CS 112 – Introduction to Computing II

Wayne Snyder
Computer Science Department
Boston University

Today:

Fields vs local variables and scope

Program Structure; the keyword static

Classes vs objects

Creating and using objects

Next time: Creating Java programs with multiple files; public vs private; Object-Oriented Design; Abstract data types; Stacks and Queues.

Reading assignments arebe posted on the web site!

Computer Science

Java Program Structure: Class = Container

Computer Science

A Java class can be thought of as a container for methods:

OverloadTest.java

```
public class OverloadTest {

    static int sum(int n, int m) {
        System.out.println("Calling sum(int n, int m)...");
        return (n+m);
    }

    static double sum(double x, double y) {
        System.out.println("Calling sum(double x, double y)...");
        return (x+y);
    }

    public static void main(String[] args) {

        System.out.println("\nTry sum(2,3)...");
       int n = sum(2, 3);
       System.out.println("Returns " + n);

       System.out.println("\nTry sum(2.3, 3.1)...");
       double x = sum(2.3, 3.1);
       System.out.println("Returns " + x);

       System.out.println("\nTry sum(2, 3.1)...");
       double y = sum(2, 3.1);
       System.out.println("Returns " + y);
    }
}
```

Method

Method

Class

Method

Java Program Structure: Class = Container

Computer Science

The contents of a class can be in any order, as far as execution is concerned: usually main is last, and the other members of the class are organized for readability: put related methods next to each other.

```
public class OverloadTest {

    public static void main(String[] args) {

        System.out.println("\nTry sum(2,3)...");
       int n = sum(2, 3);
       System.out.println("Returns " + n);

       System.out.println("\nTry sum(2.3, 3.1)...");
       double x = sum(2.3, 3.1);
       System.out.println("Returns " + x);

       System.out.println("\nTry sum(2, 3.1)...");
       double y = sum(2, 3.1);
       System.out.println("Returns " + y);
    }

    static int sum(int n, int m) {
        System.out.println("Calling sum(int n, int m)...");
        return (n+m);
    }
```

```
public class OverloadTest {

    static double sum(double x, double y) {
        System.out.println("Calling sum(double x, double y)...");
        return (x+y);
    }

    public static void main(String[] args) {

        System.out.println("\nTry sum(2,3)...");
       int n = sum(2, 3);
       System.out.println("Returns " + n);

       System.out.println("\nTry sum(2.3, 3.1)...");
       double x = sum(2.3, 3.1);
       System.out.println("Returns " + x);

       System.out.println("\nTry sum(2, 3.1)...");
       double y = sum(2, 3.1);
       System.out.println("Returns " + y);
    }
```

Java Program Structure: Class = Container

Computer Science

A class can also hold variables, which are called fields (go figure!), and can even hold other class definitions (called inner classes). We will focus on fields for now. Let's consider a class MyMath, which will provide some basic math functions:

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));

    }
}
```

```
> run MyMath
add(2,3) => 5.0
```

Java Program Structure: Class = Container

Suppose we add a method which calculates the $\log_2(..)$ of a double:

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static double log2(double x) {
        return Math.log(x) / Math.log(2.0);
    }

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(2,3) => " + log2(8.0));

    }
}
```

Recall:

$\log_A(B) * \log_B(C) = \log_A(C)$

so if A = *e* and B = 2:

$\log_2(C) = \log(C) / \log(2)$

```
> run MyMath
add(2,3) => 5.0
log2(8.0) => 3.0
```

Java Program Structure: Class = Container

But it is inefficient to calculate Math.log(2.0) each time, so we add it as a field to the container:

```
public class MyMath {

    static double logOfTwo = Math.log(2.0);

    static double add(double x, double y) {
        return (x + y);
    }

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(8.0));

    }
}
```

Java Program Structure: Class = Container

Since we can put it anywhere, we put it near its only use the program:

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(8.0));

    }
}
```

Java Program Structure: Class = Container

One more refinement: logOfTwo is actually being used as a constant value, which should never change: to make sure we don't change it, we make it final:

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(8.0));

    }
}
```

