Classes and Objects

Constructors, attributes, and other details
Goal

Will explore a number of advanced features of classes.

- Constructors, class variables, etc.
- Features will enhance ability to create useful objects.
- These are OO features!
- Need these features later when doing OO design.
**Definition:** Classes are outlines/templates/blueprints/cookie-cutters for creating an object.

- Every house blueprint can be used to create many houses.

- Every cookie-cutter can be used to create zillions of individual cookies.
  - May only be one cookie.
  - May be used to make two cookies.
  - May make zillions.
  - No requirements on number.

- Each class can be used to create zero, one, or multiple objects.
Sure, consider the main class.

```java
public class NoInstantiationExample{
    public static void main(String[] args){
        System.out.println("Yo, dude.");
    }
}
```

No instantiation, but still worthwhile.
- This class is not used to create an object.
public class OneInstantiationExample {
    public static void main(String[] args) {
        Random myRandom = new Random();
        if (myRandom.nextInt(100) > 10) {
            System.out.println("Yo, dude.");
        }
    }
}

Class called Random used to create one object called myRandom.
The “new” Keyword

- To instantiate, use “new”.
  - “Use the blueprint to make me one of them there thingies.”
  - So for every “new” there is an object!

- How many objects?

```java
public static void main(String[] args)
{
    Dog[] puppies = new Dog[20];
    for(int i = 0; i <=10; i++)
    {
        puppies[i] = new Dog();
    }
}
```

Made 12 objects: 1 array, plus 11 more puppies in the array.

Not 21 because didn’t instantiate every element of the array.
public class Garbage {

    public static void main(String[] args) {
        Yak y = new Yak();
        Garbage g = new Garbage();
        g.printGarbage();
    }

    public void printGarbage() {
        System.out.println("sjfdafdsf");
    }
}
Class “Data” or “Attributes”

- House blueprint describes all the features or “data” of a soon-to-be-built home.
  - number of doors
  - number of rooms
  - number of lights
  - etc.

- Classes do the same.
  - variable for numberOfDoors
  - variable for numberOfRooms
  - variable for numberOfLights
  - etc.

We can fill these variables with any values that we want the house to have.
Instance or Class Variables

- Data is held in variables that are defined in the class, NOT in a method.
  - So the whole object has access to the variable, and not just some method.

```java
class House {
    public int numberOfDoors = 3;
    public int numberOfRooms = 8;
    public int numberOfLights = 8;
}
```

This is a perfectly valid class!

Also called **fields**, **class members**, and **instance variables**.
Instance Variables Assigned Later

- Don’t have to give values to variables.
  - Not all houses have 8 rooms.
  - Could be assigned later!

```java
public class House {
    public int numberOfDoors;
    public int numberOfRooms;
    public int numberOfLights;
}
```

```java
public class Example {
    public static void main(String[] args) {
        House daveHouse = new House();
        daveHouse.numberOfRooms = 4;
        daveHouse.numberOfDoors = 3;
        House tracyHouse = new House();
        tracyHouse.numberOfRooms = 7;
    }
}
```

The variables are public, so accessed them and changed their values.
Warning

Some compilers don’t like you to leave the instance variables un-instantiated and unassigned.

- If so, give them default, trivial values.
  - e.g., numberOfDoors = 0;
  - e.g., Yak y = null;

Can always assign a null to an object.

- Random myRandomGenerator = null;
- Person bob = null;
Most people do not like to call these class variables.

- Implies they belong to the class.
- In fact, they belong to the object.
- So *instance variables* is a better name!

- I don’t care, as long as you understand the difference.

• BTW, later will learn to create “static” variables that belong to the class.
3rd Warning

- We are using **public** instance (class) variables.

- Later we will see that **private** instance variables are better.
  - Stay tuned.
  - We’ll stick with public for now.
**4th Warning**

- **Instance (class) variables can be put outside of all methods.**
  - e.g., in the class, but outside of methods

- **You cannot put any other code outside of methods!**
  - All other code (like “if” statements, etc.) must be inside of methods.
    - (Well, ok, there is this thing called a “static block”, but it’s a rare exception that you don’t know about yet.)
Create a class that describes attributes of a person.
- List relevant data (e.g., age).
- Create class with those variables.
- Use the class to create two objects.
  - Assign some values.
  - Test to see if the two objects have the same values (e.g., for age).
**Class Behavior or “Actions”**

- Our house blueprint may indicate that the house can do things (actions).
  - heat up
  - cool down
  - detect intruder
  - fill with poisonous gas

- Classes do the same.
  - method for heatUp()
  - method for coolDown()
  - method for detectIntruder()
  - method for fillWithPoisonousGas()
Adding Action Methods

```java
public class House {
    public int numberOfDoors;
    public int numberOfRooms;
    public int numberOfLights;

    /** turns on heater */
    public void heatUp() {
        ...
    }

    /** opens gas valve */
    public void fillWithPoisonousGas() {
        ...
    }
}
```

- Now can create a house object.
  - Specify number of rooms, doors, etc.
  - Make it heat up, cool down, etc.

- My house may have **different number of rooms** than yours.

- But our houses will **both heat up, cool down**, etc.
Methods Can See Class Variables

Class variables are accessible to all the class’ methods.
- even if they are private!

Exercise: can you make the heating only come on if your house has 3 rooms?
- Hint: don’t forget to check for null!
Create a class for the clock in the room.

