Section 5: HW6 and Interfaces

Slides adapted from Alex Mariakakis,
with material from Krysta Yousoufian, Mike Ernst, Kellen Donohue
Agenda

- Version control and tools review
- BFS
- Interfaces
- Parsing Marvel Data
331 Version control

- create
- check out
- check out
- check out
- update
- commit
- add

Working copy for grading

Repository

Working copy
Where is my code?

(personal computer)

update

(check out)

(commit)

(update)

(check out)

(repo)

(validate)

(../scratch from attu working copy)

(attu scratch copy)

(attu working copy)

(/homes/iws/CSENETID/, or other directory)

(/projects/instr/14au/cse331/YourCSENetID/REPOS/cse331)
Where is my code?

• **Main repo**: /projects/instr/etc
  - Not human readable
  - You can’t see files here

• **Personal computer**: any directory, via Subclipse or other
  - Working copy: add and edit files here
  - Must check in files for them to go to the repo

• **attu working copy**: /homes/iws/CSENETID/ or other
  - Just another working copy, same as personal computer
  - Must svn update to see changes from pc/repo

• **validate copy**: attu directory/src/…/scratch
  - NEW WORKING COPY CHECKED OUT FROM REPO
  - May NOT be the same as attu working copy if attu wasn’t updated
Concepts vs tools, 331 vs general

- Version control: concept
  - Tools: svn, TortoiseSVN, Subclipse
- Ant: tool
  - Concept: build management
  - validate 331
- Remote access: concept
  - Tools: ssh, PuTTY, WinSCP
- Javadoc: tool
  - Concept: documentation
  - @param, @return, @throws general
  - @requires, @modifies, @effects 331
Can I reach B from A?
Breadth-First Search (BFS)

- Often used for discovering connectivity
- Calculates the shortest path if and only if all edges have same positive or no weight
- Depth-first search (DFS) is commonly mentioned with BFS
  - BFS looks “wide”, DFS looks “deep”
  - Can also be used for discovery, but not the shortest path
public boolean find(Node start, Node end) {
    put start node in a queue
    while (queue is not empty) {
        pop node N off queue
        if (N is goal)
            return true;
        else {
            for each node O that is child of N
                push O onto queue
        }
    }
    return false;
}
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <B>
Q: <B, C>
DONE
Breadth-First Search with Cycle

Q: <>
Q: <A>
Q: <>
Q: <B>
Q: <>
Q: <C>
Q: <>
Q: <>
NEVER
DONE
BFS Pseudocode

public boolean find(Node start, Node end) {
    put start node in a queue
    while (queue is not empty) {
        pop node N off queue
        if (N is goal)
            return true;
        else {
            for each node O that is child of N
                push O onto queue
        }
    }
    return false;
}

Mark the node as visited!

What if there’s a cycle?
What if there’s no path between start and end?
Breadth-First Search

Q: <>
Breadth-First Search

Q: <>
Q: <A>
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C,D>
Q: <D>
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C, D>
Q: <D>
Q: <D, E>
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C, D>
Q: <D>
Q: <D, E>
Q: <E>
Breadth-First Search

Q: <->
Q: <A>
Q: <->
Q: <C>
Q: <C, D>
Q: <D>
Q: <D, E>
Q: <E>
DONE
Shortest Paths with BFS

<table>
<thead>
<tr>
<th>Destination</th>
<th>Path</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>&lt;B,A&gt;</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>&lt;B&gt;</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>&lt;B,A,C&gt;</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>&lt;B,D&gt;</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>&lt;B,D,E&gt;</td>
<td>2</td>
</tr>
</tbody>
</table>

From Node B
Shortest Paths with Weights

Paths are not the same!
Classes, Interfaces, and Types

- The fundamental unit of programming in Java is a class
- Classes can extend other classes and implement interfaces
- Interfaces can extend other interfaces
Classes, Objects, and Java

- Everything is an instance of a class
  - Defines data and methods
- Every class extends exactly one other class
  - Object if no explicit superclass
  - Inherits superclass fields
- Every class also defines a type
  - Foo defines type Foo
  - Foo inherits all inherited types
- Java classes contain both specification and implementation!
Interfaces

• Pure type declaration

```java
public interface Comparable {
    int compareTo(Object other);
}
```

• Can contain:
  • Method specifications (implicitly `public abstract`)
  • Named constants (implicitly `public final static`)

• Does not contain implementation

• Cannot create instances of interfaces
Implementing Interfaces

• A class can implement one or more interfaces
  
  class Kitten implements Pettable, Huggable

• The implementing class and its instances have the interface type(s) as well as the class type(s)

• The class must provide or inherit an implementation of all methods defined by the interface(s)
  • Not true for abstract classes
Using Interface Types

• An interface defines a type, so we can declare variables and parameters of that type
• A variable with an interface type can refer to an object of any class implementing that type

```java
List<String> x = new ArrayList<String>();
void sort(List myList) {...}
```
Guidelines for Interfaces

• Provide interfaces for significant types and abstractions
• Write code using interface types like Map instead of HashMap and TreeMap wherever possible
  • Allows code to work with different implementations later on
• Both interfaces and classes are appropriate in various circumstances
Demo
Parsing the Marvel data