Lecture March 19 - Access Control and Constructors

Bentley James Oakes

March 19, 2018

Assignment 4

- Assignment 4 was posted before reading week
- Due Tuesday, March 27nd
- A large assignment because of examples

- The first two questions deal with two-dimensional arrays
- The second question is on creating your own classes
  - You'll have the material you need after today

Midterm Stats

Statistics for the midterm:

- Overall Average: 66%
- Short Answer Average: 66%
- Long Answer Average: 85%
- More statistics available on myCourses

Midterm viewing sessions will be announced later

This Lecture

Section 1

Recap

Attributes versus Variables

- What's the difference between *attributes* in a class and *variables* in a method?

	Location	**Default Vals.**	**Scope**
Local Variables	Inside methods	No	Within block
Attributes	Outside methods	Yes	Within class

Scope Differences

Let's see the scope for local variables and attributes

```
public class LVEx{

    public static int num = 54;

    public static int add(int x, int y){
        int z = x + y;
        if (z > 0){
            int v = 10;
            z = v;
        }
        return z + num;
    }
```

The local variables exist only within their scope
Attributes exist everywhere within the class

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();` `s.numLecturesAttended++;`

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
```

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

Section 2

Instances as Parameters

Instances as Parameters

It's very common to pass an instance to a method

For example, let's write a method to compare two students, and return the student with a higher grade

compareStudents Static

```
//in Student class
public static Student compareStudents(Student first, Student second){
    if (first.grade > second.grade){
        return first;
    }else{
        return second;
    }
}

//in TestingStudents' main method
Student s = new Student();
s.name = "Bentley";
s.grade = 99;

Student t = new Student();
t.name = "Melanie";
t.grade = 100;

Student higherGrade = Student.compareStudents(s, t);
System.out.println("Better student: " + higherGrade.name);
//Better student: Melanie
```

Clarification

- A static method can access non-static methods and attributes **if the method has access to an instance**

```
//in Student class
public static Student compareStudents(Student first, Student second){
    if (first.grade > second.grade){
        return first;
    }else{
        return second;
    }
}
```

- The static method must access the `grade` attribute of an instance
- You can't just write `grade` within this method as it's a non-static attribute

Section 3

this

this

- In a non-static method, we might need to access the current instance
- For example, let's create a non-static version of `compareWith`
- This method will be called on a `Student` instance
- And it might have to return the current instance

compareWith

Let's write a non-static version of this method

```
//in Student class
public Student compareWith(Student other){
    if (this.grade > other.grade){
        return this;
    }else{
        return other;
    }
}

Student higherGrade = s.compareWith(t);
System.out.println("Better student: " + higherGrade.name);
//Better student: Melanie
```

Using this

Let's look at the static and non-static versions

```
//in Student class
public static Student compareStudents(Student first, Student second){
    if (first.grade > second.grade){
        return first;
    }else{
        return second;
    }
}
```

```
//in Student class
public Student compareWith(Student other){
    if (this.grade > other.grade){
        return this;
    }else{
        return other;
    }
}
```

The `this` keyword is taking the place of the `first` parameter

That's because `this` refers to the current instance

this Example

```
//in Student class
public Student compareWith(Student other){
    if (this.grade > other.grade){
        return this;
    }else{
        return other;
    }
}
```

We can only use `this` in a non-static method

It doesn't make sense to use `this` in a static method,
There's no current instance when you call the method!

Another this Example

```
//static method
public static void printStudent(Student s){
   System.out.println("Printing student: " + s.name);
   System.out.println("Grade is: " + s.grade);
}

//non-static method
public void print(){
    printStudent(this);
}
```

- Here we have a static method for printing the student's details
- It takes a `Student` as input and accesses the student's details

```
Student.printStudent(s);
//Printing student: Bentley
//Grade is: 99
s.print();
//Printing student: Bentley
//Grade is: 99
```

Section 4

Book Class Example

Book Class

```
public class Book{

    //non-static variables
    //different for every instance of Book
    public String title;
    public String author;
    public int numPages;

