Inheritance II

Lecture 23, Thu Mar 30 2006

based on slides by Kurt Eiselt

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

- Check your lab 7 grade
  - we haven’t yet handed out midterm solution, but the window will close soon!
  - 5/70 midterm points is 1% of your course grade!
- Yet a few more (but not all) Assignment 2s to hand back after class
- Assignment 3 due Friday Apr 7, 5pm
  - start now now now!
- Final exam: Mon Apr 24, 3:30pm, HEBB TH
- Evaluations today (beginning of class)
Recap: Comparable

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

- sorted
  - int
  - String
  - Bunny

- revisit Bunny.compareTo: checking dynamic type of object
Recap: Multiple Interfaces

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

```java
public class MyClass implements Interface1, Interface2, Interface3 {
    {
}
```
Recap: Inheritance

- Inheritance: process by which new class is derived from existing one
  - fundamental principle of object-oriented programming
- Create new child class (subclass) that extends existing parent one (superclass)
  - inherits all methods and variables
    - except constructor
  - can just add new variables and methods
Recap: Inheritance and Constructors

Subclass (child class) inherits all methods except constructor methods from superclass (parent class)

Using reserved word super in subclass constructor tells Java to call appropriate constructor method of superclass
Recap: Inheritance and Scope

- Subclasses inherits but cannot directly access private fields or variables of superclass
- Protected variables can be directly accessed from declaring class and any classes derived from it
Recap: Method Overriding

- If child class defines method with same name and signature as method in parent class
  - say child's version **overrides** parent's version in favor of its own
Recap: Object Behind the Scenes

- All classes that aren't explicitly extended from a named class are by default extended from \texttt{Object} class
  - \texttt{Object} class includes a \texttt{toString()} method
- so... class header
  
  \texttt{public class myClass}

- is actually same as
  
  \texttt{public class myClass extends Object}
Recap: Overriding Variables

- You can, but you shouldn't

- Possible for child class to declare variable with same name as variable inherited from parent class
  - one in child class is called shadow variable
  - confuses everyone!

- Child class already can gain access to inherited variable with same name
  - there's no good reason to declare new variable with the same name
Recap: Method Overloading and Overriding

- Method overloading: "easy" polymorphism
  - in any class can use same name for several different (but hopefully related) methods
  - methods must have different signatures so that compiler can tell which one is intended

- Method overriding: "complicated“ polymorphism
  - subclass has method with same signature as a method in the superclass
  - method in derived class overrides method in superclass
  - resolved at execution time, not compilation time
    - some call it true polymorphism
Objectives

- Understanding when and how to use abstract classes
- Understanding tradeoffs between interfaces and inheritance
A New Wrinkle

- Expand vending machine empire to include French fry machines
- Is a French fry machine a subclass of Coke Machine?
If We Have This Class Hierarchy...
...Does This Make Sense?

Coke Machine

French Fry Machine

Coke Machine2000

Coke MachineUA
Does This Make More Sense?

Generic Vending Machine

French Fry Machine

Coke Machine

Coke Machine2000

Coke MachineUA
Does This Make More Sense?

- Yes
  - especially if we're thinking of adding all kinds of vending machines...
Does This Make More Sense?

- Yes
  - especially if we're thinking of adding all kinds of vending machines...
  - want our classes to be more specific as we go down class hierarchy
    - is French Fry Machine more or less specific than Coke Machine?
    - neither, both specific versions of generic Vending Machine class
Does This Make More Sense?

- One way: make a VendingMachine interface like last week
- Another way...
Inheritance Solution

```java
public class GenericVendingMachine {
    private int numberOfItems;
    private double cashIn;

    public GenericVendingMachine() {
        numberOfItems = 0;
    }

    public boolean vendItem() {
        boolean result;
        if (numberOfItems > 0) {
            numberOfItems--;
            result = true;
        } else {
            result = false;
        }
        return result;
    }
}
```
Inheritance Solution

```java
public void loadItems(int n)
{
    numberOfItems = n;
}

public int getNumberOfItems()
{
    return numberOfItems;
}
```
public class CokeMachine3 extends GenericVendingMachine
{
    public CokeMachine3()
    {
        super();
    }

    public CokeMachine3(int n)
    {
        super();
        this.loadItems(n);
    }

    public void buyCoke()
    {
        if (this.vendItem())
        {
            System.out.println("Have a nice frosty Coca-Cola!"棚);
            System.out.println(this.getNumberofItems() + " cans of Coke remaining"棚);
        }
        else
        {
            System.out.println("Sorry, sold out"棚);
        }
    }
}

