Debugging

Software Engineering

Chair of Programming Methodology

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Agenda for today

1. Debugging
- “Classical” vs. “Modern” technique
1. Main Concepts
- “Devil’s” vs. “Scientific” method
- Demo with NetBeans™
1. Tips for finding and fixing bugs
2. References

Debugging

- Testing detects the error
- Debugging identifies and corrects the root cause
  - typically identification is much harder

- Can take up to 50% of development time!
- Experience can make you three times faster!

Debugging is twice as hard as writing the code in the first place.

Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

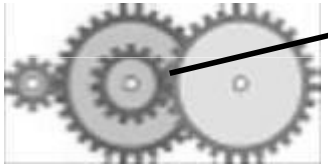
— Brian W. Kernighan

Cause-effect chain

defect in code

```
class C {
Vector vec;
void addElement (Element e) {
                    vec.add(e);
                }
}
```

infection in program state



vec uninitialised
when add invoked

visible as **failure**

Debugging

Cause-effect chain

- Not every defect causes a failure!

> *Testing can only show the presence of errors – not their absence.*
>
> — Dijkstra (1972)

- Every failure can be traced back to some infection
- Every infection is caused by some defect

- Debugging means to relate a given failure to the defect — and to remove the defect

“Classical” technique

- Print messages like
  - “I’m here!” to track control flow
  - “The value of x is 3” to track data flow
- Add toString methods to easily print object content
- if(Debug.debugLevel > 3)
  System.out.println(o);
- Still useful for finding simple defects quickly!

“Modern” technique

- Use of assert statements
  - in Java since 1.4
  - if fails: in debug mode throws exception, otherwise no effect
- Use of java.util.logging.Logger
- Use of debugging tools
  - interactive debugger
  - integrated into IDE
  - allows tracking of control and data flow

Main concepts

- Breakpoints
  - Stop execution at specified locations
- Inspecting program state
  - Peek into program state at a certain location
- Stepping through code
  - Interactively control execution of program

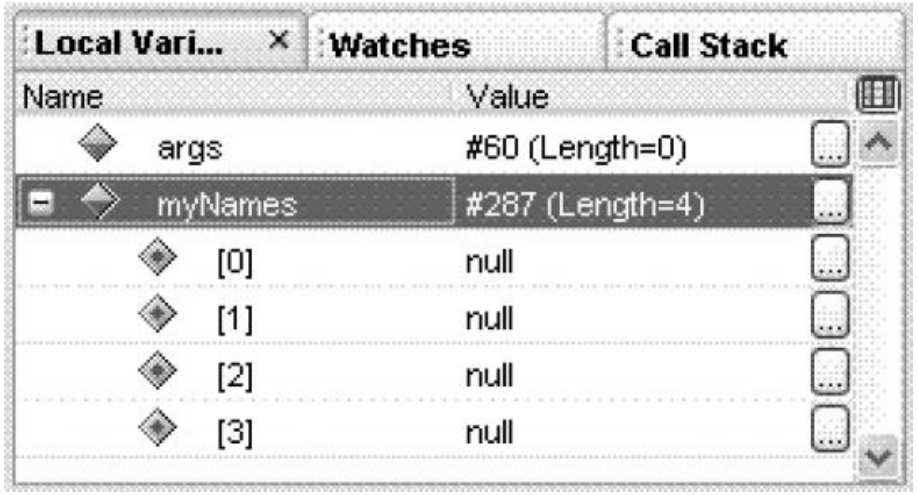
Breakpoints

- Stop execution at specified locations
- Location types
  - given method or line
  - variable or field accessed/modified
  - exception of specified type thrown, caught or not handled
  - thread starts or stops
  - class loaded into or unloaded from VM
  - conditional or unconditional

```
void creditCustomer(Customer c, double val) {
    c.balance += val;
    addToVIPIfRich(c);
}
```

