# Lecture March 14th - Object-Orientated Programming

Bentley James Oakes

March 14, 2018

# Midterm

Thanks for making the midterm go smoothly

- Should be graded by the weekend
- I'll talk about averages next week
- And maybe discuss some of the harder questions

# This Lecture

Section 1

# Recap

# Recap

What did we do right before the break?

- Started talking about using objects
- Showed examples of `Scanner` and `Random`

# Scanner

- Let's look at the **Scanner** object
- This is an **object** that has methods to get input from the user
- Need to write `import java.util.Scanner;` at top of the class

```
7           //create a variable of type Scanner
8           Scanner scan = new Scanner(System.in);
9
10          //get a number from the user
11          int x = scan.nextInt();
12          System.out.println("You entered: " + x);
13
14          //close the scanner after using it
15          scan.close();
```

Interactions | Console | Compiler Output

```
123
You entered: 123
```

# Hello Program

```
public static void sayHello(){

    //create a Scanner variable
    Scanner scan = new Scanner(System.in);

    System.out.println("What is your name and favourite number?");
    String name = scan.next();
    int num = scan.nextInt();

    System.out.println("Hello " + name + "!");
    System.out.println("I also love the number " + num + "!");

    System.out.println("Would you like a nice message?");

    boolean niceMessage = scan.nextBoolean();
    if (niceMessage){
        System.out.println("You are a wonderful person!");
    }

    scan.close();
}
```

```
> run ScannerExample
What is your name and favourite number?
Bentley 8
Hello Bentley!
I also love the number 8!
Would you like a nice message?
```

# Random Recap

- Can use the `Random` class to get a sequence of random numbers
- Seeding a pseudo-random number generator allows us to get the same sequence every time

```
int seed = 123;//the seed is any number
Random randNumGen = new Random(seed);
for (int i=0; i < 10; i++){
    int randNum = 1 + randNumGen.nextInt(100);
    System.out.println("Seeded rand num: " + randNum);
    //always gives 83, 51, 77, 90, 96, 58, 35...
}
```

Note that the seed isn't the first random number

Section 2

# Object-Oriented Programming

# Object-Oriented Programming

- The `Scanner` and `Random` classes allow you to create *instances* and call methods on those *instances*
- *Object-oriented programming* is about creating and using different *classes* and *instances*
- A *class* is a collection of related methods and attributes
- An *instance* is creating a concrete example for a class
  - For example, there can be many instances of the `Student` class

# Object-Oriented Programming

- The purpose of *object-orientated programming* is to divide up your code into different components
- As we've seen, splitting up your code makes it easier to think about
- Should re-use components in future programs

# Math Class

For instance, the *Math* class holds:

**Methods:**

`pow(double a, double b)`

`random()`

**Attributes:**

`Math.PI` is a double, with value 3.141592653589793

```
System.out.println("This is Pi.");
System.out.println(Math.PI);
//3.141592653589793

System.out.println("This is Pi squared.");
System.out.println(Math.pow(Math.PI, 2));
//9.869604401089358
```

# Roadmap

In this next part of the course, we'll talk about:

- Creating our own classes with methods and attributes
- The difference between accessing a class, and an instance of that class
- How to control how methods are called
- How to create and print instances

Section 3

# Writing Classes

# Writing our Own Math Class

- As practice, let's create the `MyMath` class
- For now, we'll just have an add method in there

```
public class MyMath{
    public static double add(double a, double b){
        return a + b;
    }
}
```

# Calling This Method

- Now we can call this method from another class
- **Note that these classes must be in two different files in the same directory**

```
public class MyMath{
    public static double add(double a, double b){
        return a + b;
    }
}
public class MathExample{
    public static void main(String[] args){
        double c = MyMath.add(14.0, 57.6);
        System.out.println(c); //71.6
    }
}
```