Inheritance Solution

```java
public void loadCoke(int n) {
    this.loadItems(this.getNumberofItems() + n);
    System.out.println("Adding " + n + " ice cold cans of Coke to this machine");
}
```
public class CokeMachine2000 extends CokeMachine3
{
    public CokeMachine2000()
    {
        super();
    }

    public CokeMachine2000(int n)
    {
        super();
        this.loadItems(n);
    }

    public void loadCoke(int n)
    {
        super.loadCoke(n);
        System.out.println("Loading in the new millennium!");
    }
}
Inheritance From Generic Objects

- Want generic VendingMachine class
  - don’t actually use to generate objects
  - use as template for specific actual classes like FrenchFryMachine and CokeMachine
Inheritance From Generic Objects

- Will we ever want to instantiate a generic Vending Machine class?
- Will we ever need to make generic Vending Machine object?
Inheritance From Generic Objects

- Will we ever want to instantiate a generic Vending Machine class?
  - Will we ever need to make generic Vending Machine object?
  - No, not in our simulated vending world!
  - How would we use one? What would be a real-life equivalent?
Inheritance From Generic Objects

- Introduced CokeMachineUA to combat vandalism and theft
- Could just add vandalize() methods to CM, CM2000, CMUA
  - but we want to ensure that all Vending Machines have vandalize() methods
  - want all of them to be different
    - if put into base class at top, easy to have them identical
    - no way to force method overriding
Abstract Classes

- **Abstract class**: not completely implemented
- Usually contains one or more **abstract methods**
  - has no definition: specifies method that should be implemented by subclasses
  - just has header, does not provide actual implementation for that method
- Abstract class uses abstract methods to specify what interface to descendant classes must look like
  - without providing implementation details for methods that make up interface
- Example: require that all subclasses of VendingMachine class implement `vandalize()` method
  - method might differ greatly between one subclass and another
  - use an abstract method
Abstract Classes

- Abstract classes serve as place holders in class hierarchy
- Abstract class typically used as partial description inherited by all its descendants
- Description insufficient to be useful by itself
  - cannot instantiated if defined properly
- Descendent classes supply additional information so that instantiation is meaningful
  - abstract class is generic concept in class hierarchy
  - class becomes abstract by including the `abstract` modifier in class header
Abstract Classes

- Use abstract class for generic template
  - can use abstract methods

- Making abstract method
  - Use restricted word `abstract` in method header
  - do not provide a method body
  - just end method header with semicolon
Vending Machine Class Revisited

```java
public abstract class VendingMachine {
    private int numberOfItems;

    public VendingMachine() {
        numberOfItems = 0;
    }

    public boolean vend() {
        boolean result;
        if (numberOfItems > 0) {
            numberOfItems--;
            result = true;
        } else {
            result = false;
        }
        return result;
    }

    public abstract void vandalize();
}
```
Abstract Methods and Abstract Classes

- What happens when we try to compile it all now?
  - Java tells us that there's an abstract class we have to implement
Abstract Methods and Abstract Classes

- What happens when we try to compile it all now?
  - Java tells us that there's an abstract class we have to implement
  - Could put this CokeMachine class:

```java
public void vandalize()
{
    System.out.println("Take all my money, and have a Coke too");
}
```
Abstract Methods and Abstract Classes

What happens when we try to compile it all now?
- Java tells us that there's an abstract class we have to implement

Could put this CokeMachine class:

```java
public void vandalize()
{
    System.out.println("Take all my money, and have a Coke too");
}
```

Do we have to implement method in CokeMachine2000 and CokeMachineUA classes too?
- Yes, if we want them to behave differently when they're vandalized
  - original intent
Which Organization?

