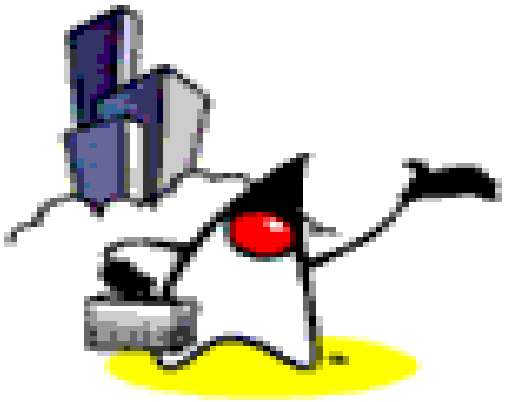
Abstract Class & Java Interface

(Importance scale: *****, Extremely important)

Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"

Agenda

- What is an Abstract method and an Abstract class?
- What is Interface?
- Why Interface?
- Interface as a Type
- Interface vs. Class
- Defining an Interface
- Implementing an Interface
- Implementing multiple Interface's
- Inheritance among Interface's
- Rewriting an Interface



What is an Abstract Method?
&
What is an Abstract Class?

What is an Abstract Method?

- Method that does not have implementation (body)
- To create an abstract method, just write the method declaration without the body and use the abstract keyword
  - Since there is no body of code, there is no { }

- For example,

```
// Note that there is no body of code
public abstract void someAbstractMethod();
```

What is an Abstract Class?

- An abstract class is a class that contains one or more abstract methods
  - Compile error occurs if no abstract method is present
- An abstract class cannot instantiated

  // You will get a compile error on the following code

  MyAbstractClass a1 = new MyAbstractClass();
- Another class (Concrete class) has to be created to provide implementation of all abstract methods
  - Concrete class has to implement all abstract methods of the abstract class in order to be used for instantiation - compile error if not all abstract methods are implemented
  - Concrete class uses extends keyword

Let's say we have an Abstract Class

```
public abstract class LivingThing {
    public void breath(){
        System.out.println("Living Thing breathing...");
    }

    public void eat(){
        System.out.println("Living Thing eating...");
    }

    /**
     * Abstract method walk()
     * We want this method to be implemented by
     * Concrete classes.
     */
    public abstract void walk();
}
```

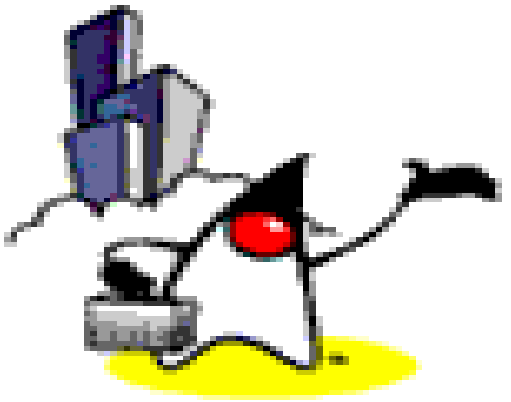
Extending an Abstract Class

- When a concrete class extends the *LivingThing* abstract class, it must implement all abstract methods - there is only one to implement *walk()*
- For example,

```
// Human class below is a concrete class
// because it implements all abstract methods
// (there is only one in this example) of LivingThing
// abstract class
public class Human extends LivingThing {

   public void walk(){
      // implementation of walk() method
      System.out.println("Human walks...");
   }
}
```

When to use Abstract Methods & Abstract Class?

- Abstract methods are usually declared where two or more subclasses (Concrete classes) are expected to fulfill a similar role in different ways through different implementations (this is Polymorphism)
  - These subclasses extend the same Abstract class and provide different implementations for the abstract methods of the Abstract class
- Use abstract classes to define broad types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.



What is Interface?

What is an Interface?

- All methods of an interface are abstract methods (while in an abstract class, only some methods are abstract methods)
  - Defines the signatures of a set of methods, without the body (implementation of the methods)
  - No need to use *abstract* modifier for the methods since all methods in an interface are considered as abstract methods
- A concrete class must implement the interface (all abstract methods of the Interface)
- Use *interface* modifier to the class declaration

What is an Interface for?

