# 1.00 Lecture 9

**Methods and Objects**
**Access**

**Reading for next time: Big Java: sections 7.6, 7.7**

# Storing weather for a city

- **We'll first show an example of storing temperature and precipitation data for cities.**
  - **Our classes have some methods with arguments**
  - **We'll examine the methods and arguments: they'll do what you expect**
- **We'll then extend our weather example to have each city store a weather information object that collects the weather data in one place**
  - **These classes also have methods with arguments**
  - **We'll examine these methods too. They also do what you expect (but your expectations will have to be a bit more sophisticated!)**

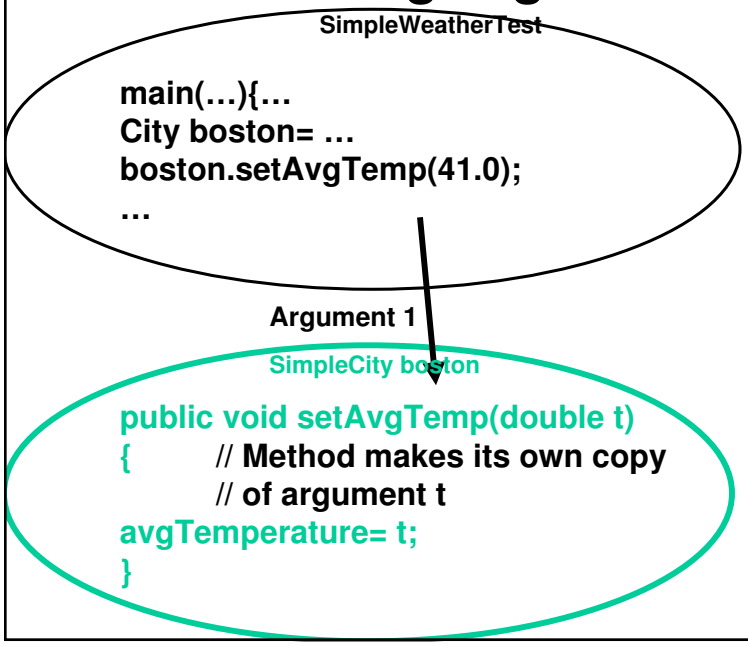
# SimpleCity

```
public class SimpleCity {
   private String name;
   private double avgTemperature;
   private double precipAmt;

   public SimpleCity(String n, double a, double p) {
        name= n;
        avgTemperature= a;
        precipAmt= p;
   }
   public String getName() {
        return name;
   }
   public double getAvgTemperature() {
        return avgTemperature;
   }
   public void setAvgTemperature(double t) {
        avgTemperature= t;
   }
   public double getPrecipAmt() {
        return precipAmt;
   }  }
```

# SimpleWeatherTest

```
public class SimpleWeatherTest {
   public static void main(String[] args) {

     SimpleCity boston= new SimpleCity("Boston", 40.0, 0.0);
     SimpleCity cambridge= new SimpleCity("Cambridge", 40.0, 0.0);

     // Now revise the Boston weather, which was corrected
     boston.setAvgTemperature(41.0);

     System.out.println("Boston: " + boston.getAvgTemperature());
     System.out.println("Cambridge: "+
         cambridge.getAvgTemperature());
   }
}

// What is the output of this program?
```

# Passing Arguments




Communi-cation only via arg list, return value

Setting the Cambridge temperature would be the same

# Method/Object Exercise

- We now change SimpleCity and SimpleWeatherTest slightly
  - We rename them City and WeatherTest
  - We also introduce a simple Weather class
  - We'll look at them briefly on the next slides

# Weather class

```
public class Weather {
   private double avgTemperature;
   private double precipAmt;
   public Weather(double a, double p) {
        avgTemperature= a;
        precipAmt= p;
   }
   public void setAvgTemp(double t) {
        avgTemperature= t;
   }
   public void setPrecipAmt(double pr) {
        precipAmt= pr;
   }
   public String toString() {
        return ("Temperature: "+avgTemperature+
                " ; Precipitation: "+precipAmt);
   }
}
```

# City class

```
public class City {
   private String name;
   private Weather cityWeather;

   public City(String n, Weather c) {
        name= n;
        cityWeather= c;
   }
   public String getName() {
        return name;
   }
   public Weather getWeather() {
        return cityWeather;
   }
}
```