- Beer Machine
- French Fry Machine
- Coke Machine
- Pizza Machine
- Coke Machine2000
- Coke MachineUA

Generic Vending Machine

- implements Beer Machine
- implements French Fry Machine
- implements Coke Machine
- implements Pizza Machine
- extends Coke Machine2000
- extends Coke MachineUA
Which Organization?

Generic Vending Machine

- Beer Machine
- French Fry Machine
- Coke Machine
- Pizza Machine

- Coke Machine2000
- Coke MachineUA
Interfaces vs. Abstract Classes

- If we can have abstract class that contains only abstract methods, why do we need interfaces?
Interfaces vs. Abstract Classes

- If we can have abstract class that contains only abstract methods, why do we need interfaces?
  - Java does not support multiple inheritance: child classes inheriting attributes from multiple parent classes
    - other object-oriented languages do
  - multiple inheritance can be good, but causes problems
    - what if child class inherits two different methods with same signature from two different parents?
      - which one should be used?
Interfaces vs. Abstract Classes

- Java's formal interface provides some of the utility of multiple inheritance without the problems
  - class can implement more than one interface
  - can do this at same time it extends class
- Interface allows us to create classes that "inherit" features from multiple places
Interfaces vs. Abstract Classes

- Java's formal interface provides some of the utility of multiple inheritance without the problems
  - class can implement more than one interface
  - can do this at same time it extends class
- Interface allows us to create classes that "inherit" features from multiple places
- Why is problem from previous slide solved?
  - might have multiple method headers with same signature
Interfaces vs. Abstract Classes

- Java's formal interface provides some of the utility of multiple inheritance without the problems
  - class can implement more than one interface
  - can do this at same time it extends class
- Interface allows us to create classes that "inherit" features from multiple places
- Why is problem from previous slide solved?
  - might have multiple method headers with same signature
  - but only one will have an actual definition
    - no ambiguity on which will be used
    - but still could be problem with different return types
Another useful feature provided by interfaces:
- inheritance happens between classes that are related
- But classes can implement completely unrelated interfaces
  - and that can be useful
Interfaces vs. Abstract Classes

- Another useful feature provided by interfaces:
  - inheritance happens between classes that are related
  - But classes can implement completely unrelated interfaces
    - and that can be useful

- Example: implement interfaces for
  - computer, printer, cell phone, vending machine
  - create class for new interactive vending machines that:
    - vend Cokes, show annoying music videos, phone their owner when they're running low on product, and spit out coupons for free prizes
How Interfaces Differ From Abstract Classes

- Abstract class is incomplete class that requires further specialization
  - interface is just specification or prescription for behavior

from *Just Java 2* by Peter van der Linden
How Interfaces Differ From Abstract Classes

- Abstract class is incomplete class that requires further specialization
  - interface is just specification or prescription for behavior
- Inheritance implies specialization, interface does not
  - interface just implies "We need something that does 'foo' and here are ways that users should be able to call it."

from Just Java 2 by Peter van der Linden
How Interfaces Differ From Abstract Classes

- Abstract class is incomplete class that requires further specialization
  - interface is just specification or prescription for behavior
- Inheritance implies specialization, interface does not
  - interface just implies "We need something that does 'foo' and here are ways that users should be able to call it."
- Class can implement several interfaces at once
  - but class can extend only one parent class

from Just Java 2 by Peter van der Linden
Interfaces vs. Abstract Classes: Bottom Line

- Use abstract class to initiate a hierarchy of more specialized classes
- Use interface to say, "I need to be able to call methods with these signatures in your class."
- Use an interface for some semblance of multiple inheritance

from *Just Java 2* by Peter van der Linden
Interfaces vs. Abstract Classes

- Interface can only extend another interface
  - cannot extend abstract class or "concrete" class
- Class can legally implement only some methods of interface if it’s abstract class
  - then must be further extended through inheritance before can be instantiated

from Just Java 2 by Peter van der Linden
Who Can Do What?

- Interface can be implemented only by class or abstract class
- Interface can be extended only by another interface
- Class can be extended only by class or abstract class
- Abstract class can be extended only by class or abstract class
- Only classes can be instantiated as objects
  - Interfaces are not classes and cannot be instantiated
  - Abstract classes may have undefined methods and cannot be instantiated