CS 213 : Software Methodology

Spring 2019

Sesh Venugopal

Lecture 3: Jan 29

Design Aspects of Static Members - 1

Why Static?
Design Aspects

Static for Non Object-Oriented Programming

Suppose you want to write a program that just echoes whatever is typed in:

```
public class Echo {
    public static void main(String[] args)
    throws IOException {
        BufferedReader br = new BufferedReader(
                                   new InputStreamReader(System.in));
        System.out.print("> ");
        String line = br.readLine();
        System.out.println(line);
    }
}
```

This program works without having to create any Echo objects – the Virtual Machine executes the main method directly on the Echo class (not via an Echo object) because the main method is declared static

Calling the main method directly on the class makes it non object-oriented; object orientation implies that there is an object or an instance of which a field is accessed, or on which a method is executed

Static Methods for "Functions"

An extreme use of static methods is in the `java.lang.Math` class in which every single method is static – why?

```
public class Math {
   public static float abs(float a) {...}
   ...
   public static int max(int a, int b) {...}
   ...
   public static double sqrt(double a) {...}
   ...
}
```

The reason is that every method implements a mathematical function (i.e. a process with inputs and outputs), and once the function returns, there is nothing to be kept around (as in a field of an object) for later recall/use.
In other words there is no state to be maintained

The `Math` methods can be called directly on the class, for example:

```
double sqroot = Math.sqrt(35);
```

In fact, you CANNOT create an instance of the `Math` class - "instantiation" is not allowed

Static Fields for Constants

`Math` is a "utility" class, in which all methods are "utility" methods – the class is just an umbrella under which a whole lot of math functions are gathered together

Apart from the utility methods, the `Math` class also has two static fields to store the values for the constants `E` (natural log base e) and `PI` (for the constant pi)

```
public class Math {
   ...
   public static final double E ...
   public static final double PI ...
   ...
}
```

Again, these constants can be directly accessed (without objects):

```
double area = Math.PI * radius * radius;
```

`E` and `PI` are constants because their values cannot be changed (final)

~~`Math.PI = Math.PI * 2;`~~