Recap: Inheritance

- Inheritance: process by which a new class is derived from an existing one
- Fundamental principle of object-oriented programming
- Create new class (subclass) that extends existing parent one (superclass)
- Inherits all methods and variables except constructor
- Can add new variables and methods

Some Coke Machine History

- Early Coke Machine
  - Mechanical
  - Sealed unit, must be reloaded at factory
  - Little protection against vandalism

Coke Machine 2000

- Electro-mechanical
- Can be reloaded on site
- Little protection against vandalism

Coke Machine UA

- Prototype cyberhuman intelligent mobile autonomous vending machine
- Can reload itself in transit
- Vandalism? Bring it on

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
### Method Overriding

**public class CokeMachineUA extends CokeMachine2000**

```java
public CokeMachineUA() {
    super();
}
```

**Public static method**

```
public static void main(String[] args) {
    CokeMachineUA x = new CokeMachineUA();
    x.vandalize();
    System.out.println("Eat lead and die, you slimy Pepsi drinker!!");
}
```

### Does This Make More Sense?

**GenericVendingMachine**

**CokeMachine**

**CokeMachine2000**

**French FryMachine**

**PizzaMachine**

**BeerMachine**

**SimVend**

**VendingMachine2000**

**mymachines**

**Object**

**myClass**

**View**

**Variables**

- **You can, but you shouldn't**
- **Possible for child class to declare variable with same name as variable inherited from parent class**
- **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**

### 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?**

### Another View of Polymorphism

- **From Just Java 2 by Peter van der Linden:**
  - **Polymorphism is a complicated name for a straightforward concept.**
  - **It merely means using the same one name to refer to different methods.**
  - **"Name reuse" would be a better term.**

- **Polymorphism made possible in Java through method overloading and method overriding**
- **Remember method overloading?**

### Method Overloading and Overriding

- **Where was it inherited from?**
- **All classes that aren't explicitly extended from a named class are by default extended from Object class**
- **Object class includes a toString() method**
- **So... class header**
  ```java
  public class myClass
  ```
  - **is actually same as**
  ```java
  public class myClass extends Object
  ```

### Does This Make More Sense?

**Yes**

- **Especially if we're thinking of adding all kinds of vending machines...**

### 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 boolean vend() {
    if (this.numberOfItems > 0) {
      this.numberOfItems--;
      return true;
    } else {
      return false;
    }
  }
}
```

Vending Machine Class Revisited

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

  public abstract boolean vend();
}
```

Inheritance Solution

```java
public class CokeMachine2000 extends CokeMachine3 {
  public CokeMachine2000() {
    super();
  }

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

Inheritance From Generic Objects

```java
public class CokeMachine3 extends GenericVendingMachine {
  public CokeMachine3() {
    super();
  }

  public void loadItems() {
    if (numberOfItems > 0) {
      numberOfItems--;
      result = true;
    } else {
      result = false;
    }
    return result;
  }
}
```

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
vendable() method
  * method might differ greatly between one subclass and another
  * use an abstract method

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 class CokeMachine {
      public abstract void vend();
    }
    ```
  * 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
```
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?
- 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.

Abstract class is complete 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.

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