Inspecting program state

- Peek into program state where debugger is at
- Values of fields, local variables or any expression
- Call stack
- Hierarchy of all loaded classes
- Threads of the program



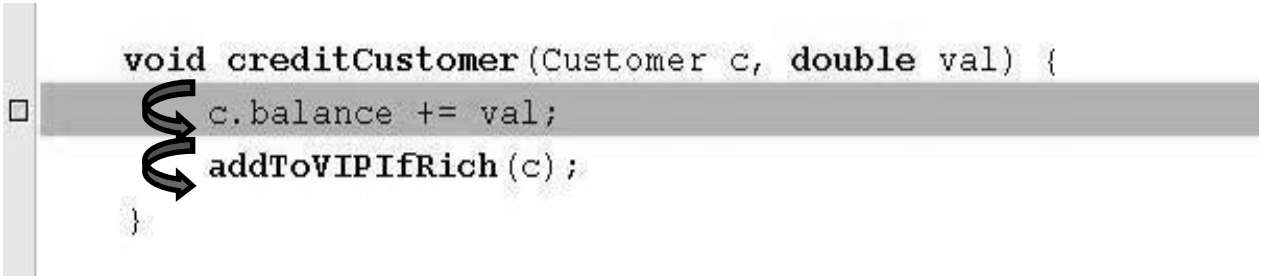
Stepping through code

- Interactively control execution of program

Stepping through code

- Step **over**, into or out

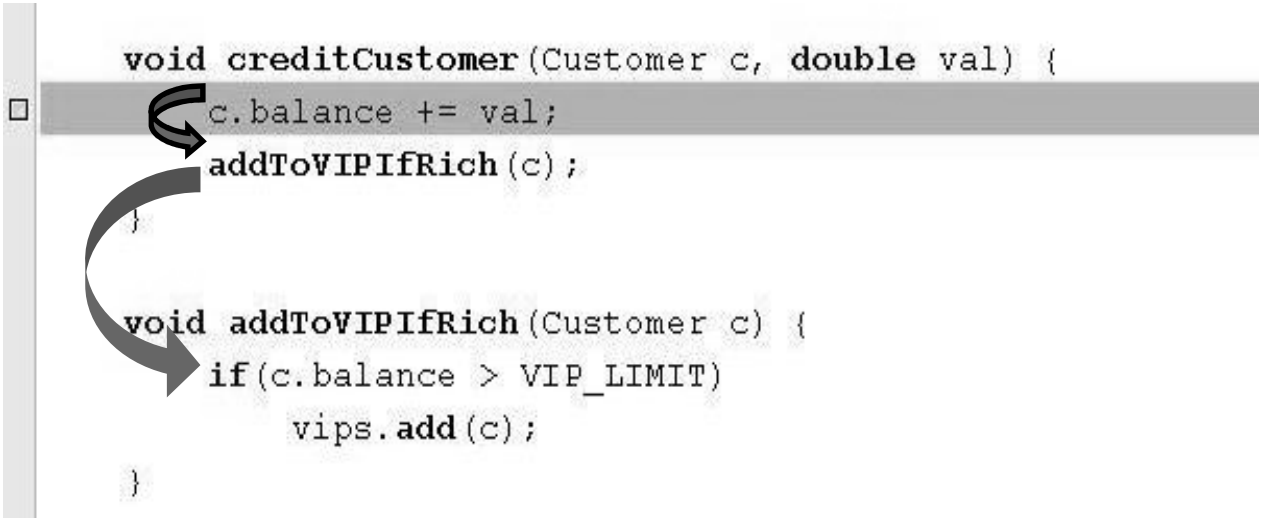
Executes one source line. If line contains method call, executes the entire routine.



