1.00 Lecture 9

Methods and Objects
Access

Reading for next time: Big Java: sections 8.6, 8.7

Passing arguments

// Recall this example from Lecture 6

public class TripleTest {
    public static void main(String[] args) {
        double z=5.0;
        System.out.println("z main 1: "+z);
        triple(z);
        System.out.println("z main 2: "+z);
    }
    public static void triple(double z) {
        System.out.println("z 1: "+z);
        z *= 3;
        System.out.println("z 2: "+z);
    }
}
// What is the output, and why?
Passing object arguments

```java
public class Number {
    private double base;
    public Number(double n) {
        base = n;
    }

    public double getBase() {
        return base;
    }

    public void setBase(double b) {
        base = b;
    }

    public String toString() {
        return " + base);";
    }
}
```

Passing objects, p. 2

// This is almost the same as TripleTest
// but with an object

```java
public class NumberTest {
    public static void main(String[] args) {
        Number n = new Number(5.0);
        System.out.println("n main 1: "+n);
        triple(n);
        System.out.println("n main 2: "+n);
    }

    public static void triple(Number n) {
        System.out.println("n 1: "+n);
        n.setBase(n.getBase()*3);
        System.out.println("n 2: "+n);
    }
}
```

// What is the output?
What happened?

```java
class Number {
    private double value;
    public Number(double value) {
        this.value = value;
    }

    public static void main(String[] args) {
        Number n = new Number(5.0);
        n = 5.0;
        System.out.println("n main 1: "+n);
        triple(n);
    }

    public static void triple(Number n) {
        Number copy = new Number(n.value * 3);
    }
}
```

What happened?

```java
class Number {
    private double value;
    public Number(double value) {
        this.value = value;
    }

    public static void main(String[] args) {
        Number n = new Number(5.0);
        n = 5.0;
        System.out.println("n main 1: "+n);
        triple(n);
    }

    public static void triple(Number n) {
        Number copy = new Number(n.value * 3);
    }
}
```
public static void main(String[] args) {
    Number n = new Number(5.0);
    System.out.println("n main 1: "+n);
    triple(n);
    System.out.println("n main 2: "+n);
}

What happened?
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 change the reference to point to another object or set it to null locally, but it won’t change the values in the calling program.
  – 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 can be made to the arguments
  – Results are returned through the return value

Exercise: Weather class

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 + ";
  }
}
Exercise: City class

```java
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;
    }
}
```

Exercise: WeatherTest

```java
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

• Compile and run WeatherTest
• What is the output?

• Assume you wanted to change (correct) only the Boston weather. Change class WeatherTest to do this

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?
Access: Variables, Methods

- Data members and methods have 4 access modifiers:
  - **Private:** Access only to own class methods
    - Data fields should be private
      - Users of an object should use its methods, not its data members
    - Objects of same class can access each others’ private variables
  - **Public:** Access to all methods in all classes
    - Methods should be public
      - They are the primary way that objects are used in Java programs
  - **Package:** Access to methods of classes in same package
    - A package is a group of classes with same first line, e.g.:
      ```java
      package water;
      ```
    - Objects in same package can access each others’ package variables
    - No ‘package’ keyword; it’s the default with no keyword
  - **Protected:** Used with inheritance (covered later)

Packages in Eclipse

In Eclipse:
1. File -> New -> Project. Type ‘Lecture9Package’
2. File -> New -> Package. Type ‘water’
   - Use lower case names by convention
3. Drag MapTest.java, Mountain.java into default pkg
4. Drag Lake.java, River.java into water package

Courtesy of The Eclipse Foundation. Used with permission.
Class Mountain

public class Mountain {
    String name;    // Package access
    private double elevation;
    public Mountain(String n, double e) {
        name= n;
        elevation= e;
    }
    public boolean isTallerThan(Mountain m) {
        if (elevation > m.elevation)
            return true;
        else
            return false;
    }  // Or return (elevation > m.elevation)
}

Class Lake

When you drag file Lake.java onto the package 'water' icon, Eclipse writes the 'package' statement for you.

package water;

public class Lake {
    String name;    // Package access
    double area;    // Package access
    public Lake(String n, double a) {
        name= n;
        area= a;
    }
}
Class River

When you drag file River.java onto the package ‘water’ icon, Eclipse writes the ‘package’ statement for you.

```java
package water;

public class River {
    String name;   // Package access
    double length;  // Package access
    Lake source;   // Package access
    public River(String n, double l, Lake s) {
        name= n;
        length= l;
        source= s;
    }
    public double getSourceLakeSize() {
        return source.area;  // River can see Lake data
    }
}
```

Packages

- Packages are a way to organize software:
  - Group related classes in a package
  - Allow related classes to share package-access data
- You import packages to use them
  - You place them in your namespace by importing them
  - `import water.*;` // At the top of the file
  - This does not bring them into your class’ package
  - If you don’t import a package, you can still use it by, e.g.:
  - `water.Lake winni= new water.Lake("winni", 44586);`
- We use the default package for convenience in 1.00
  - We’ll continue to use it for consistency
  - In code you write after 1.00, don’t use the default package
  - Place every class in a named package
- Use package access appropriately between related classes to improve clarity (less verbose) and convenience
Exercise: Lake, River, Mountain

- Mountain, MapTest (which you will complete) are in default package
- Lake, River are in water package
  - Their data members have package access:
    - Lake can see River data and River can see Lake data
  - Mountain and MapTest cannot see River, Lake data
    - MapTest can see Mountain name (package)
    - But not Mountain elevation (private)
- You will write MapTest to create a Mountain, Lake, River
  - And to output some of their data
- For MapTest to see River and Lake classes, it must in line 1:
  - `import water.*;`
  - 'Import' does not bring River and Lake into this class' package. It just makes them visible to the class

Exercise: Write Class MapTest

```java
// 1. What do you need to import to see Lake, River?

public class MapTest {
    public static void main(String[] args) {
        Mountain w= new Mountain("Washington", 6288);
        Mountain j= new Mountain("Jefferson", 5791);
        // 2. Try to print out Mt Washington elevation, name directly
        // If it doesn't work, fix it. Don't change access.
        // 3. Is Mt Washington or Mt Jefferson taller?

        Lake winni=new Lake("Winnipesaukee", 44586); // Acres
        River m= new River("Merrimack", 120, winni);
        // 4. Try to print area of Lake Winnipesaukee
        // If it doesn't work, fix it. Don't change access.
    }
} // When done, move your MapTest solution to main Lecture9 project
// so you zip up just one project to hand in for active learning
```