# WeatherTest

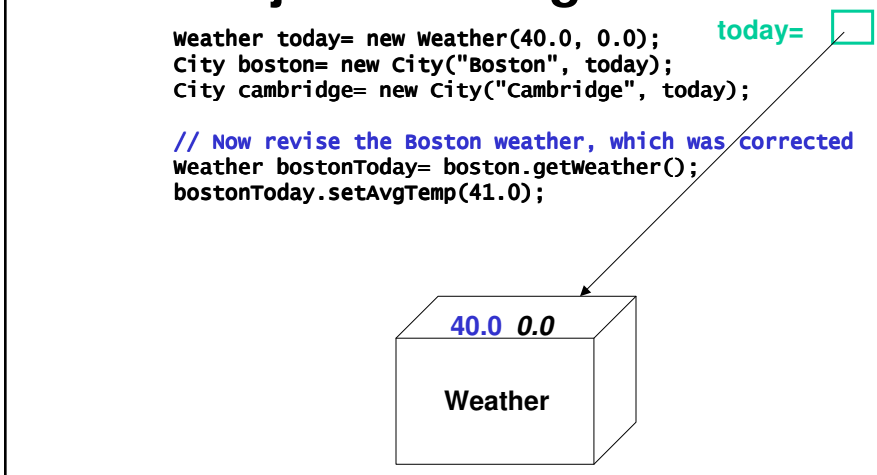
```
public class WeatherTest {
   public static void main(String[] args) {
        Weather today= new Weather(40.0, 0.0);
        City boston= new City("Boston", today);
        City cambridge= new City("Cambridge", today);

        // Now revise the Boston weather, which was corrected
        Weather bostonToday= boston.getWeather();
        bostonToday.setAvgTemp(41.0);

        System.out.println("Boston: " + boston.getWeather());
        System.out.println("Cambridge: "+ cambridge.getWeather());
   }
}
```

# Exercise- Weather classes

- Download Weather, City, WeatherTest
- Import them into Eclipse
- Before running them, think about what the output will be
- Compile and run them
- Is the output what you expected?

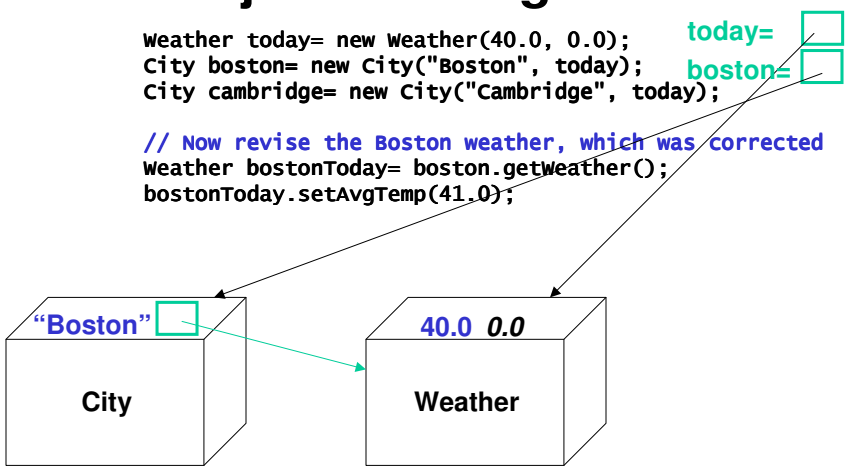
# Objects As Arguments

```
Weather today= new Weather(40.0, 0.0);
City boston= new City("Boston", today);
City cambridge= new City("Cambridge", today);

// Now revise the Boston weather, which was corrected
Weather bostonToday= boston.getWeather();
bostonToday.setAvgTemp(41.0);
```



