# Interfaces

the power of inheritance without using inheritance
*(? Unheritance ?)*



# Inheritance

Implements an “is-a” relationship

Allows hierarchies to be built

Allows manipulating heterogeneous collections

Array of Mammals? - any subclass of Mammal

Polymorphism

the *type of the object*, not the type of the reference, is used (at runtime) to determine which method to invoke



# Limitations

To manipulate these heterogeneous collections, they must have a common ancestor

with the method defined

Example:

to sort Animals by weight, Animal must define (or define as abstract) the method/property of weight ( )

Then, all subclasses will have that method and can be compared

What if, I want to sort by weight Animals or Vehicles?



# Java *Interfaces*

Instead of inheritance

When natural hierarchy doesn’t work

Interfaces define an “acts-as” relationship

Interfaces define how an object must behave

```
public interface Weighable {
    public int weight ( );
}
```

Any class can “implement” the interface



# Interface

Manipulation is done like:

- Weighable [ ] w = new Weighable[100];

…can call w[i].weight( ) …

Any class that implements the Weighable interface is eligible to participate!



# Java Interfaces

```
public interface Weighable {
  public int weight ( );
}
```

Class & Methods are abstract *( by definition )*

*cannot be instantiated*

*no constructor*

No class variables *(compiler will change them to constants)*

Class can implement or make abstract as well

```
public class Animal implements Weighable {
  @Override
  public int weight() {
    return 50;   //calculate something and return
  }
  … rest of class implementation…
```

# Example

Eclipse

Let Eclipse type some code for you

(but notice what it does)



# Interfaces

Polymorphism

the *type of the object*, not the type of the reference, is used (at runtime) to determine which method to invoke

With interfaces, collection of things that are “weighable” can be manipulated

An Inventory System can manipulate “trackable” items

A GUI manipulates “drawable” things

I can trade items that have value

I can communicate with things that speak



# API Examples

- `public interface Comparable <T>`

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the class's `compareTo` method is referred to as its *natural comparison method*.

- `int compareTo(T o);`

Implementing this method allows you to use built-in capabilities of Collections like:

`Arrays.sort(arr)` or `Collections.sort(myArrayList)`

Those sorts rely on that method existing — that is, they rely on your objects in the array or list to be *Comparable*



# API Examples

- `public interface MouseListener`

*A mouse listener receives (handles) “interesting” mouse events (press, release, click, enter, exit)*

`void mouseClicked(MouseEvent e)`
*Invoked when the mouse button has been clicked (pressed and released) on a component.*

`void mouseEntered(MouseEvent e)`
*Invoked when the mouse enters a component.*

`void mouseExited(MouseEvent e)`
*Invoked when the mouse exits a component.*

`void mousePressed(MouseEvent e)`
*Invoked when a mouse button has been pressed on a component.*

`void mouseReleased(MouseEvent e)`
*Invoked when a mouse button has been released on a component.*



# Interfaces

Implementing Class *(if it is not abstract)*

must implement all methods of the interface

*usually* implements more public/private methods (like any other class)

Interface definition

only specifies required methods

may have super interface (i.e. an interface can extend an interface)

Java API: Interface names are *italicized*

*Interface names sometimes end in: -able, -er, -or*



# Summary

Interfaces are a collection of abstract methods

An interface is not a class

Specifies *behavior* only

by specifying the methods that are needed

Cannot instantiate an interface *(since it is abstract)*

All methods are abstract

Interfaces can be part of a hierarchy *(super interfaces)*

Classes *implement* an interface

Classes may implement multiple interfaces *(comma separated)*