# Calling This Method

- To call a method in a different class, we type the class name, a period, and then the name of the method
- Again, these classes must be in the same directory on your computer

```
public class MyMath{
    public static double add(double a, double b){
        return a + b;
    }
}
public class MathExample{
    public static void main(String[] args){
        double c = MyMath.add(14.0, 57.6);
        System.out.println(c); //71.6
    }
}
```

# Adding a Class Attribute

- Let's store the value of Pi in our class
- This will be a *class attribute*, same as `PI` was for the `Math` class
- Note that the class attributes are defined within the class, above the methods

```
public class MyMath{

    //a class attribute
    //type: double
    //name: MyPi
    //value: 3.14
    public static double myPi = 3.14;

    public static double add(double a, double b){
        return a + b;
    }
```

# Accessing a Class Attribute

- We access it by writing `MyMath.myPI`

```
public class MathExample{
    public static void main(String[] args){
        double c = MyMath.add(14.0, 57.6);
        System.out.println(c); //71.6

        System.out.println(MyMath.myPi);
        //3.14
    }
}
```

# Default Values

- If you don't provide a value to a class attribute, a default value will be given
- For primitive types: 0 or false
- For reference types: `null`

# Adding a Class Method

- Let's add a method to multiply the parameter by Pi in our class
- This will be a *class method*, just like `Math.pow`
- We access it by writing `MyMath.multiplyByMyPi`
- This is very similar to what we've been writing this whole time
- We're just accessing a method in another class

```
//this is in the MyMath class
public static double multiplyByMyPi(double num){
    return num * myPi;
}
//this is in the MathExample class
double result = MyMath.multiplyByMyPi(134);
System.out.println(result); //420.76
```

# Conclusion

- Class attributes
  - Attributes belonging to the class
  - Example: pi belongs to the Math class
- Class methods
  - Methods in a class
  - Can access attributes in that class or other classes

Section 4

# Instances of Classes

# Instances

- We've seen classes as a collection of methods and attributes
- The attribute belongs to the *class*
- Let's store information about *individual* students
- For example:
  - A student named *Bentley* with a grade of 99
  - A student named *Melanie* with a grade of 100

# Student Class

- We'll create a class to define a `Student` object
- Let's store two things for now: the student's name and their grade

```
public class Student{

    public String name;
    public int grade;
}
```

- Notice that we don't have the keyword `static` anymore
- This is because we want the name and grade to be different for each student

# Instances of a Class

How do we create a `Student`?

```
Student s = new Student();
```

- We're creating a new *instance* of the `Student` class
- Just like `Random rand = new java.util.Random();`
- Or int[] arr = new int[5];

The new keyword is used to create an *instance* of a class

# Instance Attributes

- Let's go back to arrays for a second
- int[] arr = new int[5];
- We access `arr`'s length by writing `arr.length`
  - Variable name DOT attribute
- We are accessing the `length` attribute of that instance

# Instance Attributes

Let's access the name and grade of a `Student`

```
public class TestingStudents{
    public static void main(String[] args){
        Student s = new Student();
        s.name = "Bentley";
        s.grade = 99;
        System.out.println("Student Name: " + s.name);
        System.out.println("Student Grade: " + s.grade);
    }
}
```

# Different Instances

Let's create two different students and set their names and grades
And then print out the names and grades

```
Student s = new Student();
s.name = "Bentley";
s.grade = 99;
System.out.println("First student Name: " + s.name);
System.out.println("First student Grade: " + s.grade);

Student t = new Student();
t.name = "Melanie";
t.grade = 100;
System.out.println("Second student Name: " + t.name);
System.out.println("Second student Grade: " + t.grade);
```

```
First student Name: Bentley
First student Grade: 99
Second student Name: Melanie
Second student Grade: 100
```

These are different instances (representing different students), so they have different values for their attributes

# Printing a Student

- Can we print these instances out?
- `System.out.println(‘‘Student: ’’ + s);?`
- Hint: A `Student` is a reference type
- We get the class name and the address

```
System.out.println("Student s: " + s);
//Student s: Student@54c59e1c
```

# Reference Types

Recall that any instance is a reference type
This means we have to be careful
What does this print?

```
Student a = new Student();
a.grade = 100;
System.out.println("A's grade: " + a.grade);

Student b = a;
b.grade = 50;
System.out.println("A's grade: " + a.grade);
```

*A's grade: 100*
*A's grade: 50*
Why? Variable *b* points to the same address as *a*

# Instances Summary

- We create different instances of a class using the *new* keyword
- Each instance of a class can have different attribute values

Section 5

# Static vs Non-Static Attributes

# Static vs Non-static

We can see that attributes like name, and grade should be **different** for every student
What if we wanted attributes that had the **same value** for every student?

