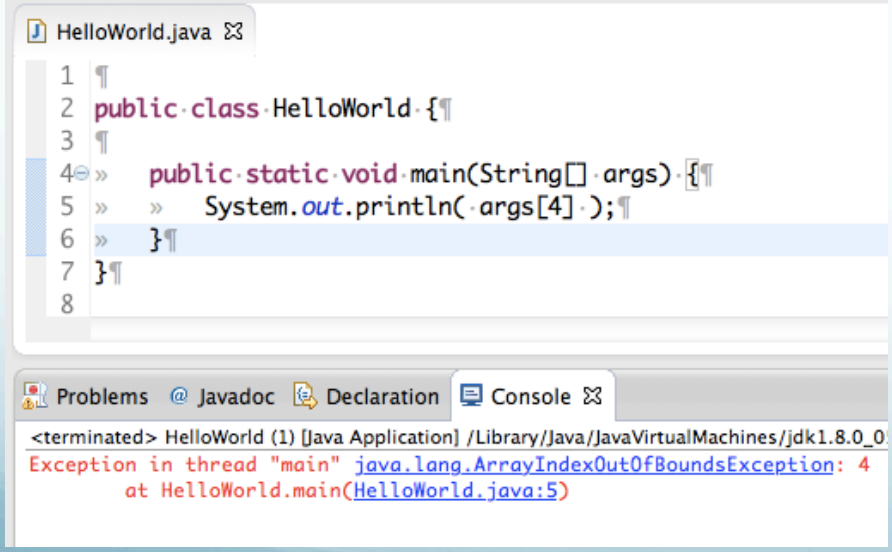
Bug Info

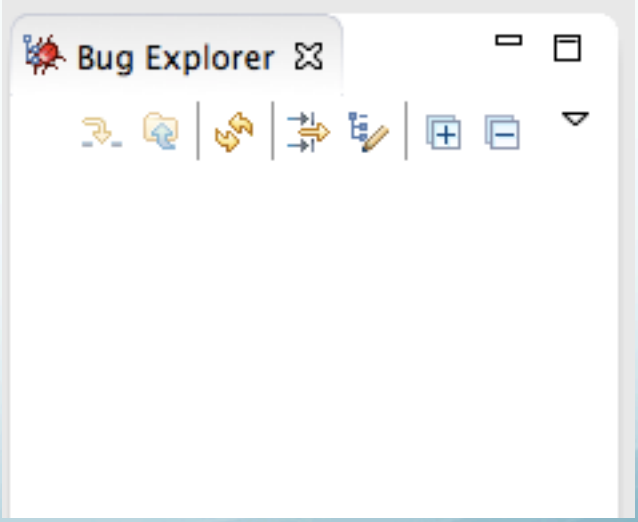
- A Bug Info window helps programmers understand the potential causes and the rank of bug



Findbugs – False Negative

- False negative
  - There is a bug, but it is unable to be identified
- Example: Programmers should check the array index, but no bug is detected by FindBugs





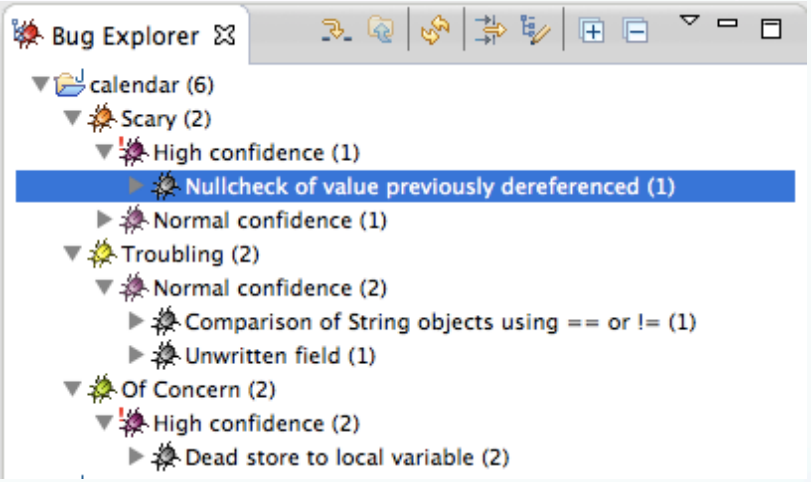
Findbugs – False positive

- False positive
  - There is no bug on a programming statement, but a bug is detected by Findbugs
  - This may not be a good example, but it demonstrates that an execution path is guaranteed not to be executed (i.e. if (false)), but a bug is still reported

```
public class HelloWorld {
    public static void main(String[] args) {
        if (false) {
            String nullString = null;
            boolean nullCompare = nullString.equals("0");
        }
    }
}
```

Bugs can be identified from our calendar project!

- Findbugs can be slow on a large software project
  - It can be configured to run in the background
- The base code provided in our calendar project is not perfect
- Some bugs (and one of them can be a serious bug) can be identified by using FindBugs



Example: Findbugs in Calendar base code

- It is a very common mistake
- A programmer checked null of an object, but later he/she/another programmer added implementation on top of this null checking code
- Solution
  - Move the null checking logic to the top of method

AppList.java

calendar ▸ src ▸ hkust.cse.calendar.gui ▸ AppList ▸

```
330     // colouring the appointment list
331     public void addAppt(Appt appt) {
332         Color color;
333         currColor = new Color(0,240-(appt.TimeSpa
334         currColorForJoint = new Color(255-(appt.T
335         if(!appt.isJoint())
336             color = currColor;
337         else
338             color = currColorForJoint;
339
340         if (appt == null)
341             return;
342
```

Example: Findbugs in Calendar base code

- Another common bug on string comparison in Java
- Most C++ programmers will have this problem!
- Solution: Uses !equals(anotherString)

```
526
527         if (tableView.getModel().getValueAt(currentRow, currentCol) != "")
528             try {
529                 currentD = new Integer((String) tableView.getModel()
530                         .getValueAt(currentRow, currentCol)).intValue();
531             } catch (NumberFormatException n) {
```

Bug: Comparison of String objects using == or != in hkust.cse.calendar.gui.CalGrid.mouseResponse()

This code compares `java.lang.String` objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the `String.intern()` method, the same string value may be represented by two different String objects. Consider using the `equals(Object)` method instead.

Example: Findbugs in Calendar base code

- A common mistake is to have some unused variables
- They can be detected as warning messages, but they are usually ignored (as there are thousands of warnings)

AppList.java | AppScheduler.java | CalGrid.java

calendar ▸ src ▸ hkust.cse.calendar.gui ▸ CalGrid

```
557     }
558
559     public Appt[] GetTodayAppt() {
560         Integer temp;
561         temp = new Integer(currentD);
562         Timestamp start = new Timestamp(0);
563         start.setYear(currentY);
564         start.setMonth(currentM-1);
```

Bug: Dead store to temp in hkust.cse.calendar.gui.CalGrid.GetTodayAppt()

This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.

Summary

- Resolving all problems in Findbugs does not mean that the software program is bug-free
- Findbugs is only a tool to help programmers detect bugs, it is not a perfect tool (e.g. false positive and false negative problems)