Lecture 14 — FindBugs

David J. Pearce

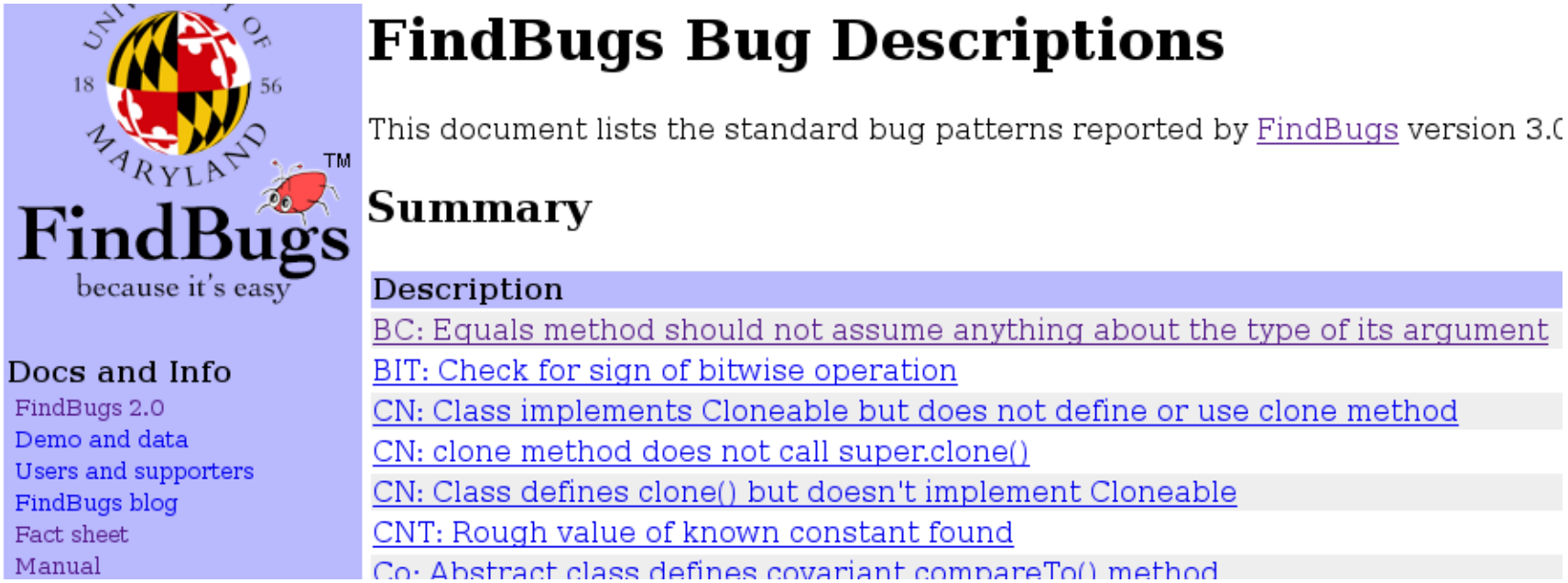
School of Engineering and Computer Science
Victoria University of Wellington

What is FindBugs?

"FindBugs is an open source program created by Bill Pugh and David Hovemeyer which looks for bugs in Java code. **It uses static analysis to identify hundreds of different potential types of errors in Java programs**. *Potential errors are classified in four ranks: (i) scariest, (ii) scary, (iii) troubling and (iv) of concern. This is a hint to the developer about their possible impact or severity. FindBugs operates on Java bytecode, rather than source code."*

–Wikipedia

Bug Patterns



FindBugs

because it's easy

Docs and Info

FindBugs 2.0
Demo and data
Users and supporters
FindBugs blog
Fact sheet
Manual

FindBugs Bug Descriptions

This document lists the standard bug patterns reported by FindBugs version 3.0

Summary

Description
BC: Equals method should not assume anything about the type of its argument
BIT: Check for sign of bitwise operation
CN: Class implements Cloneable but does not define or use clone method
CN: clone method does not call super.clone()
CN: Class defines clone() but doesn't implement Cloneable
CNT: Rough value of known constant found
Co: Abstract class defines covariant compareTo() method

- FindBugs looks for **Bug Patterns**:
  - (Also called **code smells**)
  - Patterns don't **guarantee** bugs
  - Patterns indicate **potential** bugs
  - Patterns are **simple** to check, hence typically **superficial**

Bug Patterns (Cont'd)

```
class Point {
  private int x, y;

  public Point(int x, int y) {
    this.x = x; this.y = y;
  }

  public boolean equals(Point p) {
     return this.x == p.x && this.y == p.y;
  }
  ...
```

- **Pattern 1**: Equals method should not assume anything about type of its argument.