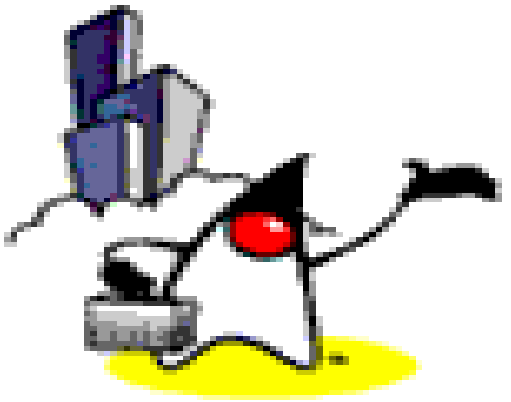
- It defines a public way of specifying the behavior of classes
  - Defines a contract between “client” (user of the interface) and “provider” (implementation) of some task

Example: Interface Relation

```
// Note that Interface contains just set of method
// signatures without any implementations.
// No need to say abstract modifier for each method
// since it assumed.
public interface Relation {
        public boolean isGreater( Object a, Object b);
        public boolean isLess( Object a, Object b);
        public boolean isEqual( Object a, Object b);
}
```

Example 2: Interface OperateCar

```
public interface OperateCar {

   // constant declarations, if any

   // method signatures
   int turn(Direction direction,
       double radius, double startSpeed, double endSpeed);
   int changeLanes(Direction direction, double startSpeed,
                   double endSpeed);
   int signalTurn(Direction direction, boolean signalOn);
   int getRadarFront(double distanceToCar,
                     double speedOfCar);
   int getRadarRear(double distanceToCar,
                     double speedOfCar);
         ......
   // more method signatures
}
```



Why Interface?

(These are critical concepts you have to understand!!!)

Why do we use Interfaces? Reason #1

- In order to reveal an object's programming interface (functionality of the object) without revealing its implementation
  - This is the concept of encapsulation
  - The implementation can change without affecting the caller (user) of the interface
  - The caller does not need the implementation class at compile time - only interface is needed
    - Different groups can work in parallel with only interfaces defined
    - During run-time, object instance gets created (from implementation) and is associated with the interface type

Why do we use Interfaces? Reason #2

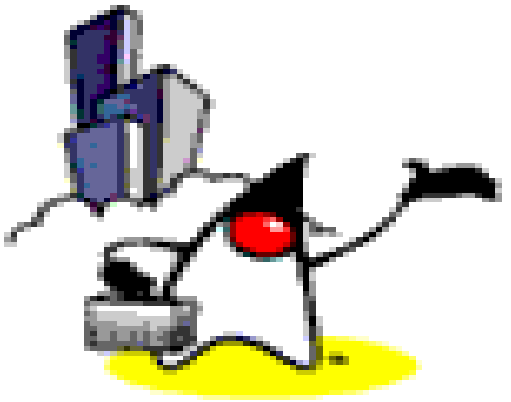
- To have unrelated classes implement similar methods (behaviors)
- Example:
  - Class Line and class MyInteger
    - They are not related through inheritance
    - But you want both to implement comparison methods
      - checkIsGreater(Object x, Object y)
      - checkIsLess(Object x, Object y)
      - checkIsEqual(Object x, Object y)
  - Define Comparison interface which has the three abstract methods above and have Line and MyInteger classes to implement the Comparison interface

Why do we use Interfaces? Reason #3

- To model multiple inheritance - you want to impose multiple sets of behaviors to your class
  - Each set is defined as an interface
  - A class can implement multiple interfaces while it can extend only one class

Interface vs. Abstract Class

- All methods of an Interface are abstract methods while some methods of an Abstract class are abstract methods
  - Abstract methods of Abstract class must have abstract modifier
- An interface can only define constants while abstract class can have fields
- Interfaces have no direct inherited relationship with any particular class, they are defined independently
  - Interfaces themselves have inheritance relationship among themselves, however

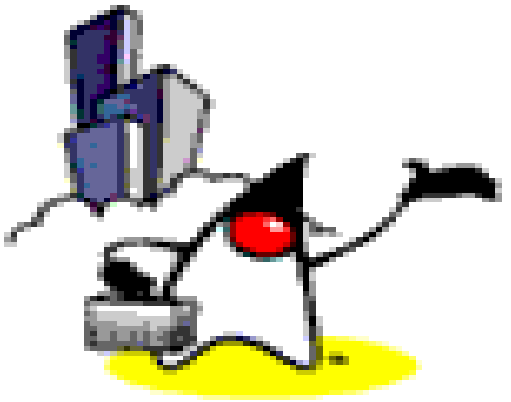


Interface as a Type

(This is a critical concept you have to understand!!!)

