Lecture 8: Interfaces and Abstract Classes

CSC 1214: Object-Oriented Programming

Outline

- Interfaces
- Abstract classes
- Abstract classes and Inheritance
- Interfaces and Inheritance

Outline

- Interfaces
- Abstract classes
- Abstract classes and Inheritance
- Interfaces and Inheritance

Interfaces: Introduction

- A Java interface is a collection of *abstract methods* and constants
- An *abstract method* is a method header without a method body/implementation
- An abstract method can be declared using the modifier **abstract**, but because all methods in an interface are abstract, usually it is left out
- An interface is used to establish a set of methods that a class will implement

Interfaces: Introduction

- An interface cannot be instantiated
- Methods in an interface have public visibility by default
- A class formally implements an interface by:
  - stating so in the class header
  - providing implementations for each abstract method in the interface
- If a class asserts that it implements an interface, it must define all methods in the interface

Interfaces: Example in Java

`interface` is a reserved word

```
public interface Animal
{
    public void makeSound();
    public void move();
}
```

None of the methods in an interface are given a definition (body)

A semicolon immediately follows each method header

Interfaces: Example in Java

`interface` is a reserved word

```
public interface EmailServer
{
   public final String SERVER_REF = “2013.001”
   public boolean ReceiveMail(String to, String from, String subject, String body);
   public String getHostName();
}
```

None of the methods in an interface are given a definition (body)

A semicolon immediately follows each method header

Interfaces: Example in Java

`interface` is a reserved word

```
public interface MobileMoney
{
   public boolean ReceiveMoney(String to, String from, float amount);
}
```

None of the methods in an interface are given a definition (body)

A semicolon immediately follows each method header

Implementing Interfaces

- A class uses the **implements** keyword to implement an interface.
- When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface.
- If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

Implementing Interfaces

```
public class Cat implements Animal
{
   public void makeSound ()
   {
      System.out.println("Meow");
   }
   public void move ()
   {
      // implementation
   }


   // Any additional methods.
}
```

`implements` is a reserved word

Each method listed in `Animal` class is given a definition

A class that implements an interface can implement other methods as well

Implementing Interfaces

```
public class Cat implements Animal
{
    public void makeSound ()
    {
        System.out.prin
    }
    public void move (
    {
        // implementati
    }


    // Any additional 
}
```

```
public class Dog implements Animal
{
    public void makeSound ()
    {
        System.out.println(“Woof”);
    }
    public void move ()
    {
        // implementation
    }


    // Any additional methods.
}
```

Implementing Interfaces

```
public class YahoomailServer implements EmailServer
{
 public boolean ReceiveMail(String to, String from, String subject, String body){

    // implementation
  }

 public String getHostName(){
    // implementation

  }
}
```

- In this example, different email servers can “talk” to each other by implementing the same **`EmailServer`** interface.

Implementing Interfaces

```
public class YahoomailServer implements EmailServer
{
```

```
public class GmailServer implements EmailServer
{
 public boolean ReceiveMail(String to, String from, String subject, String body){

    // implementation
 }

 public String getHostName(){
    // implementation


 }
}
```

- In addition to (or instead of) abstract methods, an interface can contain constants
- When a class implements an interface, it gains access to all its constants

Implementing Interfaces

```
public class MTNMobileMoney implements MobileMoney
{
   public boolean ReceiveMoney(String to, String from, float amount){

    // implementation
  }

    // Additional methods
 public void MTNBonus(){
    // implementation


  }
}
```

Implementing Interfaces

```
public class MTNMobileMoney implements MobileMoney
{
}
```

```
public class WaridPesa implements MobileMoney
{
 public boolean ReceiveMoney(String to, String from, float amount){

    // implementation
 }

 // Additional methods
 public void EnterPesaDraw(){
    // implementation

 }
}
```

- In this example, the **`MobileMoney`** interface enables implementation of a cross-platform mobile money system.

