



methodA sorts an array of Strings

- alphabetic order, using String *compareTo* method
 methodB sorts an array of BankAccounts
- order determined by comparing the balances:
 if (ba1[x].getBalance() <= ba2[y].getBalance()...

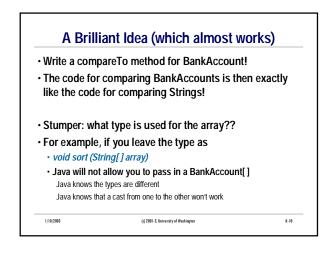
• The code for the two methods is largely the same • Parameter types are different

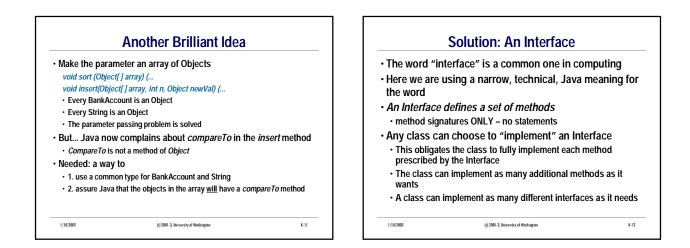
The only algorithmic difference:

1/10/2003

- How to tell if one thing is $<_{i} ==_{i}$ or > the other
- · Goal: write one method that works for both

(c) 2001-3, University of Washington





The Interface for our Example

- compareTo is the method that the sort method needs to call, so..
- 1. Define an Interface which specifies the compareTo method
- \bullet 2. Modify the sort method signature to show that the array must implement that Interface
- Make sure that both BankAccount and String both implement that Interface

All this requires is that each class implements a compareTo method

Problem solved!

1/10/2003

• When Java sees a call to *sort*, it can check that the objects in the array satisfy the Interface.

(c) 2001-3, University of Washington

<section-header><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item>

Interfaces: The Final Magic • We can declare objects of this type:

Comparable obj1;

1/10/2003

- means obj1 will refer to some object which implements the Comparable interface
- The magic: obj1 can refer either to a BankAccount or a String!
- More magic: Comparable[] can refer to either a BankAccount array or a String array!!
- Final form of the method signatures:
- public void sort(Comparable[] array)
- public void insert(Comparable[] array, int pos, Object newValue)
- Final magic: our code works now not just with String and BankAccount, but *any* objects that implement *Comparable*.

(c) 2001-3, University of Washington

K-15

K-13

Correctness and Specifications at the Java Level

- · The unit of programming in Java is the class
- What does it mean for a class implementation to be correct?
 Informally, "everything works", provided constructors and
- methods are used with suitable arguments
- More precisely,

1/10/2003

- 1. A newly constructed object has an appropriate state
- 2. If given suitable arguments, each method works properly, returns the right result, and leaves the object in an appropriate (possibly updated) state
- "Works properly" takes us back to the specification problem...

(c) 2001-3, University of Washington

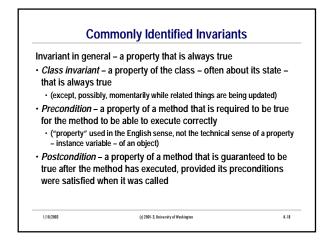


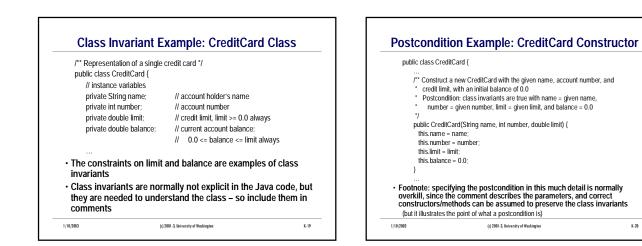
· Specifications are often given as comments in the code

- · Java programmers typically use JavaDoc conventions when writing major comments
- · Allows the comments to be extracted into a standard, widely understood format
- · A particular case of code specifications is especially important: the "invariant"
 - · Invariants are things which must be true if the program is correct
- · Invariants are sometimes described in comments, and sometimes incorporated into the code