Java Program Structure: Class = Container

Computer Science

One more refinement: logOfTwo is actually being used as a constant value, which should never change: to make sure we don't change it, we make it final, so that if we accidentally try to modify it, we will get an error:

```
7
8     static final double logOfTwo = Math.log(2.0);
9
10    static double log2(double x) {
11        logOfTwo = 0.6931;
12        return Math.log(x) / logOfTwo;
13    }
```

Interactions | Console | Compiler Output | Find/Replace

```
1 error found:
File: /Users/waynesnyder/Dropbox/Documents/Teaching/CS 112/Lectures & Course
Materials/MyMath.java  [line: 11]
Error: /Users/waynesnyder/Dropbox/Documents/Teaching/CS 112/Lectures & Course
Materials/MyMath.java:11: cannot assign a value to final variable logOfTwo
```

Java Program Structure: Class = Container

Computer Science

Summary: a Java class is a container for methods (including main), fields, and final fields (constants). Fields can be initialized just like local variables, but final fields can not be modified after initialization:

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    static double  z = 8.0;    // just an example

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }
}
```

The entities declared in a class are called its members; for now we have:

- methods
- fields
- final fields (constants)

Java Program Structure: Scope of fields and methods

Scope of the members of a class: Since order does not matter, the scope of a method or a field is the entire class:

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    static double  z = 8.0;    // just an example

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }
}
```

Scope of all members

Java Program Structure: Scope of fields and methods

Scope of the members of a class: Since order does not matter, the scope of a method or a field is the entire class:

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    static double  z = 8.0;    // just an example

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }
}
```

Scope of add, logOfTwo, log2, z, and main.

Java Program Structure: Scope of fields and methods

Computer Science

Scope of the members of a class: Since order does not matter, the scope of a method or a field is the entire class:

```
> run MyMath
add(2,3) => 5.0
log2(8.0) => 3.0
```

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }

    static final double logOfTwo = Math.log(2.0);

    static double  z = 8.0;    // just an example
}
```

Scope of x, y

Scope of x

Scope of add, logOfTwo, log2, z, and main.

Java Program Structure: Scope of fields and methods

Computer Science

Scope of the members of a class: Since order does not matter, the scope of a method or a field is the entire class:

```
> run MyMath
add(2,3) => 5.0
log2(8.0) => 3.0
```

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    static double  z = log2(256.0);    // just an example