    //static variable
    //same values for all books
    //(A book is long when it is over
    //this many pages)
    public static final int LONG_THRESHOLD=700;
```

Let's have a `Book` class with some attributes

The Main Method

```
//main method within the book class
public static void main(String[] args){
    //create an instance of a book
    Book hp = new Book();
    hp.title = "Harry Potter";
    System.out.println("Title: " + hp.title);
    System.out.println("Threshold: " + Book.LONG_THRESHOLD);
}
```

Create a instance and print attributes

this keyword

```java
//non-static method
//uses this to refer to the current instance
public void printNumPages(){
    System.out.println("Num pages: " + this.numPages);
}

//main method within the book class
public static void main(String[] args){
    //create an instance of a book
    Book hp = new Book();
    hp.title = "Harry Potter";
    hp.numPages = 340;
    System.out.println("Title: " + hp.title);
    hp.printNumPages(); //Num pages: 340
}
```

Calling a non-static method, which uses the `this keyword`

compareBooks

```
//main method within the book class
public static void main(String[] args){
    Book hp = new Book();
    hp.title = "Harry Potter";
    hp.numPages = 340;

    Book hro = new Book();
    hro.title = "Hunt for Red October";
    hro.numPages = 548;

    compareBooks(hp, hro);
    //Hunt for Red October is longer.
}

//static method to compare two books
public static void compareBooks(Book a, Book b){
    //notice that a static method can only access
    //a non-static variable if there's an
    //instance to access it on
    if (a.numPages > b.numPages){
        System.out.println(a.title + " is longer.");
    }else{
        System.out.println(b.title + " is longer.");
    }
}
```

Section 5

Getters and Setters

Creating an Instance

Recall how to create instances of a class and set attributes on those instances

```
//main method within the book class
public static void main(String[] args){
    Book hp = new Book();
    hp.title = "Harry Potter";
    hp.numPages = 340;

    Book hro = new Book();
    hro.title = "Hunt for Red October";
    hro.numPages = 548;
```

Setters

Instead of accessing the attributes of an instance directly,
Let's use a method with error checking

```
public void setNumPages(int newNumPages){
    if (newNumPages <= 0){
        String err = "Invalid num pages";
        throw new IllegalArgumentException(err);
    }else{
        this.numPages = newNumPages;
    }
}

//main method within the book class
public static void main(String[] args){
    Book hp = new Book();
    hp.title = "Harry Potter";
    hp.setNumPages(340);
    hp.printNumPages();
```

Getters

We also have methods to return the value of an attribute

```
//a getter method
public int getNumPages(){
    return numPages;
}

//a more secure getter method
public int getNumPagesPassword(String password){
    if (password.equals("prettyplease")){
        return numPages;
    }else{
        return -100;
    }
}
```

Getters and Setter

We call these patterns *Getters and Setters*
Here are the very basic versions:

```
//a setter method
public void setNumPages(int newNumPages){
    this.numPages = newNumPages;
}

//a getter method
public int getNumPages(){
    return numPages;
}
```

Section 6

Public and Private

Preventing Access

Maybe someone's code is giving our books a length of 0

How can we force them to use our error-checking `setNumPages` method?

```
public class Book{

    //non-static variables
    //different for every instance of Book
    public String title;
    public String author;
    public int numPages;

public class BadBookCode{

    public static void main(String[] args){

        Book got = new Book();
        got.numPages = 0;
        System.out.println(got.numPages);
    }
}
```

Preventing Access

If attributes or methods are *private*, they can't be accessed outside the class

```
public class Book{

    //non-static variables
    //different for every instance of Book
    public String title;
    public String author;
    private int numPages;
```

BadBookCode.java
Book.java
MidtermLongAnswer.java

```
1  public class BadBookCode{
2
3      public static void main(String[] args){
4
5          Book got = new Book();
6          got.numPages = 0;
7          System.out.println(got.numPages);
8      }
9  }
```

Interactions | Console | Compiler Output