Stepping through code

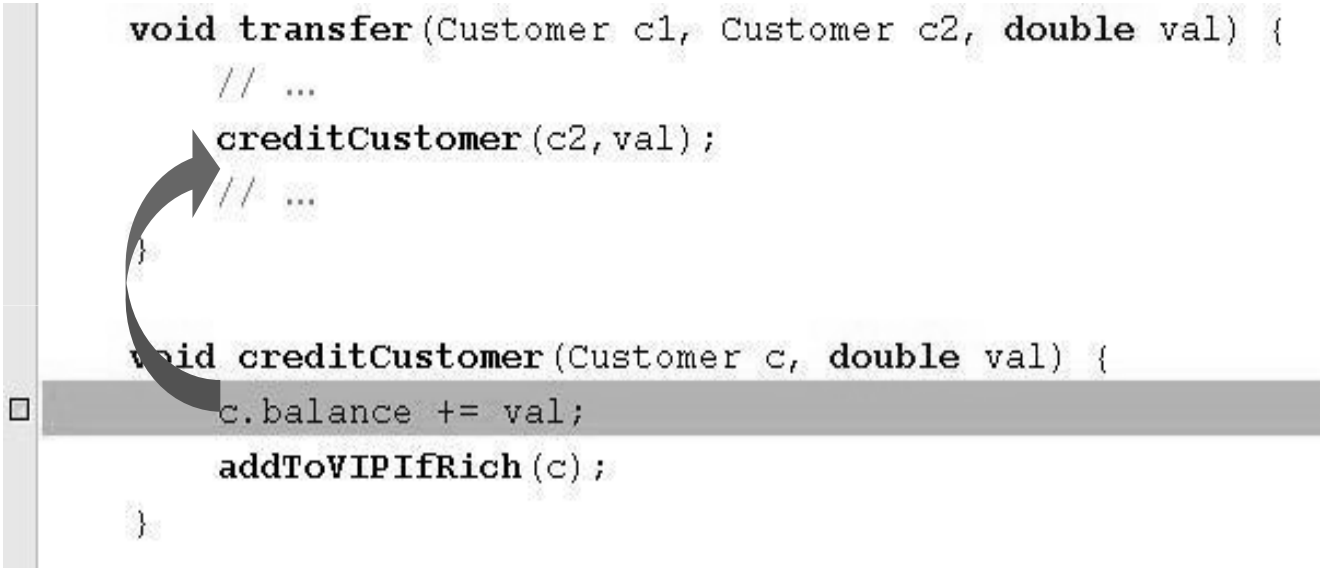
- Step over, **into** or out

Executes one source line. If line contains method call, stops just before executing the first statement of the routine.



Stepping through code

- Step over, into or **out**



Executes the remaining of the routine the line is part of and returns control to the caller of the routine.

Devil's approach

- Don't waste time on understanding the problem
- Find defect by guessing
- Fix error with most obvious fix
- Do random changes until it seems to work

Programmers do not always use available data to constrain their reasoning.

They carry out minor and irrational repairs, and they often don't undo the incorrect repairs.

— Iris Vessey

Scientific approach

- Observe through repeatable experiments
- Form hypothesis that is consistent with observations
- Make predictions based on hypothesis
- Test predictions by experiments or further observations
- Repeat steps 3 and 4 until hypothesis and experiments/observation contradict

Scientific approach — in debugging

1. Observe a failure
   - stabilize error and make it occur reliably
- Invent hypothesis for failure cause consistent with observations
- Make predictions based on hypothesis
- Test hypothesis by experiments/observations

   a) if experiment satisfies prediction, refine hypothesis

   b) otherwise create alternate hypothesis
- Repeat 3 and 4 until hypothesis can no longer be refined