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }
}
```

The scope rule for members means you can call a method to initilize a field!

Java Program Structure: Class = container for fields and methods

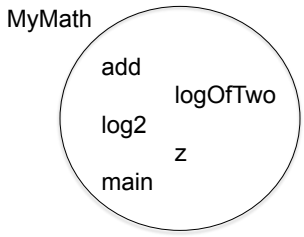
Computer Science

Summary:

A Java class is an unordered container for its members (methods and fields);

The scope of a member declaration is the entire class (more on this later), unlike local variables inside methods, whose scope is from the declaration to the next unmatched right curly brace;

Fields can be initialized just like local variables, even using methods in the class, but final fields can not be modified after initialization.



Java Program Structure: Static Containers

Computer Science

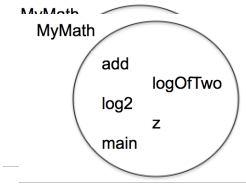
What about the keyword static?

There are two different ways classes can be used to compute:

The first is as a static container for its members.

When an entity is static it:

1. Is created when you first run the program;
2. Has a single instance which exists during the entire run of your program; and
3. Is destroyed only when your program terminates.

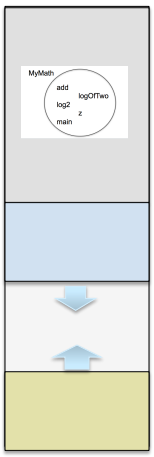


Java Program Structure: Static Containers

Computer Science

There is in fact a static region of your program in memory: this region contains all static members of classes; other regions of memory are dynamic and store values of local variables in method calls and entities created by new:

RAM



Static Region: holds your code and all static members of classes.

The Heap: holds entities created by new (e.g., arrays).

Free memory

Run-Time Stack: holds values of local variables during method calls

Static Memory: Only thing that can change here is values of fields.

Dynamic Memory: for anything else that changes during execution.

Java Program Structure: Static Containers

Computer Science

So we can think of a running program as existing in two different "worlds," static and dynamic:

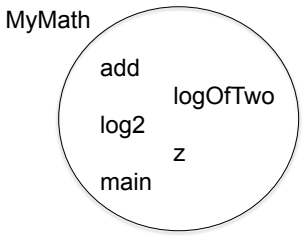
Static World | Dynamic World

Java Program Structure: Static Containers

Computer Science

When you run your program, a static instance of your class is created, exists for the entire run, and is destroyed only when your program terminates:

Static World | Dynamic World

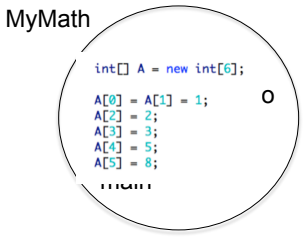


Java Program Structure: Static Containers

Computer Science

When you create a new entity using new, it exists in the Dynamic world; entities in the Static World are called Classes and entities in the Dynamic world are called Objects.

Static World | Dynamic World



The array A:

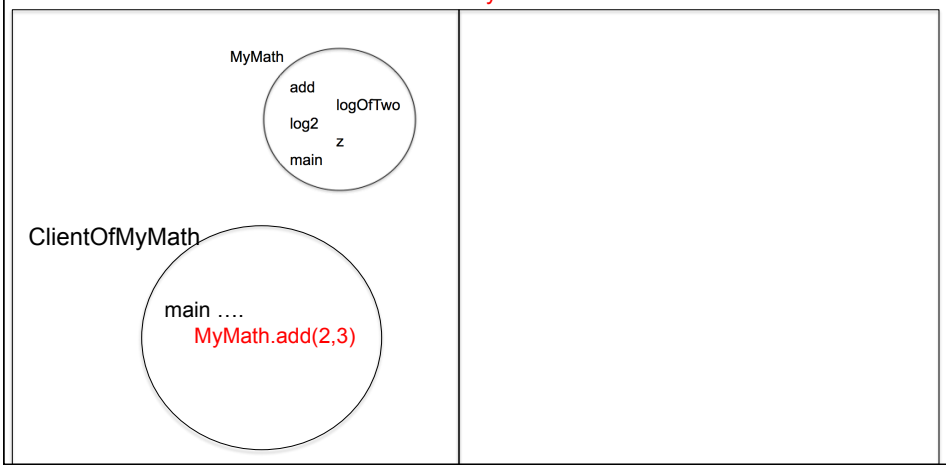
0:	1
1:	1
2:	2
3:	3
4:	5
5:	8

Java Program Structure: Programs spread over multiple files.

Computer Science

A Java class can be simply used as a Static container for its members, and used by other programs; this is a way of creating your own libraries (such as Math); just like the Math library, you refer to the methods using the name of the class:

Static World | Dynamic World



Java Program Structure: Programs spread over multiple files.

Computer Science

A Java class can be simply used as a Static container for its members, and used by other programs; this is a way of creating your own libraries (such as Math); just like the Math library, you refer to the methods using the name of the class, as long as both files are in the same folder/directory:

CS112Homework

MyMath.java

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    static double  z = 8.0;    // just an example

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }
}
```

ClientOfMyMath.java

```
public class ClientOfMyMath {

    public static void main(String[] args) {

        System.out.println("MyMath.add(2,3) => "
                           + MyMath.add(2,3));
        System.out.println("MyMath.log2(8.0) => "
                           + MyMath.log2(MyMath.z));
        System.out.println("MyMath.logOfTwo => "
                           + MyMath.logOfTwo);

    }
}
```

Java Program Structure: Programs spread over multiple files.

Computer Science

Note that you call your own static container library program just like you call other static libraries in Java (String, Character, Math): you use the name of the class plus a dot ".".