```
2 errors found:
File: /home/dcx/Dropbox/COMP 202/Lecture 17 - More OOP/BadBookCode.java  [line: 6]
Error: numPages has private access in Book
File: /home/dcx/Dropbox/COMP 202/Lecture 17 - More OOP/BadBookCode.java  [line: 7]
Error: numPages has private access in Book
```

Why Use Private?

- In the real world, you will not write every class or be able to change the source code The *private* keyword lets you control how others use the attributes and methods in your class

Private Attributes and Methods

- **Attributes:** Other code may get the value of an attribute, but not set it
  - Example: the `.length()` method gets a `String`'s length, but no method allows you to change it
- **Methods:** Might have helper methods that shouldn't be used outside the class
  - Example: In Assignment 3, the `getNextPoint` method only makes sense to be used within the `Mountains` class.

Abstraction

- **Abstraction:** We should hide details, both to reduce complexity, and to allow changes to our code
  - Example: We don't know how a `String` is stored internally. We just call the public methods `length, charAt`;

When to Use Private

General rules:

- Make all attributes in your class private if you can
- Write getters and setters if other classes need to access these attributes
- Make methods private, unless other classes need to access them

Constructor Example

```
public class Book{

    private String title;
    private String author;
    private int numPages;

    public void setNumPages(int newNumPages){
        this.numPages = newNumPages;
    }
    public int getNumPages(){
        return numPages;
    }

    public void setAuthor(String newAuthor){
        this.author = newAuthor;
    }
    public String getAuthor(){
        return author;
    }
```

Section 7

Final Keyword

The final keyword

Another access modifier: the `final` keyword
If we create a `final` attribute, its value can never be changed after it has been initialized

```
public static final int PASSING_GRADE = 60;
```

We call `static final` variables *constants*. We name them with all uppercase, with underscores (_) between words.

Cute Cats

Book.java
Cat.java
MidtermLongAnswer.java

```
1  public class Cat{
2
3      public static final boolean CATS_ARE_CUTE = true;
4
5      public static void main(String[] args){
6
7          System.out.println("Cats are cute: " + Cat.CATS_ARE_CUTE);
8          Cat.CATS_ARE_CUTE = false;
9      }
```

Interactions | Console | Compiler Output

```
1 error found:
File: /home/dcx/Dropbox/COMP 202/Lecture 17 - More OOP/Cat.java  [line: 8]
Error: cannot assign a value to final variable CATS_ARE_CUTE
```

Section 8

Constructors

Constructors

Constructors are executed when a new instance of a class is created.
Constructors can be used to set attribute values

Constructor Example

```
//constructor for the Person class
//set the name and age
public Person(String name, int birthYear){
    this.name = name;
    this.age = 2017 - birthYear;
}
```

- Name of the constructor method must be the same as the name of the class
- No return type **(not even void!)**
- Non-static method

Constructor Usage

```
public class Person{

    //can be accessed within this class
    private String name;
    private int age;

    //constructor for the Person class
    //set the name and age
    public Person(String name, int birthYear){
        this.name = name;
        this.age = 2017 - birthYear;
    }

    public static void main(String[] args){
        Person p = new Person("Bob", 1945);

        System.out.println(p.name + " is " + p.age);
        //Bob is 72
    }
}
```

Default Constructor

If you do not write a constructor, the default constructor for a `Person` class looks like:

```
public Person(){
}
```

If you write your own constructor, this will overwrite the default constructor!
As in, you will then have to use the new constructor to create an instance

Section 9

Overloading

Overloading

Might want two different methods in the same class to have the same name
But have different parameters

For example:

- Changing a method's algorithm depending on the types
- Different constructors

Overloading Example

```
public class OverloadingExample{
    public static int max(int a, int b){
        if (a > b){
            return a;
        }else{
            return b;
        }
    }
    public static int max(int a, int b, int c){
        if (a > b && a > c){
            return a;
        }else if (b > a && b > c){
            return b;
        }else{
            return c;
        }
    }
    public static void main(String[] args){
        System.out.println("Max: " + max(1, 2));
        System.out.println("Max: " + max(1, 5, 2));
    }
}
```

Another Overloading Example

`public void print(boolean b)`: Prints `boolean` value `b`.
`public void print(double d)`: Prints `double` value `d`.
`public void print(int i)`: Prints `int` value `i`.
`public void print(Object o)`: Prints `Object` `o`.
`public void print(String s)`: Prints `String` `s`.
`public void println()`: Terminates the current line by writing the line separator string.
`public void println(boolean b)`: Prints `boolean` value `b` and then terminates the line.
`public void println(double d)`: Prints `double` value `d` and then terminates the line.
`public void println(int i)`: Prints `int` value `i` and then terminates the line.
`public void println(Object o)`: Prints `Object` `o` and then terminates the line.
`public void println(String s)`: Prints `String` `s` and then terminates the line.