Scientific approach — example

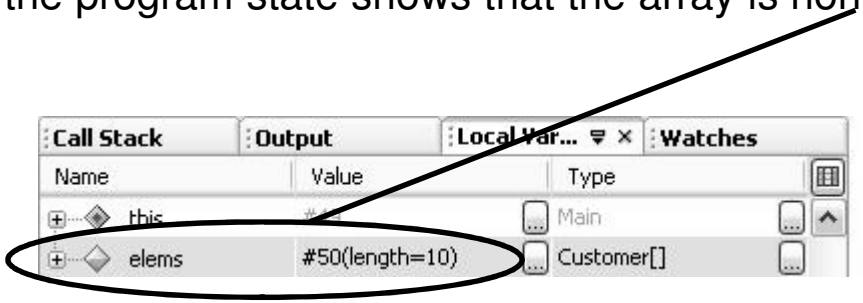
- **Failure:** Every time I run my application, it throws a NullPointerException in method initElements.

```
Exception in thread "main" java.lang.NullPointerException
        at Main.initElements(Main.java:18)
        at Main.main(Main.java:13)
```

```
void initElements(Customer elems[]) {
    for(int i=0; i < elems.length; i++) {
        elems[i].init();
    }
}
```

Scientific approach — example

- **Failure:** Every time I run my application, it throws a NullPointerException in method initElements.
- **Hypothesis:** array elems is not initialized.
- **Prediction:** at the beginning of method initElements, elems is null.
- **Prediction fails:** inspecting the program state shows that the array is non-null at beginning of initElements.




```
void initElements(Customer elems[]) {
    for(int i=0; i < elems.length; i++) {
        elems[i].init();
    }
}
```

Scientific approach — example

- **Failure:** Every time I run my application, it throws a NullPointerException in method initElements.
- **Hypothesis:** array elems is not initialized.
- **Prediction:** at the beginning of method initElements, elems is null.
- **Prediction fails:** inspecting the program state shows that the array is non-null at beginning of initElements.
- **New hypothesis:** the elements of the array are null when invoking method init on them.
- **New prediction:** at the point where method init is called, the target reference is null.

```
void initElements(Customer elems[]) {
    for(int i=0; i < elems.length; i++) {
        elems[i].init();
    }
}
```

Scientific approach — example

- **Failure:** Every time I run my application, it throws a NullPointerException in method initElements.
- **Hypothesis:** array elems is not initialized.
- **Prediction:** at the beginning of method initElements, elems is null.
- **Prediction fails:** inspecting the program state shows that the array is non-null at beginning of initElements.
- **New hypothesis:** the elements of the array are null when invoking method init on them.
- **New prediction:** at the point where method init is called, the target reference is null.
- **Prediction confirmed** by inspecting the program state.

Scientific approach — example

- **Failure:** Every time I run my application, it throws a NullPointerException in method initElements.
- **Hypothesis:** array elems that contains the elements is not initialized.
- **Prediction:** at the beginning of method initElements, elems is null.
- **Prediction fails:** inspecting the program state shows that the array is non-null at beginning of initElements.
- **New hypothesis:** the elements of the array are null when invoking method init on them.
- **New prediction:** at the point where method init is called, the target reference is null.
- **Prediction confirmed** by inspecting the program state.
- **Hypothesis need not be refined further**. Program can now be fixed, for instance, by making sure that elements of elems are non-null when method initElements is called.

Demo with NetBeans

Tips for finding defects

- Understand language & behavior of library methods
- Reproduce the error in several different ways
- Use all data available to make hypothesis
- Use negative test results too
- Narrow (and expand) the suspicious region of code
- Be suspicious of code that have had bugs before or has changed recently
- Integrate incrementally
- Take a break!

Tips for fixing defects

- Add test case that exposes the defect
- Confirm your hypothesis by test cases
- Understand problem before fixing it
- Understand program/module, not just problem
- Fix problem, not symptom
- Make one fix at a time
- Check your fix
- Look for similar defects

References

- **Andreas Zeller**: *Why Programs Fail: A Guide to Systematic Debugging,* Morgan Kaufmann*, 2005.*

- *Using NetBeans™ IDE*

- **Steve McConnell**: *Code Complete*, Microsoft Press, 2nd edition, Chapter 23.