Bug Patterns (Cont'd)

```
class Point {
  private int x, y;

  public Point(int x, int y) {
    x = x; y = y;
  }

 ...
}
```

- **Pattern 2**: Self Assignment of local rather than assignment to field.

Bug Patterns (Cont'd)

```
class Point {
  private int x, y;

  public void add(Point p) {
    if(p == null) { x = x + p.x; y = y + p.y; }
  }
  public boolean lessThan(Point p) {
    return p != null || (x < p.x && y < p.y);
  }
  ...
}
```

- **Pattern 3**: Null pointer dereference.

  (Eclipse checks for these now)

Bug Patterns (Cont'd)

```
class Point {
  private int x, y;

  public Point(int x, int y) { ... }

  public boolean equals(Object o) {
     if(o instanceof Point) {
       Point p = (Point) o;
       return this.x == p.x && this.x == p.x;
     } else {
       return false;
  }  }
  ...
```

- **Pattern 4**: Repeated conditional tests.

Bug Patterns (Cont'd)

```
class Point {
  private int x, y;

  public Point(int x, int y) { ... }

  public boolean equals(Object o) {
     if(o instanceof Point) {
       Point p = (Point) o;
       return this.x == p.x && this.y == p.y;
     } else {
       return false;
} } }
```

- **Pattern 5**: Class defines `equals()` but not `hashCode()`.

Bug Patterns Categories

- **Class & Inheritance Structure**. These bug detectors look only at structural properties.
- **Linear Code Scan**. These make linear scan through functions, ignoring control flow.
- **Control-Flow Sensitive**. These make use of control flow graph to find bugs.
- **Control & Data Flow**. These model both data and control-flow to find bugs.

See *"Finding Bugs is Easy"*, Hovemeyer and Pugh.

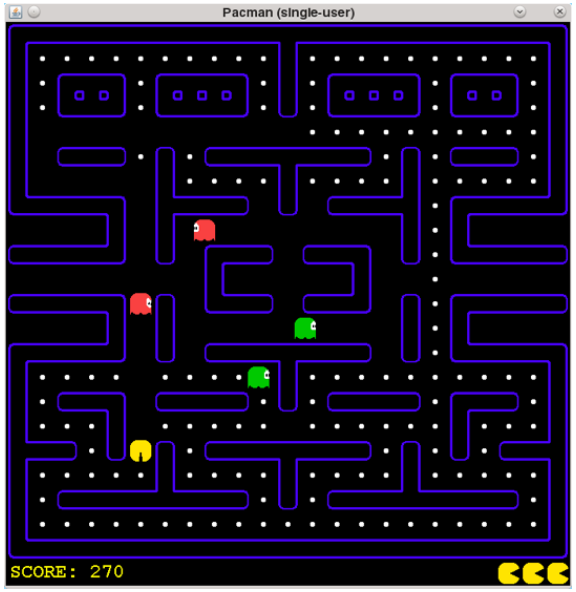
Soundness

- **False Positives**. A warning reported where no problem actually exists in practice.
- **False Negatives**. A problem actually exists, and the warning should have been reported (but was not).

FindBugs suffers both issues.