(c) 2001-3, University of Washington

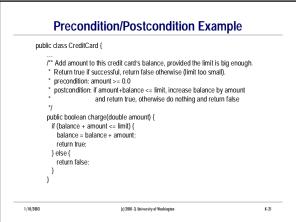
1/10/2003

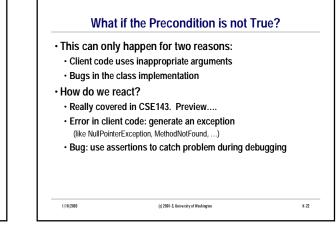




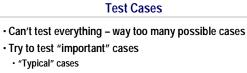
K-17

CSE142 Wi03





| Designing Methods | Testing | | |
|--|--|--|--|
| Invariants and Comments are valuable forms for method specifications | Now we know how we want it to work, how do we decide if it is working? | | |
| But who decides what methods and classes the system should have? | Goal – verify that the implementation is "correct" | | |
| Given a problem, there are usually many ways it can be divided into smaller parts such as methods and classes | Procedure Figure out what to test and what sample data to use | | |
| We focus here on method design: deciding which methods to define and how they fit together | Do this before or while coding Run tests and compare with expected results | | |
| Typical issues: | | | |
| One method or a number of smaller ones? | | | |
| What should the parameters and return values be? | | | |
| What instance variables are used and how? | | | |
| 1/10/2003 (d) 2001-3. University of Washington K-23 | | | |



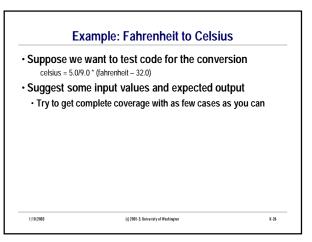
• Edge cases – 0, 1, many

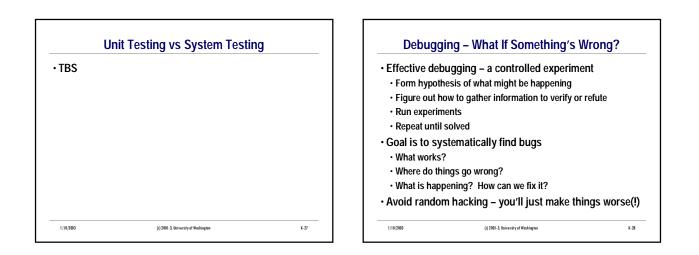
1/10/2003

- "Incorrect" cases how does the code cope with bad data?
- $\boldsymbol{\cdot}$ Goal is to find a set of cases that covers all possibilities

(c) 2001-3, University of Washington

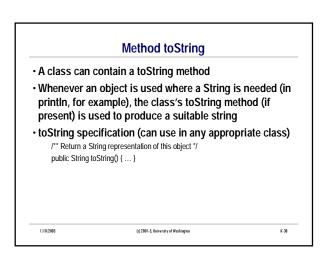
 $\boldsymbol{\cdot}$ Use representative data to cover each set of similar values

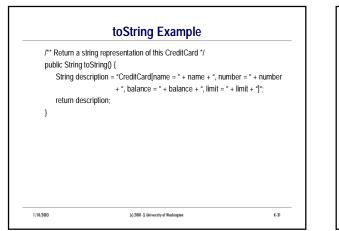


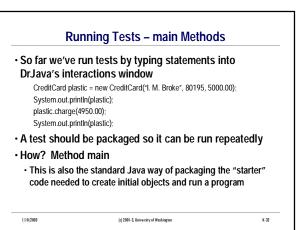


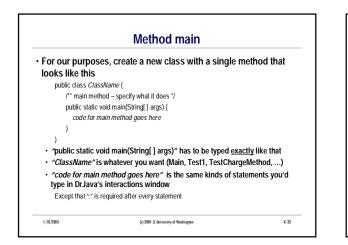


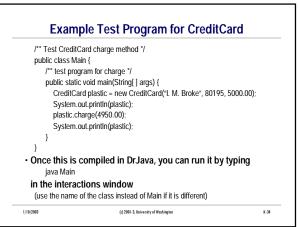
- Simplest method: insert "System.out.println(*stuff*);" at interesting points
- Figure out things you expect, then print out the actual values and compare
- Works great for basic types and objects (int, double, char, boolean, String)
- Would like to also be able to print objects to see important things 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

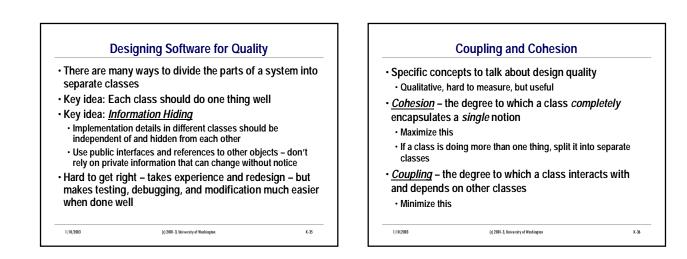












| Building qualit | ty software is not easy | |
|-------------------------------|---|---------------|
| Need good deal | sign to start | |
| Coupling, cohesi | on | |
| Reuseble parts p | romotes quality | |
| Need to check | that things work as expected | |
| Designing and in | plementing test cases | |
| Need to effecti Debugging | vely diagnose and fix any proble | ms |
| Worth the effo | rt to try to get these things ri | ght |
| • • • | software, built faster, tested and pier customers | debugged with |