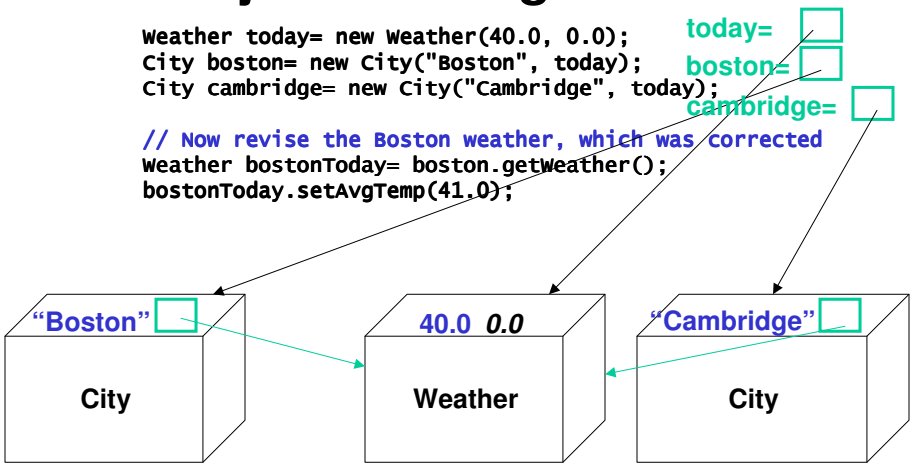

# Objects As Arguments

```
Weather today= new Weather(40.0, 0.0);
City boston= new City("Boston", today);
City cambridge= new City("Cambridge", today);

// Now revise the Boston weather, which was corrected
Weather bostonToday= boston.getWeather();
bostonToday.setAvgTemp(41.0);
```



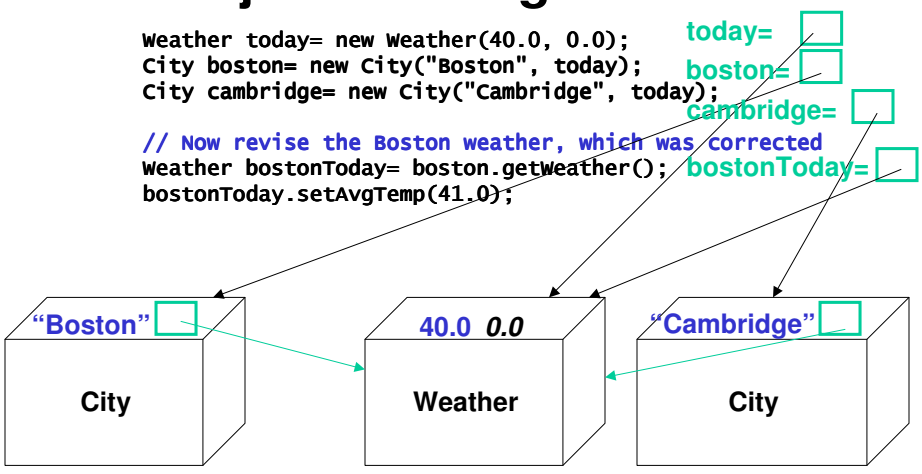
# Objects As Arguments

```
Weather today= new Weather(40.0, 0.0);
City boston= new City("Boston", today);
City cambridge= new City("Cambridge", today);

// Now revise the Boston weather, which was corrected
Weather bostonToday= boston.getWeather();
bostonToday.setAvgTemp(41.0);
```



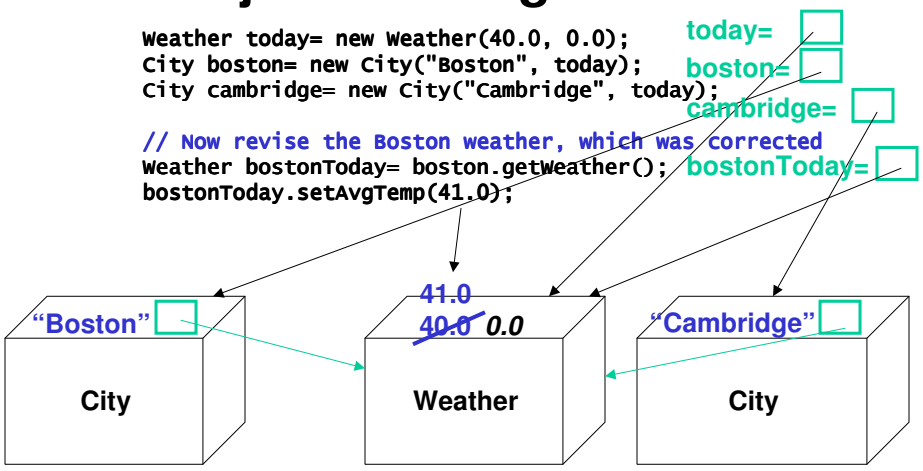

# Objects As Arguments

```
Weather today= new Weather(40.0, 0.0);
City boston= new City("Boston", today);
City cambridge= new City("Cambridge", today);

// Now revise the Boston weather, which was corrected
Weather bostonToday= boston.getWeather();
bostonToday.setAvgTemp(41.0);
```



# Objects As Arguments

```
Weather today= new Weather(40.0, 0.0);
City boston= new City("Boston", today);
City cambridge= new City("Cambridge", today);

// Now revise the Boston weather, which was corrected
Weather bostonToday= boston.getWeather();
bostonToday.setAvgTemp(41.0);
```




When objects are passed as arguments to methods, the method makes a copy of the reference to the object, not a copy of the object! Why?

