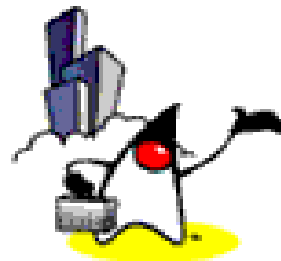




Abstract Class & Java Interface



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
- Interface and Polymorphism
- Rewriting an Interface





**What is
an Abstract Class?**

Abstract Methods

- Methods that do not have implementation (body)
- To create an abstract method, just write the method declaration without the body and use the **abstract** keyword
 - No { }
- For example,

```
// Note that there is no body  
public abstract void someMethod();
```



Abstract Class

- An abstract class is a class that contains one or more **abstract methods**
- An abstract class cannot be instantiated
 - // You will get a compile error on the following code
 - MyAbstractClass a1 = new MyAbstractClass();
- Another class (Concrete class) has to provide implementation of abstract methods
 - Concrete class has to implement all abstract methods of the abstract class in order to be used for instantiation
 - Concrete class uses **extends** keyword



Sample 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 a  
     * Concrete class.  
     */  
    public abstract void walk();  
}
```



Extending an Abstract Class

- When a concrete class extends the `LivingThing` abstract class, it must implement the abstract method `walk()`, or else, that subclass will also become an abstract class, and therefore cannot be instantiated.

- For example,

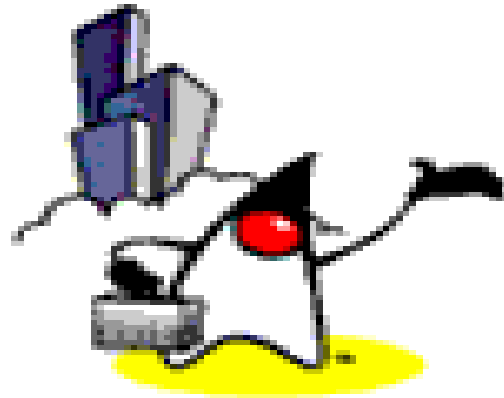
```
public class Human extends LivingThing {  
  
    public void walk(){  
        System.out.println("Human walks...");  
    }  
  
}
```



When to use Abstract Methods & Abstract Class?

- Abstract methods are usually declared where two or more subclasses are expected to fulfill a similar role in different ways through different implementations
 - These subclasses extend the same Abstract class and provide different implementations for the abstract methods
- 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?

- It defines a standard and public way of specifying the behavior of classes
 - Defines a contract
- All methods of an interface are abstract methods
 - Defines the signatures of a set of methods, **without the body (implementation of the methods)**
- A concrete class must implement the interface (all the abstract methods of the Interface)
- It allows classes, regardless of their locations in the class hierarchy, to implement common behaviors



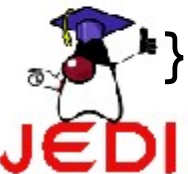
Example: Interface

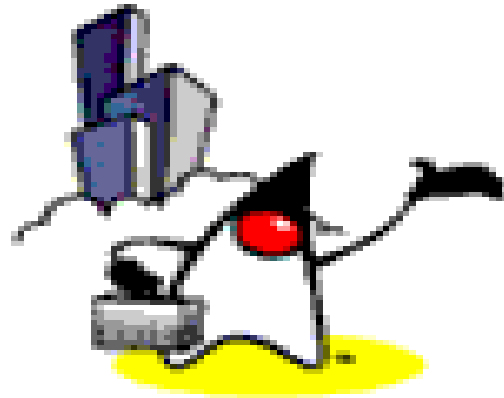
```
// Note that Interface contains just set of method
// signatures without any implementations.
// No need to say abstract modifier for each method
// since it assumed.
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
  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