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    static double  z = 8.0;    // just an example

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }
}
```

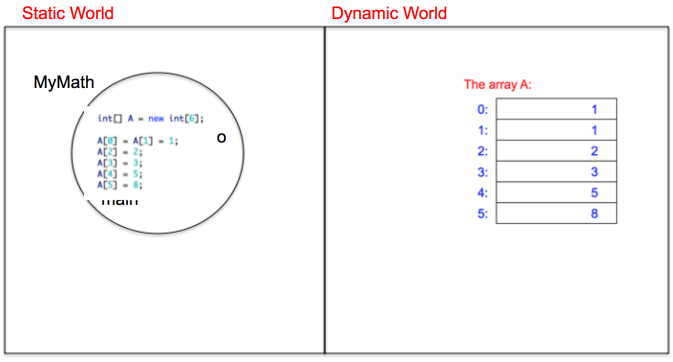
```
public class ClientOfMyMath {

    public static void main(String[] args) {

        System.out.println("MyMath.add(2,3) => "
                                + MyMath.add(2,3));
        System.out.println("MyMath.log2(8.0) => "
                                + MyMath.log2(MyMath.z));
        System.out.println("MyMath.logOfTwo => "
                                + MyMath.logOfTwo);

    }
}
```

Notice also that the library program still contains a main(…) method, and can still be run like a normal program. Usually, the main method of a library is used for testing code.

Java Program Structure: Creating and using Objects

Computer Science

Creating Objects: Recall that when we declare an array in a method, we are creating a new object that lives in dynamic memory:



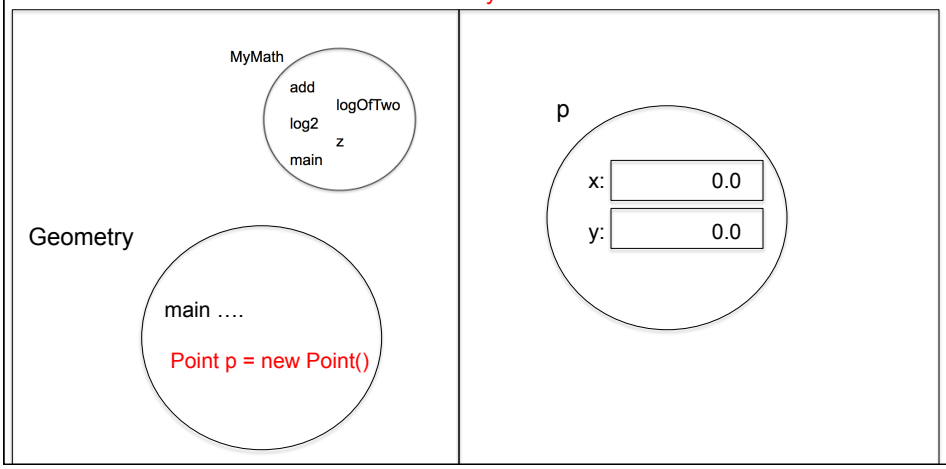
Java Program Structure: Creating and using Objects

Computer Science

But, we can create new "classes" which contain members just like static classes…..

Static World

Dynamic World



Java Program Structure: Creating and using Objects

Computer Science

But, we can create new "classes" which contain members just like static classes…..

```
> run Geometry
p =  (2.3,4.5)
```

CS112Homework

Geometry.java

```
public class Geometry {

    public static void main(String[] args) {

        Point p = new Point();
        p.x = 2.3;
        p.y = 4.5;

       System.out.println("p =  (" + p.x + "," + p.y + ")");

    }
}
```

MyMath.java

```
public class MyMath {

    static double add(double x, double y) {
        return (x + y);
    }

    static final double logOfTwo = Math.log(2.0);

    static double log2(double x) {
        return Math.log(x) / logOfTwo;
    }

    static double  z = 8.0;   // just an example

    public static void main(String[] args) {

       System.out.println("add(2,3) => " + add(2,3));
       System.out.println("log2(8.0) => " + log2(z));

    }
}
```

ClientOfMyMath.java

```
public class ClientOfMyMath {

    public static void main(String[] args) {

       System.out.println("MyMath.add(2,3) => "
                          + MyMath.add(2,3));
       System.out.println("MyMath.log2(8.0) => "
                          + MyMath.log2(MyMath.z));
       System.out.println("MyMath.logOfTwo => "
                          + MyMath.logOfTwo);

    }
}
```

Point.java

```
public class Point {

    double x = 0.0;         // create two doubles
    double y = 0.0;         // and initialize to 0.0
}
```

Java Program Structure: Creating and using Objects

Computer Science

But, we can create new "classes" which contain members just like static classes…..