# Method Calls With Objects

- When passing object references as arguments to a method:
  - The method makes its own copy of the references
  - It makes changes to the objects through its local copies of the references
  - No changes can be made to the references (arguments)
    - The method can't change the reference to another object, for example
  - Results are returned through the return value, which may be an object
- When passing built-in data types as arguments to a method:
  - The method makes its own copy of the built-in variables
  - It makes changes to its local copies only
  - No changes are made to the arguments
  - Results are returned through the return value

# If you don't like this…

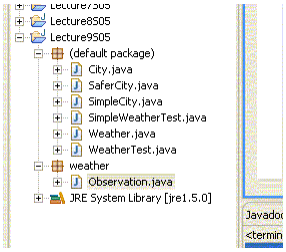
- When you pass an object reference as an argument to a method, the method may make its own local copy of the object:

```
public class SaferCity {
   private String name;
   private Weather cityWeather;

   public SaferCity(String n, Weather c) {
        name= n;
        double temp= c.getAvgTemperature();
        double prec= c.getPrecipAmt();
        cityWeather= new Weather(temp, prec);
   }
   public String getName() {
        return name;
   }
   public Weather getWeather() {
        return cityWeather;
   }
}  // Weather must have methods getAvgTemperature(), getPrecipAmt()
```

# Access: Variables, Methods

- Instance and static variables and methods have 4 access modifiers:
  - Private: Access only to own class' methods
    - Data fields should be private, almost always
    - Other objects of same class can access private variables
  - Public: Access to all methods, all classes
    - Methods intended for other class' use are public
    - Methods for internal use only are private
  - Package: Access to methods of classes in same package (a package is a group of classes)
    - This is the default, alas. Always specify scope explicitly
    - No 'package' keyword; it's the default with no keyword
  - Protected: Used with inheritance (covered later)
    - Like a private variable, except it's visible to derived or subclasses (and, in Java, to other classes in package)

# Packages in Eclipse




**In Eclipse:**
**File -> New -> Package. Type 'weather'**
**Use lower case names by convention**
**Create a new class Observation in weather**
**(File -> New -> Class … )**

# Class Observation

```
package weather;       // Eclipse wrote this for you

// Cut and paste this from the download, or import it
public class Observation {
   private double humidity;
   private double cloudCover;
   public Observation(double h, double c) {
        humidity= h;
        cloudCover= c;
   }
   public double getHumidity() {
        return humidity;
   }
   public double getCloudCover() {
        return cloudCover;
   }
   public String toString() {
        return ("Humidity: "+ humidity+
                        " ; Cloud cover: "+cloudCover);
   }
}
```

## Add Observation to City

- In your default package in Lecture 11:
  - Modify your City class to also have an Observation object:
    - Add `import weather.*;` on 1st line of City.java
    - Add a `private Observation obj`
    - Modify your constructor
    - Add a `getObservation` method
- We'll show the solution on the next slide, and then go on to modify WeatherTest to use your new City and Observation

## Modify WeatherTest

- Change WeatherTest, still in the default package, to:
  - Create a new Observation
  - Place it in Boston and Cambridge
  - Output it (System.out.println) for Boston
  - Remember to import weather.*; on line 1

# Package access

- If we added another class AdvancedObservation to package weather
- And we made humidity and cloudCover package access variables by removing the private keyword (in an Observation2 class)
  - We also remove the getXXX methods as unneeded
- Then AdvancedObservation can use Observation data members, such as humidity and cloudCover directly. It can simply say, for an Observation2 object obs:
  - obs.humidity, or obs.cloudCover

  as if they were in the AdvancedObservation class

# Modified Class Observation

```
package weather;
public class Observation2 {
   double humidity;    // No keyword means package access
   double cloudCover;  // No keyword means package access
   public Observation2(double h, double c) {
        humidity= h;
        cloudCover= c;
   }
}
```

# Class AdvancedObservation

```
package weather;
public class AdvancedObservation {
   double dewpoint;           // Package access
   Observation2 obs;          // Package access
   public AdvancedObservation(double d, Observation2 o) {
        dewpoint= d;
        obs= o;
   }
   public String toString(){
        return ("Humidity: "+obs.humidity+" ; Cloud cover: " +
                obs.cloudCover+ " ; Dewpoint: "+ dewpoint);
        // We can use obs.XXX directly
        // Observation2 could use dewpoint directly also
   }
}
```