Section 5: HW6 and Interfaces

SLIDES ADAPTED FROM ALEX MARIAKAKIS,
WITH MATERIAL FROM KRYSTA YOUSOUFIAN, MIKE ERNST, KELLEN DONOHUE
How is Homework 5 going?
Agenda

* Breadth-first search (BFS)
* Interfaces
* Parsing Marvel Data
Reminder: Enabling asserts in Eclipse

To enable asserts:
Go to Run -> Run Configurations… -> Arguments tab -> input -ea in VM arguments section
Reminders:

Expensive CheckReps are **BAD**
(at least when assignments are turned in, but can be useful for finding hard-to-discover problems – so need to be able to control expensive checks)

Debug flags are **GOOD**
(or enums to indicate depth of debug)
Don’t forget your CheckReps!
Graphs

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”
  - DFS can also be used for discovery, but not the shortest path
BFS Pseudocode

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

START:
Q: <A>
Pop: A, Q: <>
Q: <B, C>
Pop: B, Q: <C>
Q: <C>
Pop: C, Q: <C>
Q: <>
DONE

Starting at A
Goal: Fully explore
Breadth-First Search with Cycle

START:
Q: <A>
Pop: A, Q: <>
Q: <B>
Pop: B, Q: <>
Q: <C>
Pop: C, Q: <>
Q: <A>
NEVER DONE

Starting at A
Goal: Fully Explore
BFS Pseudocode

```pseudocode
put start node in a queue
while (queue is not empty):
    pop node N off queue
    mark node N as visited
    if (N is goal):
        return true
    else:
        for each node O that is child of N:
            if O is not marked visited:
                push O onto queue
return false
```
Breadth-First Search

Problem: Find everything reachable from A

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

From Node B

<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></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shortest path to D? to E? What are the costs?
Shortest Paths with BFS

From Node B

<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>

Shortest path to D? to E? What are the costs?
Shortest Paths with Weights

Weights are not the same! Are the paths?
Shortest Paths with Weights

From Node B

<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>2</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>5</td>
</tr>
<tr>
<td>D</td>
<td>&lt;B,A,C,D&gt;</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>&lt;B,A,C,E&gt;</td>
<td>7</td>
</tr>
</tbody>
</table>
Interfaces
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
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 aList) {...}
```
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