Instance as a Type (Very important)

- When you define a new interface, you are defining a new reference type
  - You can use an interface anywhere you can use a class as a reference type
- If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface (implementation class)
- Let's say Person class implements *PersonInterface* interface, now you can do
  - Person p1 = new Person();
  - PersonInterface pi1 = p1;
  - PersonInterface pi2 = new Person();



Interface vs. Class

Interface vs. Class: Commonality

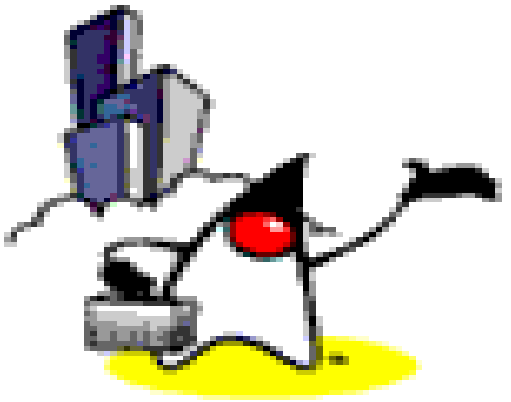
- Interface and Class can both define methods
- Interfaces and classes are both types
  - This means that an interface can be used in places where a class can be used
  - For example:

```
// Recommended practice
PersonInterface pi = createPerson();

// Not recommended practice because
// - Person class implementation has to be
//   present even during compile time
// - Implementation change in Person class
//   forces this code to be recompiled.
Person pc = createPerson();
```

Interface vs. Class: Differences

- The methods of an Interface are all abstract methods
  - They cannot have bodies
- You cannot create an instance from an interface - you have to create a concrete class that implements the interface for object instantiation

```
// Compile error
PersonInterface pi = new PersonInterface();
```



Defining Interface

Defining an Interface

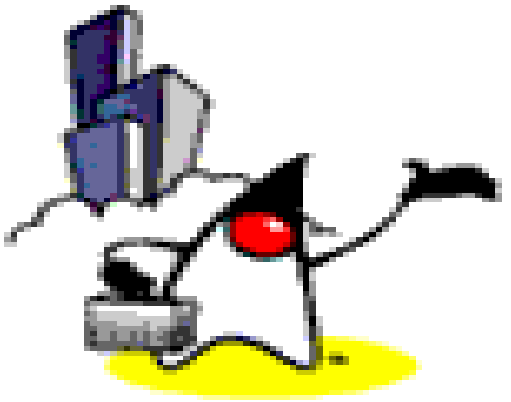
- To define an interface, use *interface* keyword

```
public interface [InterfaceName] {
      //some methods without the body
}
```

Defining an Interface

- As an example, let's create an interface that defines relationships between two objects according to “some order” of the objects.

```
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```



Implementing Interface

Implementing an Interface

- To create a concrete class that implements an interface, use the **implements** keyword.

```
/**
 * Line class implements Relation interface
 */
public class Line implements Relation {
      private double x1;
      private double x2;
      private double y1;
      private double y2;

      public Line(double x1, double x2, double y1, double y2){
          this.x1 = x1;
          this.x2 = x2;
          this.y1 = y1;
          this.y2 = y2;
      }

      // More code follows
```

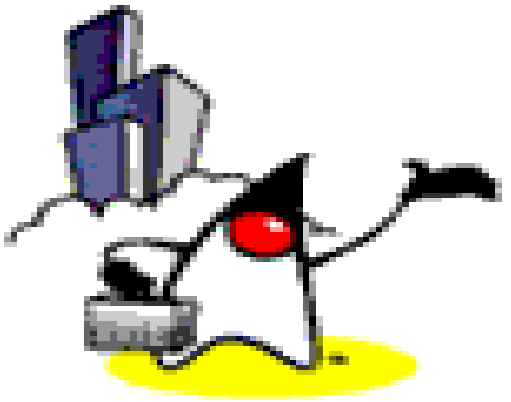
Implementing an Interface

```
  public double getLength(){
    double length = Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)* (y2-y1));
    return length;
  }

  // Line class has to implement the following abstract methods
  public boolean isGreater( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen > bLen);
  }

  public boolean isLess( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen < bLen);
  }

  public boolean isEqual( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen == bLen);
  }
}
```

Implementing an Interface

- When your class tries to implement an interface, always make sure that you implement all methods of that interface, or else, you would encounter this error,