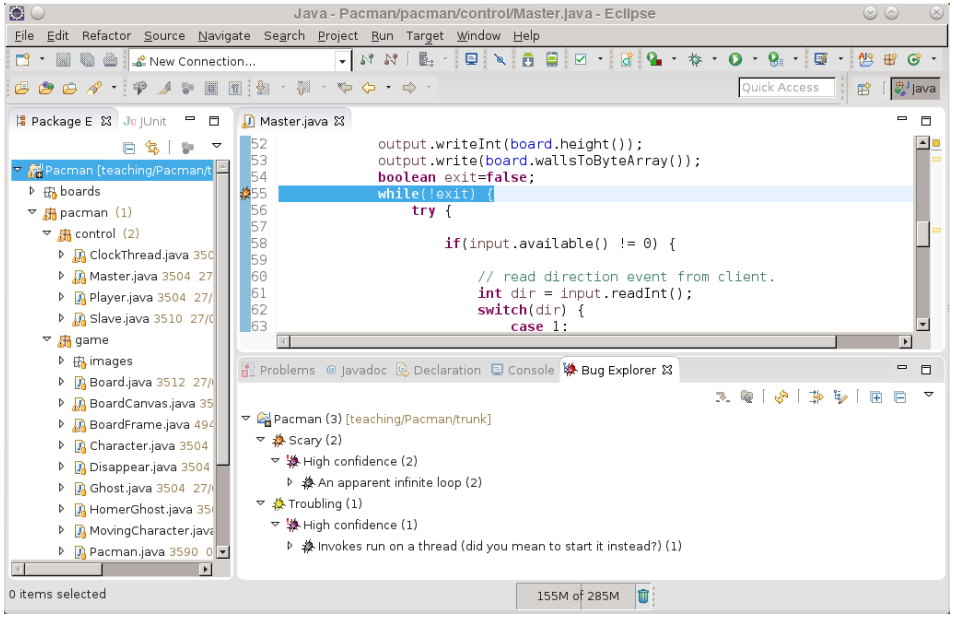
Example: Pacman

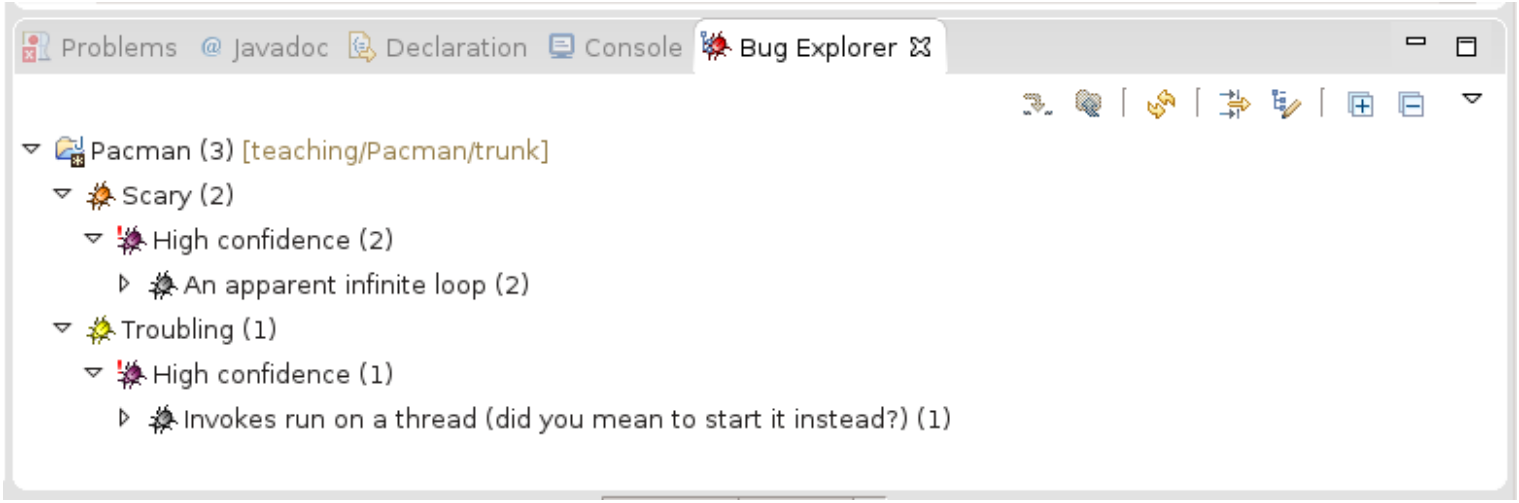
Metrics:



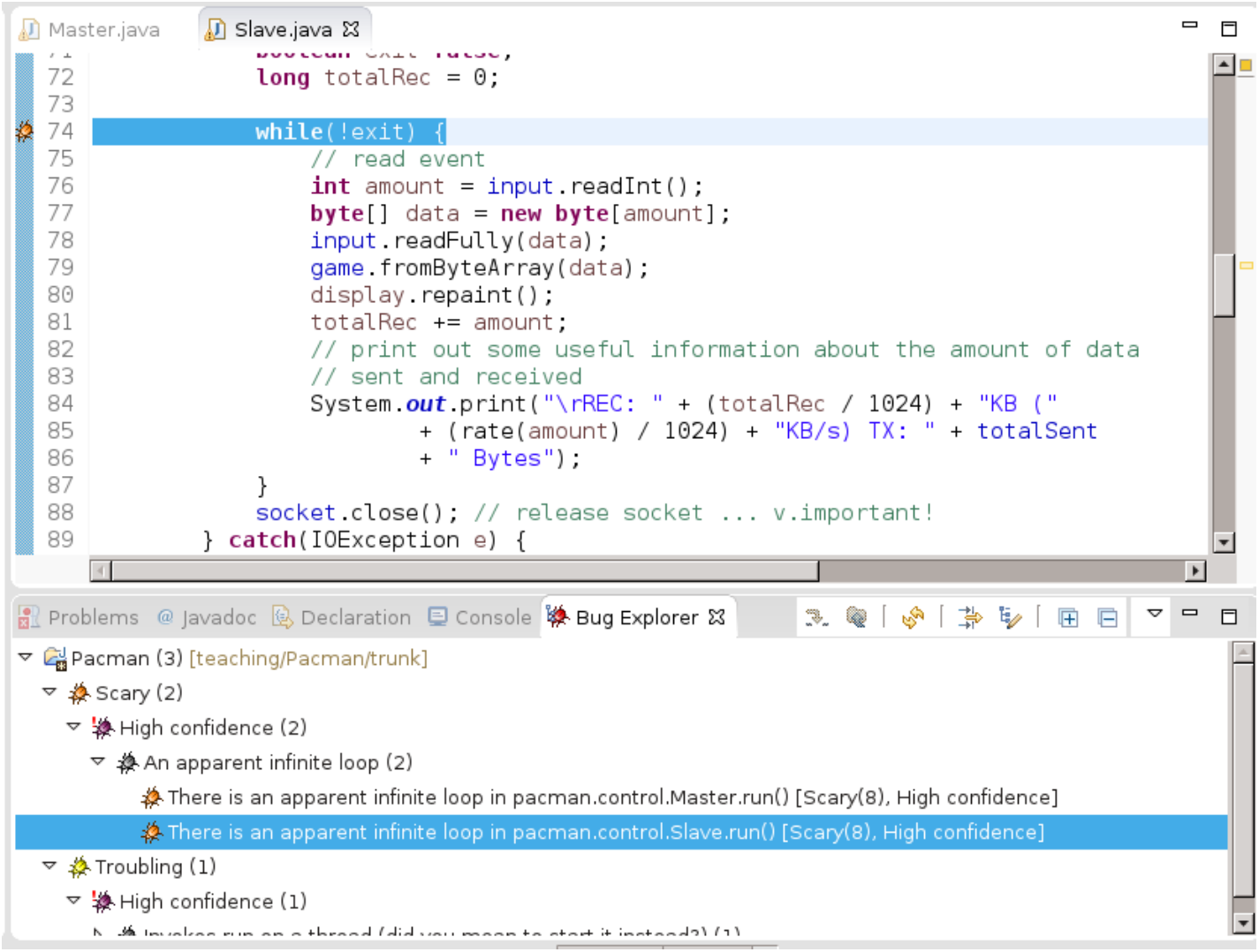
- **2458** Lines of Code (LOC) including whitespace
- **1502** Lines of Code (LOC) excluding whitespace, 722 Comment Lines
- **15** Classes
- **4** Packages

Example: Pacman (Cont'd)