Example 2: OperatorCar Interface

```
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?

Why do we use Interfaces?

Reason #1

- 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 of the interface
 - The caller does not need the implementation at the compile time
 - It needs only the interface at the compile time
 - During runtime, actual object instance is associated with the interface type



Why do we use Interfaces?

Reason #2

- To have unrelated classes implement similar methods (behaviors)
 - One class is not a sub-class of another
- Example:
 - Class `Line` and class `MyInteger`
 - They are not related through inheritance
 - 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



Why do we use Interfaces?

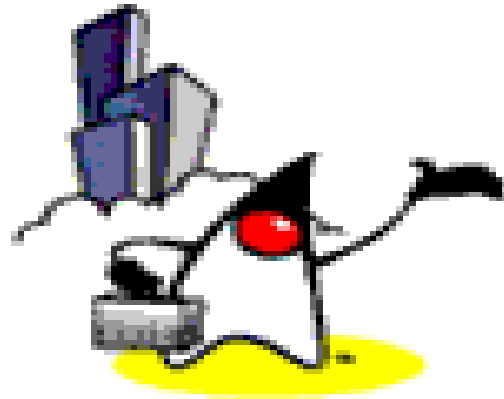
Reason #3

- To model multiple inheritance
 - 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 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





Interface as a Type

Interface as a Type

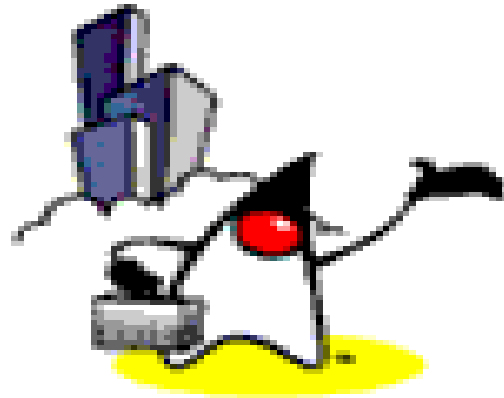
- When you define a new interface, you are defining a new reference type
- You can use interface names anywhere you can use any other type name
- If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface



Example: Interface as a Type

- Let's say Person class implements PersonInterface interface
- You can do
 - `Person p1 = new Person();`
 - `PersonInterface pi1 = p1;`
 - `PersonInterface pi2 = new Person();`





Interface vs. Class

Interface vs. Class: Commonality

- 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 = new Person();

// Not recommended practice
Person pc = new Person();
```

- Interface and Class can both define methods



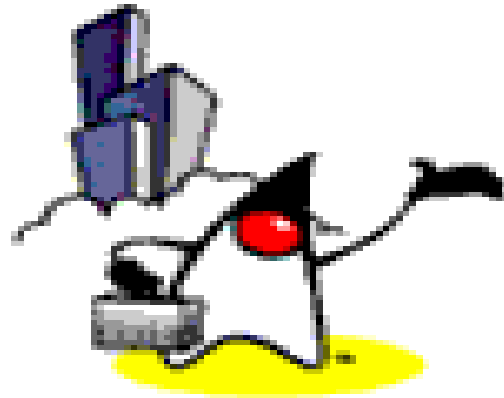
Interface vs. Class: Differences

- The methods of an Interface are all abstract methods
 - They cannot have bodies
- You cannot create an instance from an interface
 - For example:

```
PersonInterface pi = new PersonInterface(); //ERROR!
```

- An interface can only be implemented by classes or extended by other interfaces





Defining Interface

Defining Interfaces

- To define an interface, we write:

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



Defining Interfaces

- As an example, let's create an interface that defines relationships between two objects according to the “natural 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 Interfaces

- 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
```



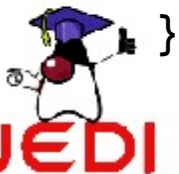
Implementing Interfaces

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