- Choose a few instance variables.
  - e.g., numberOfSeconds
- Choose a few actions/methods.
  - e.g., advanceMinuteHand()
  - e.g., soundAlarm()

Can you make the minute hand advance only when the number of seconds is 60?
Consider Person. *Every* time create a person object, we set the name.

- Person dan = new Person();
  dan.name = “Daniel”;

Wouldn’t it be nicer to set it when instantiating?

- Person dan = new Person(“Daniel”);
- Called a constructor!
**Definition:** A constructor is code that gets run when the object is created.

- Looks just like a method.
  - code is inside this special “method”.

- Like methods, can run *any* code at all.
  - Could call your grandmother and nuke popcorn.

- Typically used to fill in some class variables.
  - Person joe = new Person("Joseph", 22, 67);
Constructor Example

Constructor “method” always has the same name as the class.

```java
public class Person {
    public int age;
    public int height;
    public String name;

    public Person(String personsName) {
        name = personsName;
    }
}
```

No return type!
i.e., no void, int, char, double…
Constructor Example (cont.)

- Now we can use the constructor as
  
  Person john = new Person("Cool Dude");

- The object “john” goes by the name “Cool Dude”.

- What would happen if the constructor was private?
public class LightBulb
{
    public int wattage = 60;
    public String brand = "GE";

    public LightBulb(int bulbWattage, String bulbBrand)
    {
        wattage = bulbWattage;
        brand = bulbBrand;
    }
}
Your Turn (Constructors)

- Create a Clock class
  - Give it attributes.
  - Give it a constructor.
  - Use the constructor when creating the object.

- Create a class for something else in the room.
  - ditto
Multiple Constructors

- Sometimes I want to specify
  - the name.
  - the name and age.
  - the name, age, and height.
  - don’t know which one in advance.

- So create multiple constructors!
  - just like multiple methods.
  - called overloading.
    - (We’ll talk about this more later.)
Multiple Constructors Example

```java
public class Person {
    public int age;
    public int height;
    public String name;

    public Person(String personsName) {
        name = personsName;
    }

    public Person(String personsName, int personAge) {
        name = personsName;
        age = personAge;
    }

    public Person(String personsName, int personAge, int personHeight) {
        name = personsName;
        age = personAge;
        height = personHeight;
    }
}
```

Note: multiple constructors are NOT required.

Just create the ones you think you (or another developer) might need.
Default Constructors

What happens if just use “new Person()”? 
- **Error:** doesn’t exist.

So why do we always see instantiations like
Randon r = new Random()?

1. If no constructor specified, then “empty” one is the default!

2. We could also add our own default.

```java
public class Person {
    public Person() {
        //do nothing (or nuke popcorn, or whatever)
    }
    ...
}
```
Adding Code to Constructors

**Remember, constructors can do anything.**

```java
public class Person {
    public String name;

    public Person() {
        System.out.println("What, the other constructors" +
                          "aren’t good enough for you?");
    }

    public Person(String personName) {
        if(personName.equals("dave")) {
            System.out.println("I laugh at you.");
        }
        name = personName;
    }
}
```
An Annoyance

- Having to create “personsName” is annoying.
  - But can’t use “name” – which one is the local variable?

```java
public class Person {
    public int age;
    public int height;
    public String name;

    public Person(String name) {
        name = name; // WON'T WORK!
    }
}
```

- “this” keyword to the rescue…
“this” as the Object

- Suppose you create a Person object.
  - Person wu = new Person(“Lulu”);

- Can access a public instance variable with
  - wu.age
  - wu.name
  - etc.

- What if you are inside the Person class itself?
  - usually just say “name”, “age”
  - e.g., look at constructors where we used “name = personName”.
“this” as the Object (cont. 1)

- But we could use the “.” notation.
  - `wu.name` outside the class
  - `this.name` inside the class!

- Keyword “this” refers to the object that was instantiated from the class.
  - see next example for clarification
public class Person
{
    public int age;
    public int height;
    public String name;

    public Person(String personsName)
    {
        this.name = personsName;
    }

    ...

    Who gives a rat’s ass?

    Same as just “name = personsName;”
"this" as the Object (cont. 3)

Can simplify annoying problem of needing two variable names.
- personsName, personsAge, personsHeight

```java
public class Person {
    public int age;
    public int height;
    public String name;

    public Person(String name, int age, int height) {
        this.name = name;
        this.age = age;
        this.height = height;
    }

    ...  
}
```

instance variables: age, height, name

local constructor variables: name, age, height
public class Person
{
    public int age;
    public int height;
    public String name;

    public Person(String name)
    {
        this.name = name;
    }

    public void increaseAge()
    {
        this.age += 1;
    }

    public void changeName(String name)
    {
        this.name = name;
    }
}
“this” as the Constructor

- Sometimes want to call one constructor from another.
  - Use “this(arg1, arg2, arg3, …)”. It will grab whatever constructor has that many arguments.
  - The arguments must be of the correct type and in that same order.
public class Person {
    public int age;
    public int height;
    public String name;

    public Person(String name) {
        this(name, 0);
    }

    public Person(String name, int age) {
        this(name, age, 0);
    }

    public Person(String name, int age, int height) {
        this.name = name;
        this.age = age;
        this.height = height;
    }
}

Why do this?
1. Don’t have to rewrite the same code.
2. Easy way to assign default values.
“this” Constructor Warning

- If you use “this(…)” to call another constructor, it must be the first line of your constructor!
  - Compile error otherwise.
Your Turn (‘‘this’’)

Write class for something in the room.

- Give at least two attributes.
- Create multiple constructors.
- Use this keyword for
  1. object
  2. constructor