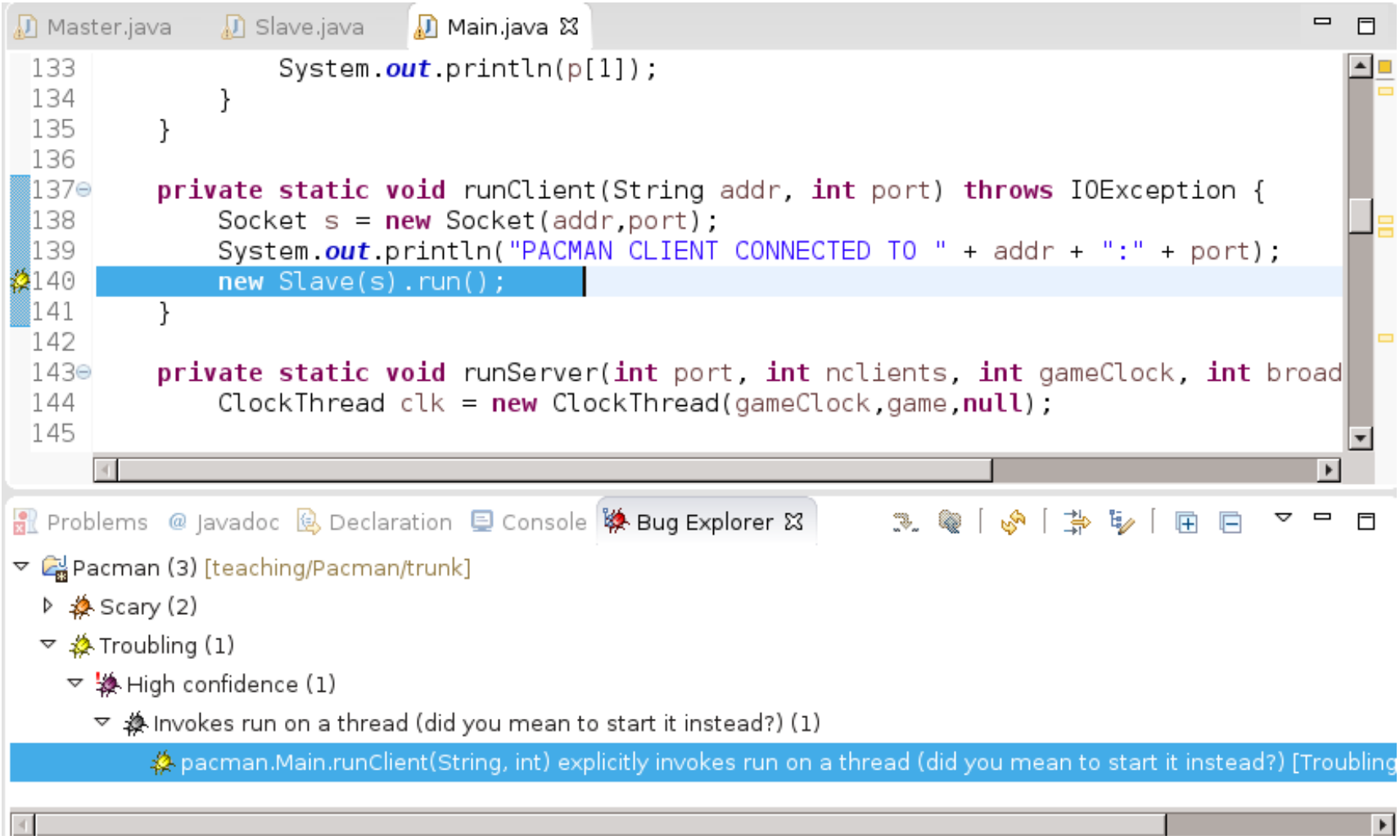


Example: Pacman (Cont'd)



