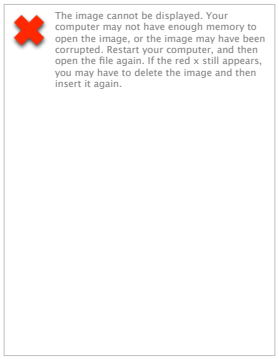


April 19: Interfaces Continued...

CMPSCI 121, Spring 2012

Introduction to Problem Solving with Computers

Prof. Learned-Miller

Polymorphism: Operating on arbitrary types of objects.

```
 1 import java.util.Random;
 2
 3 public class Chance extends Random{
 4
 5   public int throwDie(){
 6     return (1 + nextInt(6));}
 7
 8   public int throwDice(){
 9     return (throwDie() + throwDie());}
10
11   public void shuffle(int[] nums){
12     // method randomly shuffles an array of integers
13     int swapPos, temp;
14     for (int i = nums.length-1; i > 0; i--) {
15       swapPos = nextInt(i+1); // pick pos from 0 -> i (i is possible)
16       temp = nums[swapPos]; //  swap vals at i, swapPos
17       nums[swapPos] = nums[i];
18       nums[i] = temp;
19     }
20   }
21 }
```

DrJava

Interface Example:

```
public interface Scoring{

  public double getScore();

  public void setScore(double newScore);
}
```

-Kind of like a class, but can't make one of these
-Doesn't specify implementation of methods, just what they should do.

```
public class CookieSeller implements Scoring
{
  private String name;
  private double boxesSold;

  public CookieSeller(String n, double sold)
  {
    name = n;
    boxesSold = sold;
  }

  public String getName()
  {
    return name;
  }
  public double getBoxesSold()
  {
    return boxesSold;
  }

  public void setName(String newName)
  {
    name = newName;
  }
  public void setBoxesSold(double sold)
  {
    boxesSold = sold;
  }
  public double getScore() // implements interface method
  {
    return boxesSold;
  }

  public void setScore(double sold)// implements interface method
  {
    boxesSold = sold;
  }
}
```

One method for multiple classes.

```
public static int scoreMax(Scoring[] theArray){
  // returns position of entry in array theArray with highest score
  // array theArray is an array of objects from class that implements
  // Scoring interface
  int highPos = 0;
  for(int j = 1; j < theArray.length; j++){
    if (theArray[j].getScore() > theArray[highPos].getScore())
      highPos = j;}
  return highPos;
}
```

One method for multiple classes

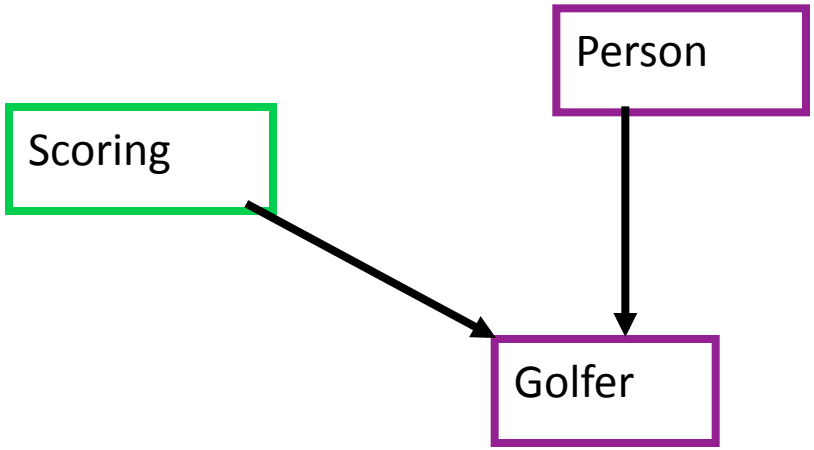
```
public class Scorefns  {
  // contains methods that exploit the Scoring interface

  public static int scoreMax(Scoring[] theArray){
    // returns position of entry in array theArray with highest score
    // array theArray is an array of objects from class that implements
    // Scoring interface
    int highPos = 0;
    for(int j = 1; j < theArray.length; j++){
      if (theArray[j].getScore() > theArray[highPos].getScore())
        highPos = j;}
    return highPos;
  }
}
```

One method for multiple classes

```
public class Scorefns  {
  // contains methods that exploit the Scoring interface

  public static int scoreMax(Scoring[] theArray){
    // returns position of entry in array theArray with highest score
    // array theArray is an array of objects from class that implements
    // Scoring interface
    int highPos = 0;
    for(int j = 1; j < theArray.length; j++){
      if (theArray[j].getScore() > theArray[highPos].getScore())
        highPos = j;}
    return highPos;
  }
}
```

```
int which = Scorefns.scoreMax(golfers);
```

Almost “multiple inheritance”



Golfer extends Person implements Scoring

Another Example:

```
public interface Directions{

   final int NORTH = 0;
   final int EAST = 1;
   final int SOUTH = 2;
   final int WEST = 3;

}
```

-“implements Directions”
is just like adding these attributes to a class.

-Could have achieved the same thing by deriving from a class.

