# Visibility

```
public class Printer extends Thread
{
  public void run()
  {
    while (!Main.ready)
      System.out.println(".");
  }
}
public class Main
{
  public static boolean ready = false;
  public static void main(String[] args)
  {
    new Printer().start();
    Main.ready = true;
  }
}
```

# Visibility

```
public class Printer extends Thread
{
  public void run()
  {
    while (!Main.ready)
      System.out.println(Main.number); // modified
  }
}
public class Main
{
  public static boolean ready = false;
  public static int number = 0; // new
  public static void main(String[] args)
  {
    new Printer().start();
    Main.ready = true;
    Main.number = 1; // new
  }
}
```

# Visibility

Synchronized methods and blocks can be used to guarantee that one thread sees the effects of another in a predictable manner. When thread A executes a synchronized block, and subsequently thread B enters a synchronized block guarded by the same lock, the values of variables that were visible to A prior to releasing the lock are guaranteed to be visible to B upon acquiring the lock.

# Software . . .

. . . is everywhere . . .

. . . yet is full of bugs.

# Bugs

"A computer malfunction at Bank of New York brought the Treasury bond market's deliveries and payments systems to a near standstill for almost 28 hours . . . it seems that the primary error occurred in a messaging system which buffered messages going in and out of the bank. The actual error was an overflow in a counter which was only 16 bits wide, instead of the usual 32. This caused a message database to become corrupted. The programmers and operators, working under tremendous pressure to solve the problem quickly, accidentally copied the corrupt copy of the database over the backup, instead of the other way around."

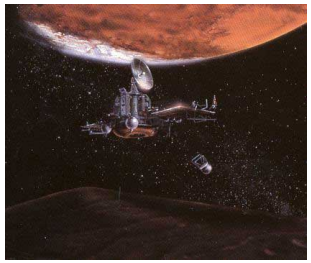
Wall Street Journal, November 25, 1985

Bank of New York

# Bugs

"The Soviet Mars probe was mistakenly ordered to commit suicide when ground control beamed up a 20 to 30 page message in which a single character was inadvertently omitted. The change in program was required because the Phobos 1 control had been transferred from a command center in the Crimea to a new facility near Moscow. The changes would not have been required if the controller had been working the computer in Crimea. The commands caused the spacecraft's solar panels to point the wrong way, which would prevent the batteries from staying charged, ultimately causing the spacecraft to run out of power."

comp.risks, September 13,1988

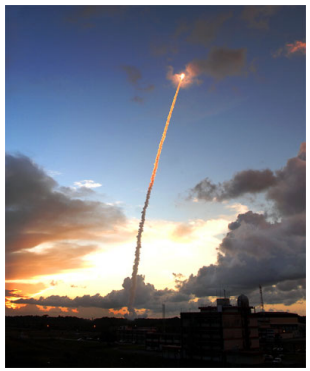


Phobos 1

# Bugs

"On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 meters, the launcher veered off its flight path, broke up and exploded. ... The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception. ... The data conversion instructions (in Ada code) were not protected from causing an operand error, although other conversions of comparable variables in the same place in the code were protected."

Report of the Ariane Inquiry Board



Ariane 5

# Bugs

“To correct an anomaly that caused inaccurate results on some high-precision calculations, Intel Corp. last week confirmed that it had updated the floating-point unit (FPU) in the Pentium microprocessor. The company said that the glitch was discovered midyear and was fixed with a mask change in recent silicon. “This was a very rare condition that happened once every 9 to 10 billion operand pairs,” said Steve Smith, a Pentium engineering manager at Intel.”

EE Times, November 7, 1994


Pentium

# Bugs

“One of the more startling problems concerned the program’s handling of the Greenwich meridian. The National Airspace Package, designed by IBM’s Federal Systems division, contains a model of the airspace it controls, that is, a map of the airlanes and beacons in the area. But, because the program was designed for air traffic control centres in the US, the designers had taken no account of a zero longitude; the deficiency caused the computer to fold its map of Britain in two at the Greenwich meridian, plonking Norwich on top of Birmingham.”

Peter H. Roosen-Runge. Software Verification Tools. 2000.

# Bugs

“In November 1992, the computerized dispatch system for the London (England) Ambulance Service, broke down suddenly and ‘locked up altogether’, leaving calls for assistance which had already been stored in the system unavailable, and forcing a taxing and cumbersome return to a manual, paper-based dispatch system. Luckily, the breakdown occurred in early morning hours, so that there were no severe consequences for patients.”

Peter H. Roosen-Runge. Software Verification Tools. 2000.

# Bugs

"A clear example of the risks of poor programming and verification techniques is the tragic story of the Therac-25—one in a series of radiation therapy machines developed and sold over a number of years by Atomic Energy Canada Limited (AECL). As a direct result of inadequate programming techniques and verification techniques, at least six patients received massive radiation overdoses which caused great pain and suffering and from which three died."

Therac-25

Peter H. Roosen-Runge. Software Verification Tools. 2000.

# The Cost of Bugs

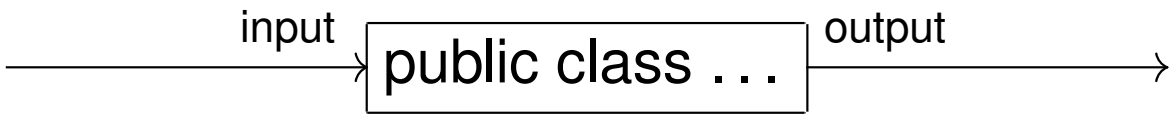
“The cost of software bugs to the U.S. economy is estimated at $60 billion per year.”

National Institute of Standards and Technology, 2002

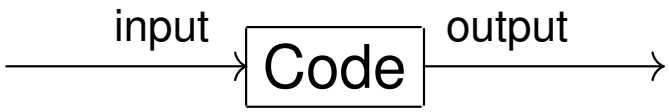
# Testing


- Provide the input.
- Run the code.
- Compare the output with the expected output.

# White Box Testing



# Black Box Testing



# Terminology

Test case: an input that satisfies the precondition.

Test suite/test vector: a collection of test cases.

# Which Test Cases?

- Likely cases (black box and white box testing).
- Boundary cases (black box and white box testing).
- Cases that cover all execution paths (white box testing only).

# How to Provide the Test Cases?

- Enter the test cases manually.
- Read the test cases from files.
- Generate the test cases by an app.
- Use a testing framework such as JUnit.

# How to Determine the Expected Result?

- Use a different solution to the problem that is known to be correct.
- Use an approximate solution to the problem.
- ...

# How to Compare the Result with the Expected Result?

- Check it manually.
- Read the expected result from a file.
- Generate the expected result by an app.
- Use a testing framework such as JUnit.

Sometimes, it is much easier checking that the output is correct than computing the output. For example, it is much easier checking that a list of elements is sorted than sorting a list of elements.