Implementing Interfaces

- A class can implement multiple interfaces
- The interfaces are listed in the implements clause
- The class must implement all methods in all interfaces listed in the header

```
class ManyThings implements interface1, interface2
{
   // All methods implementations.
}
```

A Note on Designing an Interface

Once agreed upon, this is a VERY costly thing to change, so make sure an interface is well thought out. Think long-term and make sure to only put things that REALLY should be required in there. ***For instance, imagine that an interface for email servers needs to be changed. This would imply modifying the implementation of different email servers across the globe.***

Outline

- Interfaces
- Abstract classes
- Abstract classes and Inheritance
- Interfaces and Inheritance

Abstract Classes & Inheritance

- An abstract class often contains abstract methods with no definitions (like an interface does).
- An abstract class may also contain non-abstract methods (with bodies), further distinguishing abstract classes from interfaces
- Unlike an interface, the **`abstract`** modifier must be applied to each abstract method
- However, a class declared as abstract does not need to contain abstract methods

Abstract Classes & Inheritance

- An abstract class **cannot be instantiated**
- We use the modifier **abstract** on the class header to declare a class as abstract:

```
public abstract class Whatever
{
    // contents
}
```

Abstract Classes & Inheritance

- Abstract classes make sense with inheritance. An abstract class can be seen as a placeholder in a class hierarchy that represents a generic concept.

- The child class of an abstract class **must override** the abstract methods of the parent, or it **too will be considered abstract**

- An abstract method cannot be defined as **`final`** (because it must be overridden) or **`static`** (because it has no definition yet)

- The use of abstract classes is a design decision – it helps us establish common elements in a class that is too general to instantiate

Abstract Classes & Inheritance: Example

```
public abstract class Animal
{
    public String name;
    public abstract void makeSound();
    public abstract void move();
    public Animal(String animalName){
        name = animalName;
    }
    public String getName(){
        return name;
    }
    public String toString(){
        return getName();
    }
}
```

`abstract` is a reserved word

Abstract methods

Non-abstract methods

Abstract Classes & Inheritance: Example

```
public abstract class Animal
{
    public String name;
    public abstract void makeSound();
    public abstract void move();
    public Animal(String animalName){
        name = animalName;
    }
    public String getName(){
        return name;
    }
    public String toString(){
        return getName();
    }
}
```

abstract is a reserved word

Abstract methods

Non-abstract methods

Abstract Classes & Inheritance: Example

The child class of an abstract class must override the abstract methods of the parent, or it too will be considered abstract

```
public class Cat extends Animal
{
        public Cat(String catName){
         super(catName);
        }
        public void makeSound(){
         System.out.println(" Meow Meow");
        }
        public void move(){
         System.out.println("The cat is walking...");
        }
}
```

Overriding the abstract methods of the parent class

Abstract Classes & Inheritance: Example

```
public class Dog extends Animal
{
        public Dog(String dogName){
         super(dogName);
        }
        public void makeSound(){
         System.out.println(" Woof Woof");
        }
        public void move(){
         System.out.println("The dog is walking...");
        }
}
```

The child class of an abstract class must override the abstract methods of the parent, or it too will be considered abstract

Overriding the abstract methods of the parent class

Driver Class

```
class AnimalDriver {
      public static void main(String args[]){
            Cat myCat = new Cat("Camelos");
            Dog myDog = new Dog("Hero");

            System.out.print(myCat);
            myCat.makeSound();

            System.out.print(myDog);
            myDog.makeSound();
      }
}
```

Output

```
Camelos Meow Meow
Hero Woof Woof
```

Outline

- Interfaces
- Abstract classes
- Abstract classes and Inheritance
- Interfaces and Inheritance

Interface Hierarchies

- Inheritance can be applied to interfaces as well as classes
- One interface can be derived from another interface
- The child interface inherits all abstract methods of the parent
- A class implementing the child interface must define all methods from both the ancestor and child interfaces
- All members of an interface are public

Interface Hierarchies: Example

```
public interface Animal {
   public void makeSound();
   public void move();
}
```

Parent interface

```
public interface Mammal extends Animal {
  public void breathe();
}
```

Child interface

The child interface inherits all abstract methods of the parent

Interface Hierarchies: Example

```
public interface Animal {
   public void makeSound();
   public void move();
}
```

Parent interface

```
public interface Mammal extends Animal {
  public void breathe();
}
```

Child interface

The child interface inherits all abstract methods of the parent

Interface Hierarchies: Example

```
public class Dog implements Mammal
{
  public void makeSound(){
   System.out.println(" Woof Woof");
  }
  public void move(){
    System.out.println("The dog is walking...");
  }
  public void breathe(){
   System.out.println("The dog is breathing");
  }
}
```

A class implementing the child interface must define all methods from both the ancestor and child interfaces. In this example, the Dog class must implement all the abstract methods of the Mammal and Animal interfaces