

Lecture March 21 - Advanced OOP

Bentley James Oakes

March 20, 2018

- Assignment 4 due date changed
- Now due **Saturday, March 31st**

This Lecture

- 1 Recap
- 2 this
- 3 Getters/Setters and Public/Private
- 4 Final Keyword
- 5 Constructors
- 6 Overloading
- 7 toString()
- 8 Storing Instances
- 9 Person Example
- 10 Apartment Example
- 11 Adding a Player

Section 1

Recap

Section 2

this

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 compareTo(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 3

Getters/Setters and Public/Private

Getters and Setters patterns
Here are the very basic versions:

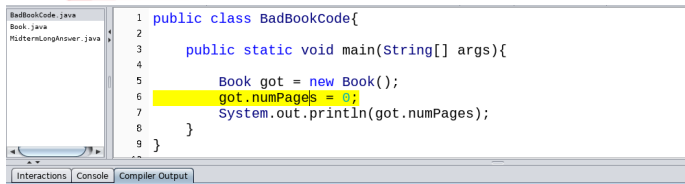
```
//a setter method  
public void setNumPages(int newNumPages){  
    this.numPages = newNumPages;  
}
```

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

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;  
}
```



```
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 }
```

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

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

Public/Private 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 4

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     }
10 }
```

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 5

Constructors

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

```
//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
    }
}
```

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 6

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

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

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

public static void main(String[] args){

    System.out.println(add(12, 15)); //27

    System.out.println(add(15.6, 34.8)); //50.4
}
```

Overloading Not Working

- 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

```
22     public static int multiply(int x){
23         return x * 3;
24     }
25
26     public static double multiply(int y){
27         return y * 2;
28     }
29
30     public static void main(String[] args){
31
32         int a = multiply(4);
33         double b = multiply(6);
34     }
```

Interactions Console Compiler Output

1 error found:

File: /home/dcx/Dropbox/COMP 202/Lecture 17 - More OOP/Code/OverloadingExample
[line: 26]

Error: method multiply(int) is already defined in class OverloadingExample

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 7

toString()

```
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 8

Storing Instances

Arrays of Objects

Let's look at how to store instances in an array
It's very similar to Strings

```
String[] arr = new String[3];

System.out.println("New str arr: " + Arrays.toString(arr));
//New str arr: [null, null, null]

arr[0] = "Hello";

System.out.println("Str arr: " + Arrays.toString(arr));
//Str arr: [Hello, null, null]
```

Arrays of Paintings

Let's put a Painting in an array

And use Arrays.toString()

Calls toString on each element in the array

```
Painting[] pArr = new Painting[3];
System.out.println("New painting arr: " + Arrays.toString(pArr));
//New painting arr: [null, null, null]

pArr[1] = new Painting("Picasso", 10);

System.out.println("Painting arr: " + Arrays.toString(pArr));
//Painting arr: [null, Artist: Picasso Value: 10.0, null]
```

Arrays of Paintings

- Here three Paintings are placed in an array
- Then we search the array to find the Painting with the highest value

```
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 9

Person Example

Person Example

- Let's create the Person class
- Where each person has a best friend
- Therefore, each instance of the Person class can store the address of another Person instance

Person Start

Here are the attributes for the person's name and their best friend (another Person)

```
public class Person{

    //store a person's name and
    //their best friend
    private String name;
    private Person bestFriend;

    //constructor that only takes a name
    //note that the bestFriend attribute
    //has a default null value
    public Person(String name){
        this.name = name;
    }
}
```

Person Other Constructor

We also have another constructor that takes a bff as well. This uses the `setBestFriend` method.

Note that the `setBestFriend` method isn't needed, but it can be handy to have once place where

```
//take a name and the best friend
public Person(String name, Person bff){
    this.name = name;

    //make sure they are each
    //other's best friend
    this.setBestFriend(bff);
    bff.setBestFriend(this);
}

//setter for the best friend
public void setBestFriend(Person bff){
    this.bestFriend = bff;
}
```



```
//print out a person's name
//and their best friend's name too
public String toString(){
    String s = "";
    s += "Name: " + this.name;
    s += " Best friend: ";

    //make sure to check that there's
    //a best friend first
    if (bestFriend == null){
        s += "None :(";
    }else{
        s += this.bestFriend.getName();
    }
    return s;
}
```

```
//the main method
public static void main(String[] args){

    Person b = new Person("Bob");
    System.out.println(b);
    //Name: Bob Best friend: None :(

    Person s = new Person("Sally", b);
    System.out.println(b);
    //Name: Bob Best friend: Sally
    System.out.println(s);
    //Name: Sally Best friend: Bob
}
```

Section 10

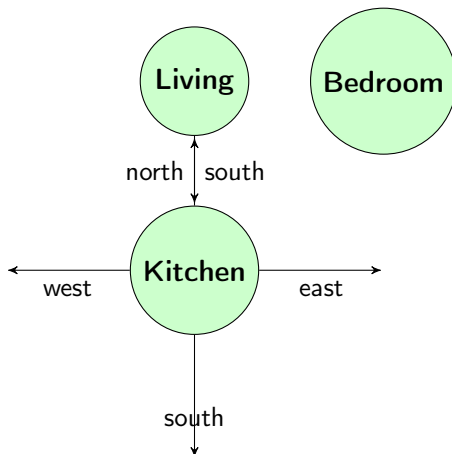
Apartment Example

Connected Rooms Example

- We're going to represent an apartment with a few rooms
- We'll have an `Apartment` class with `Rooms` connected to each other
 - For example, the living room will be to the north of the kitchen
- We'll store the length and width for each room, so that we can calculate the apartment's total area

Room Connections

- We are going to connect instances of these rooms together.
- For example, we might place a living room to the north of the kitchen.
- In this case, we must store the address of the living room in the kitchen, and vice versa.



Room Start

Here we define the room name, width and length, as well as the Room constructor.