```
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 Interfaces

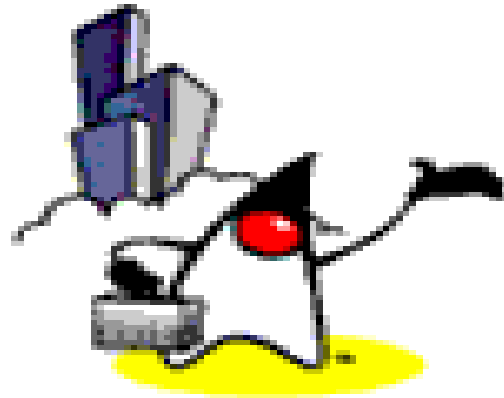
- When your class tries to implement an interface, always make sure that you implement all the methods of that interface, or else, you would encounter this error,

```
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 Class

- Implementing class can have its own methods
- Implementing class extend a single super class or abstract class



Implementing Multiple Interfaces

Relationship of an Interface to a Class

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



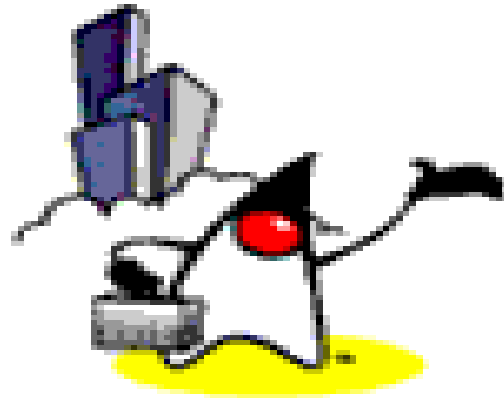
Example: Implementing Multiple Interfaces

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

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

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





Inheritance Among Interfaces

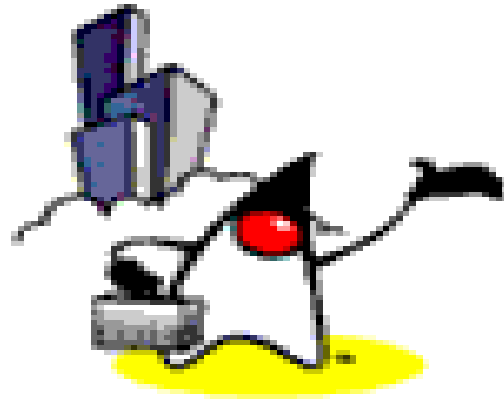
Inheritance Among Interfaces

- 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();  
}
```

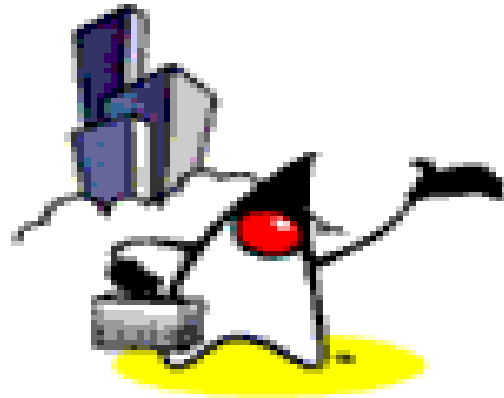




Interface & Polymorphism

Interface and Polymorphism

- Interfaces exhibit polymorphism as well, since program may call an interface method, and the proper version of that method will be executed depending on the type of object instance passed to the interface method call



Rewriting Interfaces

Problem of Rewriting an Existing Interface

- Consider an interface that you have developed called Dolt:

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

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

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

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



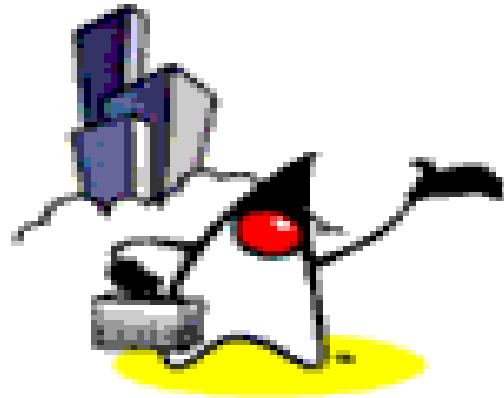
Solution of Rewriting an Existing Interface

- Create more interfaces later
- For example, you could create a DoltPlus interface that extends Dolt:

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

- Now users of your code can choose to continue to use the old interface or to upgrade to the new 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 sub-classes without dictating how to implement it
- Remember a concrete can extend only one super class whether that super class is in the form of concrete class or abstract class





Abstract Class & Java Interface

