
CSE 142

Testing, Debugging, and Program Design

2/4/2004

(c) 2001-4, University of Washington

K-1

Outline for Today

- How do we ensure that software works?
- Testing
- Debugging
 - toString methods (review)
- Main methods
- Some thoughts on design

2/4/2004

(c) 2001-4, University of Washington

K-2

Goals

- **Verify** that software works *correctly*
(whatever that means)
- **Diagnose and Fix** problems effectively
(figuring out a systematic way to approach this)
- **Design** software to increase the chances it works properly, and can be debugged and modified effectively and efficiently
(a hard problem)

2/4/2004

(c) 2001-4, University of Washington

K-3

What Does It Mean for Software to be "Correct"?

- Some possible definitions
 - Does what the programmer wrote in the code
 - Works as intended
 - Does what the end user/customer wants/expects
- What do you think is the right definition?

2/4/2004

(c) 2001-4, University of Washington

K-4

Classes

- The unit of programming in Java is the class
 - So, in Java, for code to be correct, it means the class implementation is correct
- What does it mean for an implementation to be correct?
- Informally, everything works, provided constructors and methods are used with suitable arguments
- More precisely,
 - A newly constructed object has an appropriate state
 - If given suitable arguments, each method works properly, returns the right result, and leaves the object in an appropriate (possibly updated) state

2/4/2004

(c) 2001-4, University of Washington

K-5

Preconditions, Postconditions, and Invariants

- Still more precisely
 - Invariant – a property that is always true
 - Class invariant – a property of the class – often about its state – that is always true
 - (except, possibly, momentarily while related things are being updated)
 - Precondition – a condition that must be true for the method to be able to execute correctly
 - Postcondition – a condition that is guaranteed to be true after a method has executed, provided its preconditions were satisfied when it was called

2/4/2004

(c) 2001-4, University of Washington

K-6

Class Invariant Example: CreditCard Class

```
/** Representation of a single credit card */
public class CreditCard {
    // instance variables
    private String name;      // account holder's name
    private int number;       // account number
    private double limit;     // credit limit, limit >= 0.0 always
    private double balance;   // current account balance;
                                // 0.0 <= balance <= limit always
    ...
}
```

- The constraints on limit and balance are examples of class invariants
- Class invariants are normally not directly apparent in the raw Java code, but they are needed to understand the class – so include them in comments

2/4/2004

(c) 2001-4, University of Washington

K-7

Postcondition Example: CreditCard Constructor

```
public class CreditCard {
    ...
    /** Construct a new CreditCard with the given name, account number, and
     * credit limit, with an initial balance of 0.0
     * Postcondition: class invariants are true with name = given name,
     * number = given number, limit = given limit, and balance = 0.0
     */
    public CreditCard(String name, int number, double limit) {
        this.name = name;
        this.number = number;
        this.limit = limit;
        this.balance = 0.0;
    }
    ...
}
```

- Footnote: specifying the postcondition in this much detail is normally overkill, since the comment describes the parameters, and correct constructors/methods can be assumed to preserve the class invariants (but it illustrates the point of what a postcondition is)

2/4/2004

(c) 2001-4, University of Washington

K-8

Precondition/Postcondition Example

```
public class CreditCard {
    ...
    /** Add amount to this credit card's balance, provided the limit is big enough.
     * @return true if successful, otherwise return false
     * precondition: amount >= 0.0
     * postcondition: if amount+balance <= limit, increase balance by amount
     *                and return true, otherwise do nothing and return false
     */
    public boolean charge(double amount) {
        if (balance + amount <= limit) {
            balance = balance + amount;
            return true;
        } else {
            return false;
        }
    }
}
```

2/4/2004

(c) 2001-4, University of Washington

K-9

What if the Precondition is not True?

- This can only happen for two reasons:
 - Client code uses inappropriate arguments
 - Bugs in the class implementation
- How do we react?
 - Really covered in CSE143. Preview....
 - Error in client code: generate an exception
(like NullPointerException, MethodNotFound, ...)
 - Bug: use assertions to catch problem during debugging

2/4/2004

(c) 2001-4, University of Washington

K-10

Testing

- Now we know how we want it to work, how do we decide if it is working?
- Goal – verify that the implementation is “correct”
- Procedure
 - Figure out what to test and what sample data to use
Do this before or while coding
 - Run tests and compare with expected results

2/4/2004

(c) 2001-4, University of Washington

K-11

Test Cases

- Can't test everything – way too many possible cases
- Try to test “important” cases
 - “Typical” cases
 - Edge cases – 0, 1, many
 - “Incorrect” cases – how does the code cope with bad data?
- Goal is to find a set of cases that covers all possibilities
 - Use representative data to cover each set of similar values

2/4/2004

(c) 2001-4, University of Washington

K-12

Example: Fahrenheit to Celsius

- Suppose we want to test code for the conversion
$$\text{celsius} = 5.0/9.0 * (\text{fahrenheit} - 32.0)$$
- Suggest some input values and expected output
 - Try to get complete coverage with as few cases as you can

2/4/2004

(c) 2001-4, University of Washington

K-13

Debugging – What If Something's Wrong?

