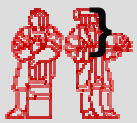
Lecture 10:Static & Final

Kenya 2005

MyMath Example

```
public class MyMath {

  public double PI = 3.14159;

  public double square (double x) {
    return x * x;
  }

  public static void main(String[ ] args) {
   MyMath m = new MyMath();
  System.out.println("m: value of PI is " +
  m.PI);
    System.out.println("m: square of 5 is " +
  m.square(5));

    MyMath n = new MyMath();
    System.out.println("n: value of PI is " +
  n.PI);
    System.out.println("n: square of 5 is " +
  n.square(5));
  }
}
```



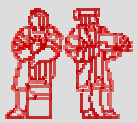
Objects Review

- In Example 1, to calculate the square of 5 we need to create an instance of MyMath class:

```
MyMath m = new MyMath();
```

- Then we invoke it's square() method with the argument 5:

```
m.square(5);
```



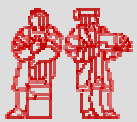
MyMath Output

- The results of invoking square() method on instances m and n are the same:

```
m: value of PI is 3.14159
m: square of 5 is 25
n: value of PI is 3.14159
n: square of 5 is 25
```

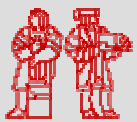
- `square()` behaves the same no matter which instance it is called on.

- So . . . why not have one `square()` method for the entire class?



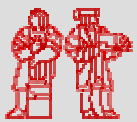
Also . . .

- The value of `PI = 3.14159` is the same for all instances of `MyMath` class.
- Why do we need to store a value of `PI` separately for each instance of `MyMath`?
- Instead, can we have only one common value of `PI` for the whole `MyMath` class?



MyMath with static

```
public class MyMath {

  // add keyword "static" to field declaration
  public static double PI = 3.14159;

  // add keyword "static" to method declaration
  public static double square (double x) {
    return x * x;
  }

  // main method is always declared "static"
  public static void main( String[ ] args) {
    // MyMath m = new MyMath(); - No longer need this
   line!
    // MyMath n = new MyMath(); - No longer need this
   line!

    // Now invoke square() method on the MyMath class
    System.out.println("Value of PI is " + MyMath.PI);
    System.out.println("Square of 5 is"  +
   MyMath.square(5));
  }
}
```

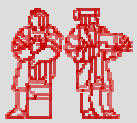


Static Pi Field

- We added word `static` to the declaration of the final variable `PI`:

```
public static double PI = 3.14159;
```

- It means that now we have only one value of variable PI for all instances of `MyMath` class; `PI` is now a *class* data field



The `final` keyword

- We declared `PI` as

```
public static double PI = 3.14159;
```

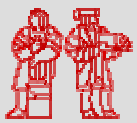
but this does not prevent changing its value:

```
MyMath.PI = 999999999;
```

- We use keyword **`final`** to denote a constant:

```
public static final double PI = 3.14159;
```

- Once we declare a variable to be **`final`**, it's value can no longer be changed!



Final References

- Consider this final reference to a `Point`:

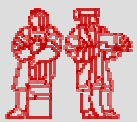
```
public static final Point ORIGIN =
                                new Point(0,0);
```

- This prevents changing the reference `ORIGIN`:

```
MyMath.ORIGIN = new Point(3, 4);
```

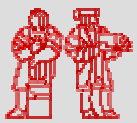
- BUT! You can still call methods on ORIGIN that change the state of ORIGIN.

```
MyMath.ORIGIN.setX(4);
```



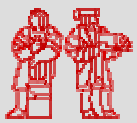
MyMath with static & final

```
public class MyMath {

  // add keyword final to field declaration
  public static final double PI = 3.14159;

  public static double square (double x) {
    return x * x;
  }

  public static void main( String[ ] args) {
    System.out.println("Value of PI is " +
                                MyMath.PI);
    System.out.println("Square of 5: " +
                                MyMath.square(5));
  }
}
```



Static Fields

- Only one instance of a static field data for the entire class, not one per object.
- "static" is a historic keyword from C/C++
- "Class fields" is a better term
  - As opposed to "instance fields"

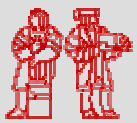


Static Square Method

- We also added the word "static" to the declaration of the method `square()`:

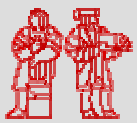
```
public static double square(double x) {
  return x * x;
}
```

- Now the method `square()` is shared by all instances of the class—only one `square` method for the class, not one for each instance.



Static Methods

- Static methods do not operate on a specific instance of their class
- Have access only to static fields and methods of the class
  - Cannot access non-static ones
- "Class methods" is a better term
  - As opposed to "instance methods"



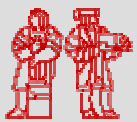
Java's Math Class

- Let's take a look at Java's Math class in the API

- You cannot create an instance of the `Math` Class, it's just a place to store useful static methods

- All the methods and fields are static:

```
Math.sqrt(16)
Math.PI
Math.abs(-3)
```



Static Field Examples

- Constants used by a class
  (usually used with `final` keyword

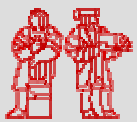
- We only need to have one per class; don't need one in each object- Look at the following declaration:

```
public static final double TEMP_CONVERT= 1.8;
```

  - If the variable TEMP_CONVERT is in class Temperature, it is invoked by:

```
double t = Temperature.TEMP_CONVERT * temp;
```

  - Constants are all capital letters by tradition (C, C++) - for example: PI , TEMP_CONVERT etc.



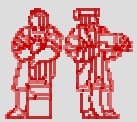
Static Method Examples

- For methods that use only the arguments and therefore do not operate on an instance

```
public static double pow(double b, double p)
// Math class, takes b to the p power
```

- For methods that only need static data fields

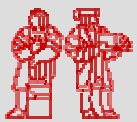
- We use the static key word on the main method in the class that starts the program
  - No objects exist yet for the main method to operate on!



POP QUIZ

Should it be static or non-static?

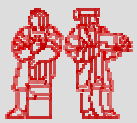
- Speed of light field
- `getName()` method in a Person class
- A `sum` method that returns the resulting of adding both its arguments
- Width data field in a Rectangle class



POP QUIZ

Should it be static or non-static?

- A field that stores the speed of light **static**
- `getName()` method in a Person class **non**
- A `sum` method that returns the resulting of adding both its arguments **static**
- Width data field in a Rectangle class **non**



EC.S01 Internet Technology in Local and Global Communities
Spring 2005-Summer 2005