A **static** attribute has the same value for every instance of the class
The variable belongs to the class itself

# Passing Student

Let's add a `static` variable to the `Student` class
Let's say that if a student has a grade less than 50, they get a warning email

```
public class Student{

    public String name;
    public int grade;

    public static int gradeThreshold = 50;
}
```

# Static Attributes

Let's print out the value of this attribute for two students

```
Student c = new Student();
System.out.println(c.gradeThreshold); //50
Student d = new Student();
System.out.println(d.gradeThreshold); //50
System.out.println(Student.gradeThreshold); //50
```

This gives the same value

In fact, the usual way to access this attribute is
`Student.gradeThreshold`
Remember, this attribute belongs to the `Student` class, just like
`MyMath.MyPi`

# Changing Static Attributes

Let's change the value of this attribute

```
Student e = new Student();
System.out.println(e.gradeThreshold); //50
Student f = new Student();
f.gradeThreshold = 75;
System.out.println(e.gradeThreshold); //75
System.out.println(Student.gradeThreshold); //75
```

This changes the grade threshold

Why? The attribute is being stored once in the class

# Non-static Attributes

So what's a *non-static class attribute*?

This is a variable like:
`public int grade;`

The value of this attribute is different for each student

Doesn't make sense to write `Student.grade = 76;`

# Static vs Non-Static Attribute Summary

| **Static** | **Non-Static** |
|---|---|
| Belongs to the class | Belongs to an instance |
| Same value for all instances of the class | Potentially different value |
| **Example:** | |
| Num. of lectures in the course | Num. of lectures the student has attended |
| **Access Example:** | |
| `Student.numTotalLectures` | `Student s = new Student();`<br>`s.numLecturesAttended++;` |

# Examples

Let's create a `Person` class. Should the following attribute be static or non-static?

- Name
  - Non-static (different for every `Person`)
- Age
  - Non-static (different for every `Person`)
- Scientific Name (Example: *homo sapiens*)
  - Static (every `Person` is a *homo sapien*)
  - But this value could change in the future
    - `Static` doesn't mean non-changing!

In most cases, your attribute will be non-static.
Can each instance have a different value, or do they all have the same value?

Section 6

# Static vs Non-Static Methods

# Static vs Non-Static Methods

We also have static and non-static methods as well

Can be summarized as:

*Non-static methods can access static and non-static attributes/methods in the class*

*Static methods can only access static attributes/methods in the class*

# Person Example

Let's make a small example to show this

The `Person` class has a first name and a scientific name of *homo sapiens*

```
public class Person{

    public String firstName;
    public static String scientificName = "homo sapiens";
}
public class TestingPersons{
    public static void main(String[] args){

        Person p = new Person();
        p.firstName = "Bob";

        System.out.println(p.firstName);
        //bob

        System.out.println(p.scientificName);
        //homo sapiens
    }
```