```
System.out.println(true);
System.out.println(1);
System.out.println(56.7);
System.out.println("Hello!");
System.out.println('a');
```

Overloading Details

- Java allows overloading based on the changes in method **parameters** (# and type)
- So `public static int add(int i, int j)` and `public static double add(double a, double b)` are okay
  - Java automatically figures out when to call the int version, and when to call the double version
- Changing the return type and static/non-static of a method doesn't allow overloading
  - Java can't tell which version of the method to call
  - `public int add(int a, int b)` and `public String add(int i, int j)` are called the same way, so this isn't allowed

Overloading for Constructors

We've already seen overloading for constructors...

```
Random rng1 = new Random();
Random rng2 = new Random(123);
```

We have two ways to create a `Random` instance: with a seed, and without a seed

Overloading for Constructors

```
public class Painting{
    public String artist;
    public double value; //in millions

    //constructor if we know the artist and value
    public Painting(String artist, double value){
        this.artist = artist;
        this.value = value;
    }

    //constructor if we don't know the artist
    public Painting(double value){
        this.artist = "Unknown";
        this.value = value;
    }

    public static void main(String[] args){
        Painting starryNight = new Painting("Van Gogh", 6.2);
        Painting sunset = new Painting(1);

        System.out.println("Artist: " + starryNight.artist);
        System.out.println("Value: " + starryNight.value);
```

Section 10

toString()

toString() method

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

- When we print out an instance, we get its address
- This is because the `println` method actually calls the `toString` method on the instance
- The default code for the `toString` method is to print out the class name and address

The toString method

Returns a `String` when the instance is passed to a print method
Must have the header: `public String toString()`

```
public String toString(){
    return "Artist: " + this.artist + " Value: " + this.value;
}

//main method in Painting class
public static void main(String[] args){
    Painting starryNight = new Painting("Van Gogh", 6.2);
    Painting sunset = new Painting(1);

    System.out.println(starryNight);
    System.out.println(sunset);

    //before adding toString method
    //Painting@27dbfe7e
    //Painting@53ecb0f7

    //after adding toString method
    //Artist: Van Gogh Value: 6.2
    //Artist: Unknown Value: 1.0
}
```

Section 11

Storing Instances

Arrays of Objects

Let's look at how to store instances in an array

It's very similar to `Strings`

```
public static Painting maxValue(Painting[] pArr){
    Painting bestPainting = pArr[0];
    for (int i=1; i < pArr.length; i++){
        if (pArr[i].value > bestPainting.value){
            bestPainting = pArr[i];
        }
    }
    return bestPainting;
}

public static void main(String[] args){
    Painting starryNight = new Painting("Van Gogh", 6.2);
    Painting sunset = new Painting(1);
    Painting nighthawks = new Painting("Hopper", 4);

    Painting[] collection = {starryNight, sunset, nighthawks};
    Painting mostExpensive = maxValue(collection);
    System.out.println("Most expensive worth: " + mostExpensive.value + " million");
}
```

Section 12

Exercise

Exercise

- Write a class that describes a cat. A cat should have a name and an age
  - Should these be public or private? Static or non-static?
- Write two possible constructors (overloaded). Both of them take the name of the cat as input. One of them creates a newborn kitten with age 0.0. The other creates a cat with the age as a parameter to the constructor
- Write a `meow()` method: If the cat has age less than 1.0, the method prints the name of the cat, plus "mews", otherwise it prints the name of the cat, plus "meows"
- Write a `birthday()` method that increments the age of the cat by 1.0, and prints out "Happy birthday to ", plus the name of the cat