- Effective debugging – a controlled experiment
 - Form hypothesis of what might be happening
 - Figure out how to gather information to verify or refute
 - Run experiments
 - Repeat until solved
- Goal is to systematically find bugs
 - What works?
 - Where do things go wrong?
 - What is happening? How can we fix it?
- Avoid random hacking – you'll just make things worse(!)

2/4/2004

(c) 2001-4, University of Washington

K-14

Gathering Debugging Information

- Simplest method: insert `"System.out.println(stuff);"` at interesting points
 - Figure out things you expect, then print out the actual values and compare
- Works great for basic types and objects (int, double, char, boolean, String)
- Would like to also be able to print objects to see important things about their state
 - `System.out.println(checking);`
 - Default Java prints memory address (mostly meaningless)
 - But – we can make our classes smarter so we get something useful when we print an object

2/4/2004

(c) 2001-4, University of Washington

K-15

Method toString

- Recall: a class can contain a `toString` method
- Whenever an object is used where a `String` is needed (in `println`, for example), the class's `toString` method (if present) is used to produce a suitable string
- `toString` specification (can use in any appropriate class)

```
/** Return a String representation of this object */  
public String toString() { ... }
```

2/4/2004

(c) 2001-4, University of Washington

K-16

toString Example

```
/** Return a string representation of this CreditCard */
public String toString() {
    String description = "CreditCard[name = " + name + ", number = " + number
        + ", balance = " + balance + ", limit = " + limit + "];"
    return description;
}
```

2/4/2004

(c) 2001-4, University of Washington

K-17

Running Tests

- So far we've run tests and programs by typing statements into DrJava's interactions window

```
CreditCard plastic = new CreditCard("I. M. Broke", 80195, 5000.00);
System.out.println(plastic);
plastic.charge(4950.00);
System.out.println(plastic);
```

- A test can be packaged so it can be run repeatedly
- How?

- Best: automated testing frameworks like JUnit
Something we'll cover in detail in CSE 143
- Alternatives: create methods containing the test code and execute them

2/4/2004

(c) 2001-4, University of Washington

K-18

Starting a Program

- DrJava's ability to create an instance of a class and use it directly is not typically available in all Java systems
- Standard Java convention for starting a program
 - Pick a class that is to control startup of the program
 - Provide a method named "main" in that class
 - Tell Java to start execution by identifying the class containing the main method we want to run
Command: `java ClassName`
- Execution begins by running the main method in that class
Typically, main creates a starting collection of objects & sends messages to them
- Can use this both to run the program, or to test things

2/4/2004

(c) 2001-4, University of Washington

K-19

Method main

- Any class can have a main method, but for simplicity we'll create a separate class for main

```
public class ClassName {
    /** main method - specify what it does */
    public static void main(String[] args) {
        code for main method goes here
    }
}
```

- "public static void main(String[] args)" has to be typed exactly like that
- "ClassName" is whatever you want (Main, Test1, TestChargeMethod, ...)
- "code for main method goes here" is the same kinds of statements you'd type in DrJava's interactions window or in another method

2/4/2004

(c) 2001-4, University of Washington

K-20

Example Test Program for CreditCard

```
/** Test CreditCard charge method */
public class Main {
    /** test program for charge */
    public static void main(String[] args) {
        CreditCard plastic = new CreditCard("I. M. Broke", 80195, 5000.00);
        System.out.println(plastic);
        plastic.charge(4950.00);
        System.out.println(plastic);
    }
}
```

- Once this is compiled in DrJava, you can run it by typing
java Main
in the interactions window
(use the name of the class instead of "Main" if it is different)

2/4/2004

(c) 2001-4, University of Washington

K-21

Designing Software for Quality

- There are many ways to divide the parts of a system into separate classes
- Key idea: Each class should do one thing well
- Key idea: **Information Hiding**
 - Implementation details in different classes should be independent of and hidden from each other
 - Use public interfaces and references to other objects – don't rely on private information that can change without notice
- Hard to get right – takes experience and redesign – but makes testing, debugging, and modification much easier when done well

2/4/2004

(c) 2001-4, University of Washington

K-22

Coupling and Cohesion

- Specific concepts to talk about design quality
 - Qualitative, hard to measure, but useful
- **Cohesion** – the degree to which a class *completely* encapsulates a *single* notion
 - Maximize this
 - If a class is doing more than one thing, split it into separate classes
- **Coupling** – the degree to which a class interacts with and depends on other classes
 - Minimize this

2/4/2004

(c) 2001-4, University of Washington

K-23

Summary

- Building quality software is not easy
 - Need good design to start
Coupling, cohesion
 - Need to check that things work as expected
Designing and implementing test cases
 - Need to effectively diagnose and fix any problems
Debugging
- Worth the effort to try to get these things right
 - Higher-quality software, built faster, tested and debugged with less grief, happier customers

2/4/2004

(c) 2001-4, University of Washington

K-24