# Static Person Method

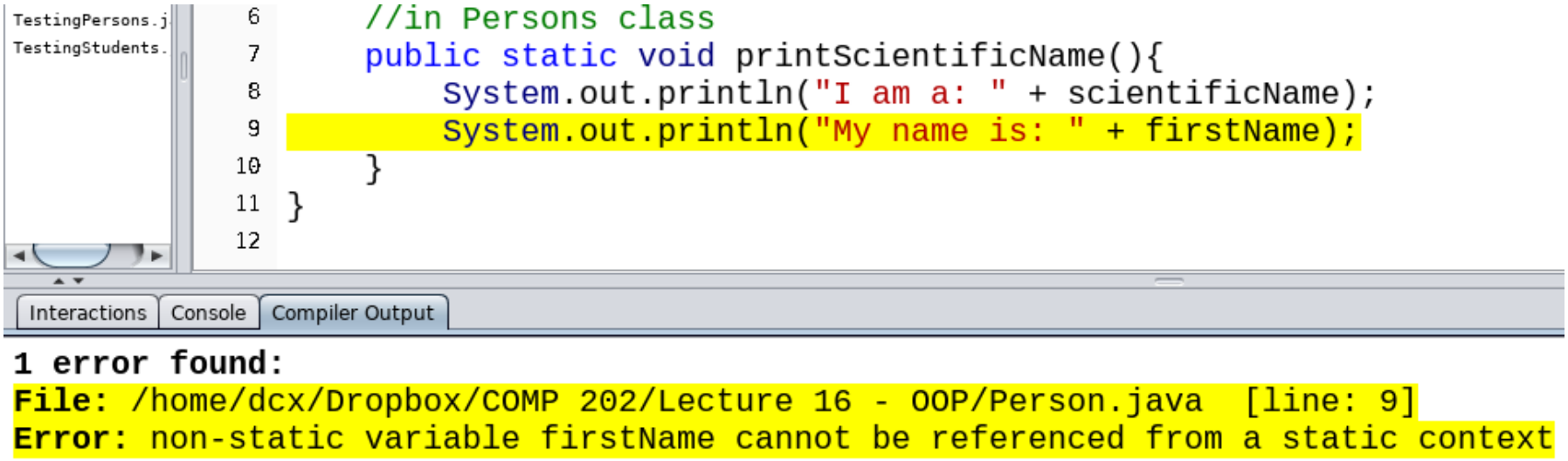
Let's write a method that prints out the scientific name

```
//in Persons class
public static void printScientificName(){
    System.out.println("I am a: " + scientificName);
}
```

```
//in TestingPersons main method
Person p = new Person();
p.firstName = "Bob";

System.out.println(p.firstName);
//Bob

p.printScientificName();
//I am a: homo sapiens

Person.printScientificName();
//I am a: homo sapiens
```

# Accessing Non-Static Attribute

What if `printScientificName` (a static method) tried to print out `firstName` (a non-static variable)?




Not everybody has the same first name

**Static methods can only access static attributes**

# Creating a Non-Static Method

- Let's create the `printFirstName()` method
- Note that the method has the header: `public void printFirstName()`
- Note that we don't add the word static in here
- This is therefore a **non-static** method

```
//in Persons class
public void printFirstName(){
    System.out.println("My name is: " + firstName);
}
```

# Calling a Non-Static Method

```
Person a = new Person();
a.firstName = "Alice";

Person b = new Person();
b.firstName = "Bob";

a.printFirstName();
//My name is: Alice

b.printFirstName();
//My name is: Bob
```

The non-static methods can access the non-static `firstName` attribute

# Static vs Non-Static Decision

- First decide which attributes are non-static:
  - Does each instance has a different value for this attribute?
- Then decide if the method will access a non-static variable or method
  - If the method accesses a non-static variable or method, it must be non-static

# Static vs Non-Static Examples

For a `Book` class:

Non-static attributes:

- Title
- Author
- Number of pages

Static attributes:

- Threshold for being a long book (for example, 500 pages)

Non-static methods;

- printSummary - title, author, numPages
- getTitle - return the title of the book
- isBookLong - check to see if the current book is longer than the threshold

Static methods:

- getThreshold - return the threshold
- getDescription - Return a description of what a book is

# Summary

| | |
|---|---|
| **Static attributes:** | Common value for all instances of that class, stored in the class |
| **Non-static attributes:** | Different values for all instances of that class, stored in the instance |
| **Static methods:** | Can only access static attributes/methods |
| **Non-static methods:** | Can access static or non-static attributes/methods |