Either of these works...

```
public class Directions {

  final int NORTH = 0;
  final int EAST = 1;
  final int SOUTH = 2;
  final int WEST = 3;

}

public class Car extends Directions {
  ...
}
```

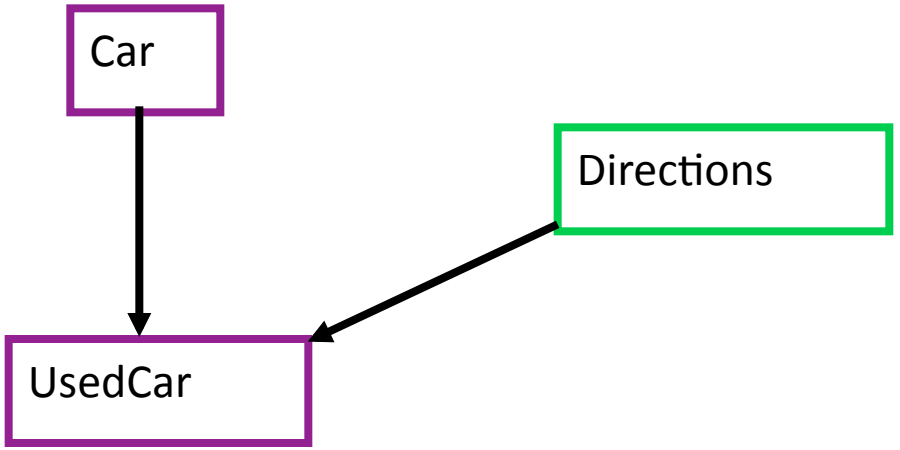
```
public interface Directions {

  final int NORTH = 0;
  final int EAST = 1;
  final int SOUTH = 2;
  final int WEST = 3;

}

public class Car implements Directions {
  ...
}
```

However...

there is only one way to do this...

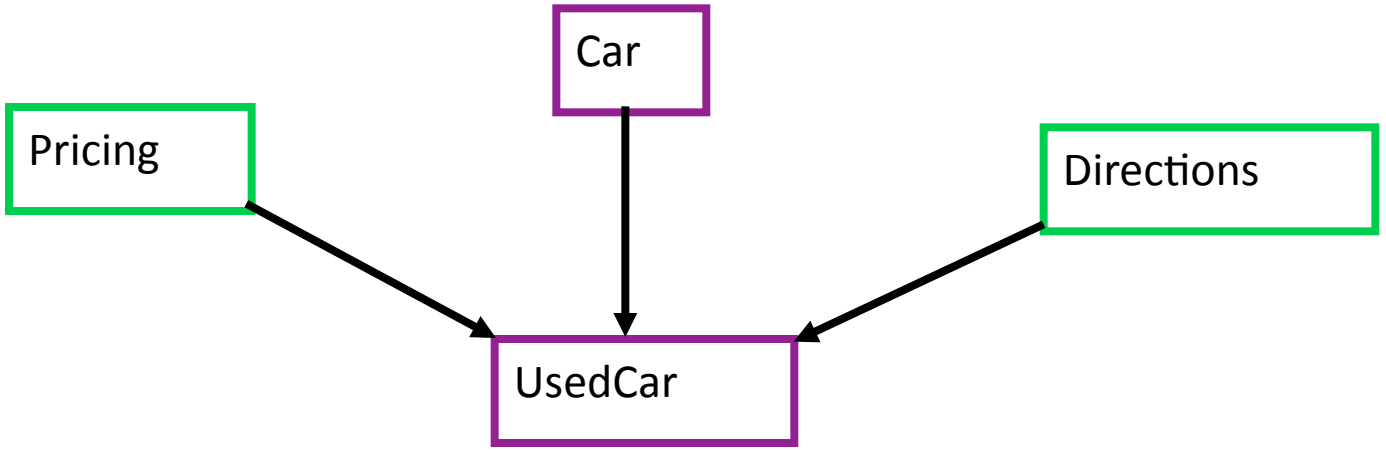
```
public interface Directions {

  final int NORTH = 0;
  final int EAST = 1;
  final int SOUTH = 2;
  final int WEST = 3;

}

public class Car {
  ...
}

public class UsedCar extends Car implements Directions {
  ...
}
```

Almost “multiple inheritance”



UsedCar extends Car implements Directions

Almost “multiple inheritance”



UsedCar extends Car implements Directions, Pricing

Implements vs. Extends

- Add engine type and tire type to Car:
  - public class HotRod extends Car
- Add getCost() and setCost() to Car:
  - Might implement these as a Pricing interface...
    - They are generic and we could use them for other classes.
  - public class sellableCar extends Car implements Pricing
    OR
  - public class Car implements Pricing

More uses of interfaces

```
public int compareTo(Object other);
```

More uses of interfaces

```
public class Infant implements Comparable
{

  private  String name;
  private int age;   // in months

  public Infant(String who, int months){
    name = who;
    age = months;
  }

  public String getName(){
    return name;}

  public int getAge(){
    return age;}

  public void anotherMonth(){age = age + 1;}

  public int compareTo(Object other){
    int b = ((Infant)other).getAge();
    int a = this.age;
    return(a-b);
  }
}
```

Using one interface to implement another

```
public int compareTo(Object other)
{
 String b = ((Infant)other).getName();
 String a = this.name;
 return(a.compareTo(b));
}
```