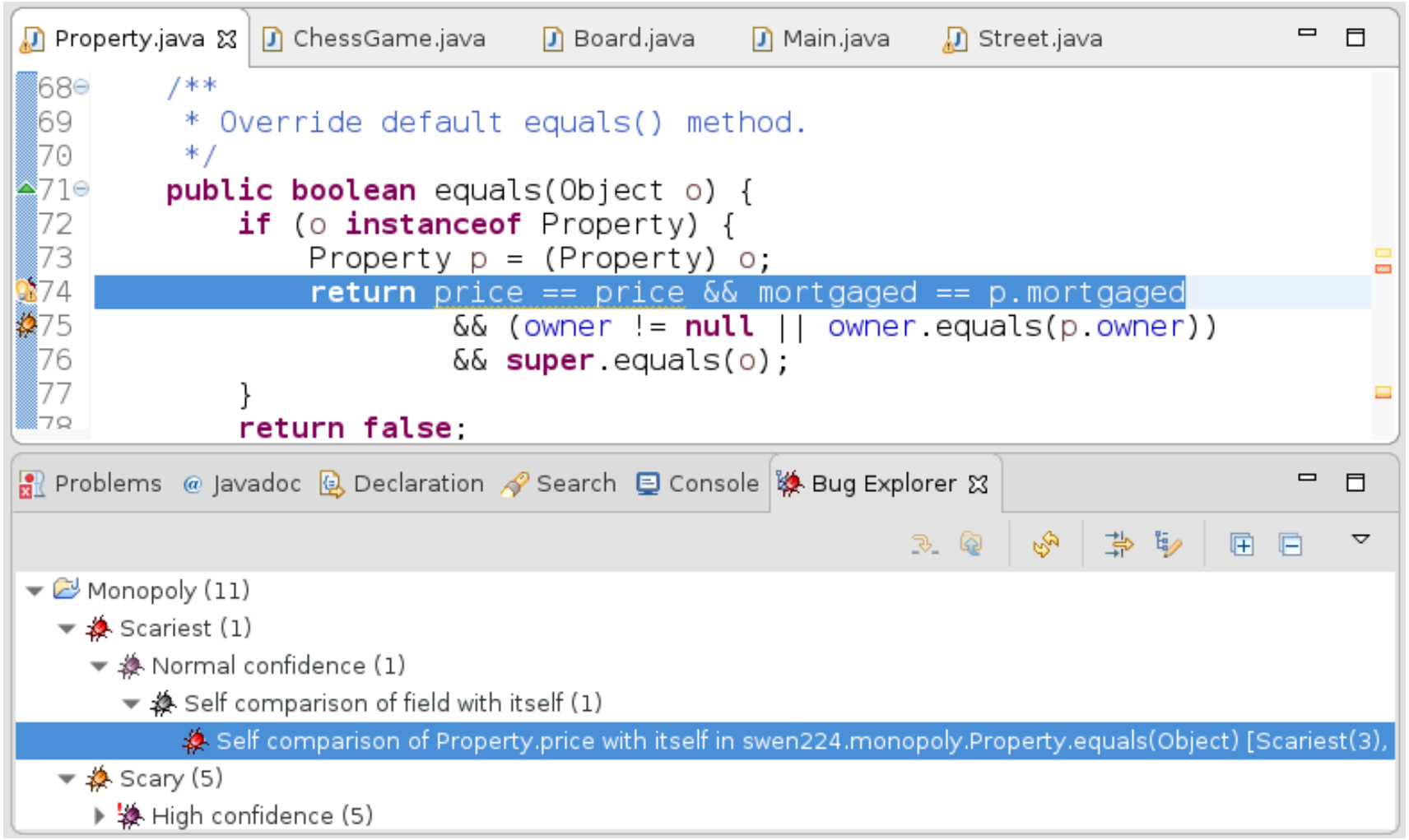
Q) *Why does FingBugs think there is an infinite loop?*

Example: Pacman (Cont'd)



Q) *Do you agree with FindBugs?*

Example: Monopoly



Further Reading

- *Finding Bugs is Easy*, Hovemeyer & Pugh, OOPSLA 2004.
- *Using Static Analysis for Software Defect Detection*, Bill Pugh, Google Talk, 2006. `https://youtu.be/8eZ8YWVl-2s`
- *Using Static Analysis to Find Bugs*. Ayewah, Hovemeyer, Morgenthaler, Penix & Pugh, IEEE Software, 2008.
- *A comparison of bug finding tools for Java*, Rutar, C.B. Almazan & J.S. Foster, ISSRE'04.
- *Scaling static analyses at Facebook*, Communications of the ACM, 2019.
- *Lessons from building static analysis tools at Google*, Communications of the ACM, 2018.
- *Why don't software developers use static analysis tools to find bugs?* Johnson, Song, Murphy-Hill & Bowdidge, ICSE'13.