University of British Columbia
CPSC 111, Intro to Computation
Jan-Apr 2006

Tamara Munzner

Interfaces, Polymorphism II

Lecture 21, Thu Mar 23 2006

based on slides by Kurt Eiselt and Paul Carter

http://www.cs.ubc.ca/~tmm/courses/cpsc111-06-spr

News

- labs this week
  - midterms returned
  - work through what you got wrong on midterm
  - **can earn back up to 5 out of 70 points**
  - if you don't finish during your normal lab, can show TAs your work next week or at other labs this week
- Assignment 2 handed back at end of class
  - most, but not all
- Assignment 3 posted
  - due Friday Apr 7, 5pm

Recap: Method Overloading

- Can have multiple methods of same name
- Distinguishes between them with signature
  - method name, parameter types and order
- Cannot have two methods with same signature
- Return type is not part of signature

- Any method can be overloaded
  - constructors are very common case

Recap: Interfaces

- Interface is collection of constants and abstract methods
  - different meaning than set of public methods that are documented, as in API
  - to implement interface must provide definitions for all its methods
- Abstract methods have no implementation or body
  - method header followed by semicolon
  - specifies how to communicate with method, not what it does

Recap: Interface Example

```
public interface VendingMachine
{

  public void vendItem();

  public int getItemsRemaining();

  public int getItemsSold();

  public double getCashReceived();

  public void loadItems(int n);

}
```

```
public class CokeMachine2005 implements VendingMachine
{
```

Recap: Interface Syntax

- Use reserved word **`interface`** instead of **`class`** in header
  - no need to use reserved word abstract in method headers, is automatic with interfaces

- Use reserved word **`implements`** followed by interface name in class header

Recap: Polymorphism

- Polymorphism: behavior varies depending on actual type of object
  - variables can be declared with interface as type, can invoke interface methods on them
  - cannot construct interface
    - can only construct objects of some particular class that implements interface
- Polymorphism determined at runtime
  - vs. method overloading, determined at compilation

Recap: Polymorphism Example

```
public class SimCoke2005
{
  public static void main (String[] args)
  {
    VendingMachine foo1 = new CokeMachine2005();
    VendingMachine foo2 = new FrenchFryMachine2005();

    foo1.vendItem();
    foo2.vendItem();
  }
}
```

```
Adding another CokeMachine to your empire
Adding another FrenchFryMachine to your empire
Have a Coke
9 cans remaining
Have a nice hot cup of french fries
9 cups of french fries remaining
```

Recap: Bunny Example

```
public interface Bunnies
{
  public void moveBunny(int direction);

}
```

```
public class BigBunny implements Bunnies {

public void moveBunny(int direction) {
    if (direction == 12) {
      y = y + 3;
      carrots = carrots - 2;
    } ...
}
```

```
public class LittleBunny implements Bunnies {

public void moveBunny(int direction) {
    if (direction == 12) {
      y = y + 1;
      carrots = carrots - 1;
    } ...
}
```

Polymorphism

- reference to interface type can reference instance of *any* class implementing that interface
  - static type: type that variable declared to be
    - determines which members of class can be invoked
  - dynamic type: type that variable actually references
    - determines which version of method is called

Interfaces as Contract

- Can write code that works on anything that fulfills contract
  - even classes that don't exist yet!
- Example: Comparable
  - useful if you need to sort items
  - **`compareTo(object)`**
    - returns -1 if this object less than object o
    - returns 0 if same
    - returns 1 if this object greater than parameter

Comparable

- sort method that works on array of objects of any type that implements **`Comparable`**
  - type guaranteed to have **`compareTo`** method

- we need to sort
  - **`Bunny`**
  - **`Giraffe`**
  - **`String`**
  - **`...`**

Selection Sort For Int Primitives

```
// selection sort
public class SortTest1
{
  public static void main(String[] args)
  {
    int[] numbers = {16,3,19,8,12};
    int min, temp;
    //select location of next sorted value
    for (int i = 0; i < numbers.length-1; i++)
    {
      min = i;
      //find the smallest value in the remainder of
      //the array to be sorted
      for (int j = i+1; j < numbers.length; j++)
      {
        if (numbers[j] < numbers[min])
        {
          min = j;
        }
      }
      //swap two values in the array
      temp = numbers[i];
      numbers[i] = numbers[min];
      numbers[min] = temp;
    }

    System.out.println("Printing sorted result");
    for (int i = 0; i < numbers.length; i++)
    {
      System.out.println(numbers[i]);
    }
  }
}
```

Wrappers

- Many classes implement Comparable interface
  - Byte, Character, Double, Float, Integer, Long, Short, String
  - each implements own version of compareTo
- Wrapper classes
  - wraps up (encapsulates) primitive type
  - Double: object wrapping primitive double
    - No: `sort( double[] myData );`
    - Yes: `sort( Double[] myData );`

Multiple Interfaces

- Classes can implement more than one interface at once
  - contract to implement all abstract methods defined in every interface it implements

```
public class MyClass implements Interface1, Interface2,
   Interface3
{
}
```