Section 3!

Including, but not limited to, some or all of the following...

Subversion  
test coverage  
handling invalid input
Bookkeeping:
Stuff you should know

• Krysta can’t remember faces... 😞 (or names)
  o It’s VERY awkward. But it’s genetic.
  o I’m not being rude, just oblivious. I promise!
  o Yes, I want you to call me out if I forget we’ve met

• I also talk too fast... call me out on that too!

• Krysta’s office hours policy
  o I’m in the labs pretty often, working on my own stuff
  o You can ALWAYS ask me for help!
  o If it’s a bad time for me, I will say so – so don’t be afraid to ask
  o (But do try to work through things on your own first... it will make you a better programmer)
Bookkeeping: Stuff we want to know

• Piazza
  o Like? Dislike?

• Important announcements
  o Piazza OK? Mailing list? Both?

• Office hours
  o Would Thurs OH be useful?
  o (Good chance I’ll be there anyway after 1pm... see previous slide)
Version Control

(in which we build big software without losing our sanity)
Overview

• System for tracking changes to code
• Essential for managing big projects
  o Learn it now – you WILL use it again and again!
• Makes it easy to:
  o Merge multiple developers’ changes
  o Avoid overwriting each others’ changes
  o Revert back to an older version of a file
  o See a history of changes
  o Back up your work
  o …and more!
• You’ll use Subversion (SVN) this quarter
  o There are others: Mercurial, Git, CVS, …
Organization

- A repository stores the master copy of the project
  - Someone creates the repo for a new project
  - Then nobody touches this copy directly
  - Lives on a server everyone can access

- Each person checks out her own working copy
  - Makes a local copy of the repo
  - You’ll always work off of this copy
  - The version control system syncs the repo and working copy (with your help)
Common Actions

Everyday commands:

• **Commit / checkin**
  - integrate changes *from* your working copy *into* the repository

• **Update**
  - integrate changes *into* your working copy *from* the repository
Common Actions

Less frequent commands:

• **Add, delete**
  - add or delete a file in the repository

• **Revert**
  - wipe out your local changes to a file

• **Resolve, diff, merge**
  - Handle a **conflict** — two users editing the same code
Getting Started

• Multiple ways to use SVN
  o **Subclipse**: plugin for Eclipse
  o Can also use command-line, TortoiseSVN/NautilusSVN (GUI)

1. Create repository (command-line):
   Run the following on attu (Linux lab machine or SSH):
   ```
   $ svnadmin create /projects/instr/11sp/cse331/GROUPNAME
   $ chmod -R g+rw /projects/instr/11au/cse331/GROUPNAME
   ```
   to turn your shared group directory into a repository. Totally lost? *That’s OK!!* Email me to meet for a 5-minute intro to Linux.

1. Install Subclipse
   o Should already be installed in labs
   o See section handout and [http://www.cs.washington.edu/education/courses/cse331/11sp/groups.shtml](http://www.cs.washington.edu/education/courses/cse331/11sp/groups.shtml)

3. Create or checkout project
Using Subclipse

• “Team Synchronization” perspective
  o Can use to perform updates, commits, etc.
  o Eclipse will ask you if you want to use this, or go to Windows -> Open Perspective -> Other...
  o For most commands, right-click in “Synchronization” tab
  o Updates: may need to click “Synchronize SVN” button first

• Ordinary Java perspective
  o Team Sync view not great while you’re busy coding (Sync tab only shows certain files, etc.)
  o Restore “regular” perspective from Windows -> Open Perspective -> Other… -> Java (Default) or icons in top-right corner
  o In Package Explorer, right-click on your project and choose “Team” to do updates (“Update to HEAD”), commits, etc.
Using Subclipse

• “Team Synchronization” perspective
  o Use to perform updates, commits, etc.
  o Eclipse will ask you if you want to use this, or go to Windows -> Open Perspective -> Other…
  o For most commands, right-click in “Synchronization” tab
  o Updates: may need to click “Synchronize SVN” button first

• Ordinary Java perspective
  o Team Sync view not great while you’re busy coding (Sync tab only shows certain files, etc.)
  o Restore “regular” perspective from Windows -> Open Perspective -> Other… -> Java (Default) or icons in top-right corner
Using Subclipse

Demo!

By the way, http://svnbook.red-bean.com/ is a great resource for SVN
Handling Invalid Input

(a.k.a. expecting the unexpected)
Invalid Input (Callee)

• Reference: *Effective Java*, pg. 181
• Note: confusion around Assignment 1...
  o Replace anything I said earlier with what I’m saying now
  o Because of GUI design, you couldn’t always follow these recommendations
• **Assume nothing**: many reasons preconditions violated
  o Buggy code, malicious code, sloppy code
Invalid Input (Callee)

- **Fail early and often**: easier to locate bugs
- **Fail friendly**: make the caller’s job easier
  - Throw an exception, document with `@throws`
  - e.g. `IllegalArgumentException`, `IndexOutOfBoundsException`, `NullPointerException`
  - Don’t leave data structures or operations in intermediate states
- But remember: fancy input validation might be expensive
  - E.g. binary search: verifying that the list is sorted defeats the point of doing binary search
Invalid Input (Caller)

- Know what might cause unexpected values
  - User input
  - Data access: failure to open file, connect to database, etc. (null values?)
- Validate parameters before calling ...
  - User input especially!
- ... or be prepared to catch exceptions
  - Use a try...catch block
  - Are you sure the method validates input?
Test Coverage

(knowing what to test and when to stop)
Input Categories

• Classes of input that could be expected to cause different behavior
  o Negative integers, positive integers, zero
  o Reversing a string: odd, even length

• Run at least one test from each class

• Sometimes multiple ways to categorize

• Example: testing that Item.toString() prints two digits after decimal point
Input Categories

Example: testing that `Item.toString()` prints two digits after decimal point: what if...

- Price is an integer? ($10.00)
- Price has one digit after decimal point? ($10.50)
- Price has two digits after decimal point? ($10.99)
- Price has 3+ digits after decimal point? ($10.895)
- Price is negative? Zero? Positive?
- Price has zero/one/two digits before decimal? ($0.05, $1.05, $10.05)
Boundary Conditions

• Values on the edges between input categories
• Example: ShoppingCart discounts total if cart contains at least $q$ items
• What if cart contains exactly $q$ items? $q-1$ items? $q+1$?
• (Not really a boundary condition, but... what if cart contains $q$ items and then one is removed?)
Edge Cases

• The uncommon case: extreme or unexpected values
• Empty/null/zeros: search an empty list, reverse an empty string
• Ones: search a one-element list, one-element string
• Minima/maxima
• Unusual patterns
• Sorting algorithm: list already sorted, reverse-sorted
• Strings: non-alphabetic characters? non-ASCII characters?
Invalid Input

- What should happen with invalid input?
- Make sure the program doesn’t crash, at least
- JUnit: use
  @Test(expected=ExceptionName.class) to test that exception is thrown
Where to stop?

• You can never test all possible inputs
• With each new test, ask: “What is this testing that has not been covered in a previous test?”
  o A different input category?
  o A boundary condition?
  o An edge case?
SDET Test Buckets

- DON’T need to know for this course
- DO need to know for job interviews (SDET, also SDE)

Test buckets:
- Input categories
- Boundary conditions
- Edge cases
- Internationalization
- Accessibility
- Security
- Performance
- Stress/load testing
- Possibly more...