```
Line.java:4: Line is not abstract and does not
  override abstract method
  isGreater(java.lang.Object,java.lang.Object) in
  Relation
public class Line implements Relation
       ^
1 error
```



Implementing Multiple Interfaces

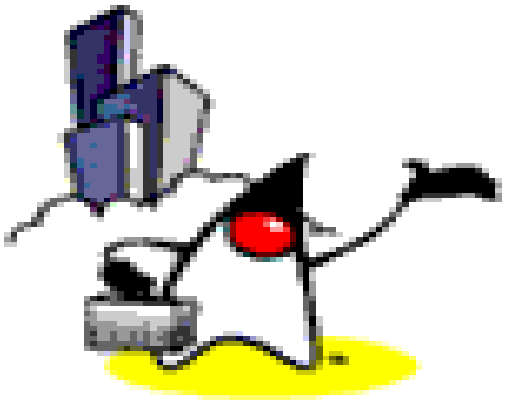
Relationship of an Interface to a Class

- A concrete class can only extend one super (parent) class, but it can implement multiple Interfaces
  - The Java programming language does not permit multiple inheritance, but interfaces provide an alternative.
- All abstract methods of all interfaces have to be implemented by the concrete class, however

Example: Implementing Multiple Interfaces

- A concrete class extends one super class but multiple Interfaces:

```
public class ComputerScienceStudent
                extends Student
                implements PersonInterface,
                           AnotherInterface,
                           ThirdInterface{

   // All abstract methods of all interfaces
   // need to be implemented.
}
```

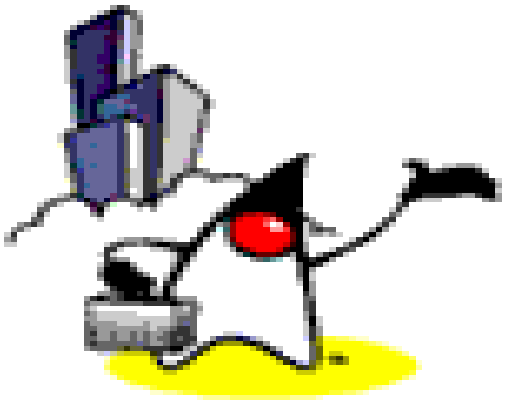


Inheritance Among Interfaces

Inheritance Among Interfaces

- Interfaces are not part of the class hierarchy
- However, interfaces can have inheritance relationship among themselves

```
public interface PersonInterface {
   void doSomething();
}

public interface StudentInterface
                  extends PersonInterface {
   void doExtraSomething();
}
```



Rewriting Interfaces

Problem of Rewriting an Existing Interface

- Consider an interface that you have developed called *DoIt*:

```
public interface DoIt {
   void doSomething(int i, double x);
   int doSomethingElse(String s);
}
```

- Suppose that, at a later time, you want to add a third method to *DoIt*, so that the interface now becomes:

```
public interface DoIt {
   void doSomething(int i, double x);
   int doSomethingElse(String s);
   boolean didItWork(int i, double x, String s);
}
```

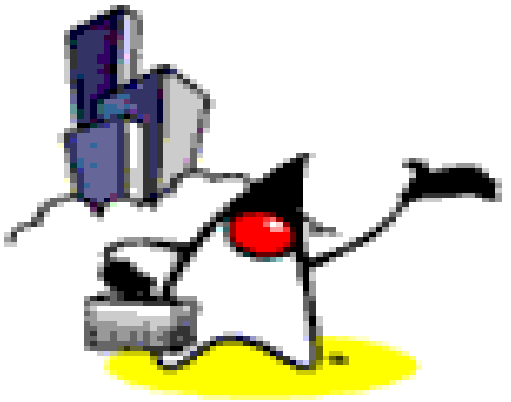
- If you make this change, however, all classes that implement the old *DoIt* interface will break because they don't implement all methods of the the new *Doit* interface anymore

Solution of Rewriting an Existing Interface

- The solution is to create a child interface from an existing interface
- For example, you could create a *DoItPlus* interface that extends *DoIt*:

```
public interface DoItPlus extends DoIt {
   boolean didItWork(int i, double x, String s);
}
```

- Now, old code of *DoIt* interface still work while new code can be written using the *DoItPlus* interface



When to Use an Abstract Class over an Interface?

When to use an Abstract Class over Interface?

- For non-abstract methods, you want to use them when you want to provide common implementation code for all sub-classes
  - Reducing the duplication
- For abstract methods, the motivation is the same with the ones in the interface – to impose a common behavior for all child-classes without dictating how to implement it
- Remember a concrete class can extend only one super class whether that super class is in the form of concrete class or abstract class

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with JPassion!"