```
public class Room{
    //name of the room
    private String name;

    //private dimensions
    private double width;
    private double length;

    //this constructor takes a room name,
    //the width and the length
    public Room(String name, double width, double length){
        this.name = name;
        this.width = width;
        this.length = length;
    }

    //return the name of the room
    public String getName(){
        return name;
    }
}
```

This method returns the area of a room.

```
//returns the area of the room
public double getArea(){
    return width * length;
}
```

- For your future programs, think about the units
- Is the area in metres? feet? yards?
- Need to write documentation so everyone uses the same units

In the Room class, have attributes for the other rooms connected to this Room instance

```
//connections to other rooms
//default value is null
private Room north;
private Room east;
private Room south;
private Room west;
```


We also need methods to set the Rooms in the four directions

```
//set the north of this room
//and set the south attribute of the other room
public void setNorth(Room other){
    this.north = other;
    other.south = this;
}

public void setEast(Room other){
    this.east = other;
    other.west = this;
}
```

Room toString

The toString method prints a few details about the room. As well, if there is a room in a direction (as in the room is not null), then that room's name is printed

```
public String toString()
{
    String s = "";
    s += "Room: " + name + " Area: " + getArea();

    //check to see if there's a room to the north
    //if so, print north's name
    if (north != null){
        s += "\nRoom to the north: " + north.name;
    }
    if (east != null){
        s += "\nRoom to the east: " + east.name;
    }
    if (south != null){
        s += "\nRoom to the south: " + south.name;
    }
    if (west != null){
        s += "\nRoom to the west: " + west.name;
    }
    return s;
}
```

In a main method, let's start by creating three rooms

```
public static void main(String[] args)
{
    //create a room named kitchen
    Room kitchen = new Room("Kitchen", 10, 12);
    System.out.println("Area of the kitchen: " + kitchen.getArea());

    //create more rooms
    Room living = new Room("Living", 20, 40);
    Room bedroom = new Room("Bedroom", 100, 3000);
}
```

Next, let's connect the rooms together

And to test, print out the kitchen

```
kitchen.setNorth(living);  
living.setEast-bedroom);  
  
System.out.println(kitchen);
```

Room: Kitchen Area: 120.0

Room to the north: Living

Placing Rooms in an Array

These rooms can then be placed in an array. We then iterate through the array and print out each room. Make sure to check to see if an entry in the array is *null*

```
//an array that stores rooms
Room[] rooms = new Room[4];
rooms[0] = kitchen;
rooms[1] = living;
rooms[2] = bedroom;

//print out the details of all rooms
for (int i = 0; i < rooms.length; i++)
{
    Room r = rooms[i];
    if (r != null){
        System.out.println(r);
    }
}
```

Apartment Area

It is also easy to iterate through the array and calculate the total area for all the rooms

```
//sum up the area for all rooms
double apartmentArea = 0;
for (int i = 0; i < rooms.length; i++)
{
    Room r = rooms[i];
    if (r != null){
        apartmentArea += r.getArea();
    }
}
System.out.println("Total square metres: " + apartmentArea);
```

Section 11

Adding a Player

- To make our apartment more interesting, let's add a player to walk around
- The Player will have a current room
- The user will be able to input commands using a Scanner
- This will be very similar to Assignment 5

Player Start

Let's have the Player with a name and their current room

```
import java.util.Scanner;

public class Player
{

    //simple class with a name and currentRoom
    private String name;
    private Room currentRoom;

    public Player(String name, Room currentRoom)
    {
        this.name = name;
        this.currentRoom = currentRoom;
    }
}
```

Player Move

Let's add methods to travel between rooms

We'll need getter methods in our Room class to access north, south, east, west

```
//get the room that is to the north of the currentRoom
//then set our currentRoom to be the northRoom
public void goNorth()
{
    Room northRoom = currentRoom.getNorth();
    this.currentRoom = northRoom;
}
|
//this method is safer than goNorth()
//as first we check to make sure that we
//are not travelling to a null room
public void goSouth()
{
    if (currentRoom.getSouth() == null)
    {
        System.out.println("You can't go south");
    }
    else
    {
        Room southRoom = currentRoom.getSouth();
        this.currentRoom = southRoom;
    }
}
```

Let's add a look command, to see which room we're currently in

```
//print the name of the currentRoom
//note that this assumes that the currentRoom is never null
//(which might not be a correct assumption)
public void look()
{
    System.out.println("I am in the " + currentRoom.getName());
    System.out.println(currentRoom);
}
```

Player Input

A non-static method within the Player class that asks for commands, and executes them

```
public void getInput(){
    //set up a scanner that gets user input
    Scanner sc = new Scanner(System.in);
    String input = "";
    //keep looping until quit is entered
    while (!input.equals("quit"))
    {
        System.out.println("Enter a command");
        input = sc.nextLine();
        //call various methods based on the input
        //we have a game now!
        if (input.equals("look")){
            this.look();
        }
        else if (input.equals("north")){
            this.goNorth();
        }
        else if (input.equals("south")){
            this.goSouth();
        }
        else{
            System.out.println("That is not a command.");
        }
    }
}
```

Ways to Extend This

- Many ways to extend this example
- Making a game is a great way to learn
- Things to try:
 - Add items to the room that you can pick up and use
 - Example: Read and write notes that can be dropped in each room
 - Have a monster to fight, where both characters deal randomly-calculated damage
 - Have the player head through a maze, where they have to visit different locations before they win