ICS 111
Introduction to Computer Science I

Nikki Manuel
Leeward Community College

Objects & Classes

Week 12
Fall 2019

Object-Oriented Programming

Object-Oriented Programming

So far, we have learned how to program with: loops, arrays, methods.

But to create larger and more complex software systems, we need to learn object-oriented programming.

- We view our program as made up of objects.
- We think about: what objects we want to manipulate, and how these objects relate to each other

Object-Oriented Programming (OOP)

Programming with objects : we've been doing this all along!

When we use the word **`new`** we are allocating memory for an object:

```
    Scanner myScanner = new Scanner(System.in);
```

Objects

What is an object?

An **object** represents something in the real world that can be distinctly identified.

For example: a student, a desk, a circle, a button, a midterm 😊

An object has characteristics and behaviors

- Characteristics -> instance variables, data fields
- Behaviors -> methods

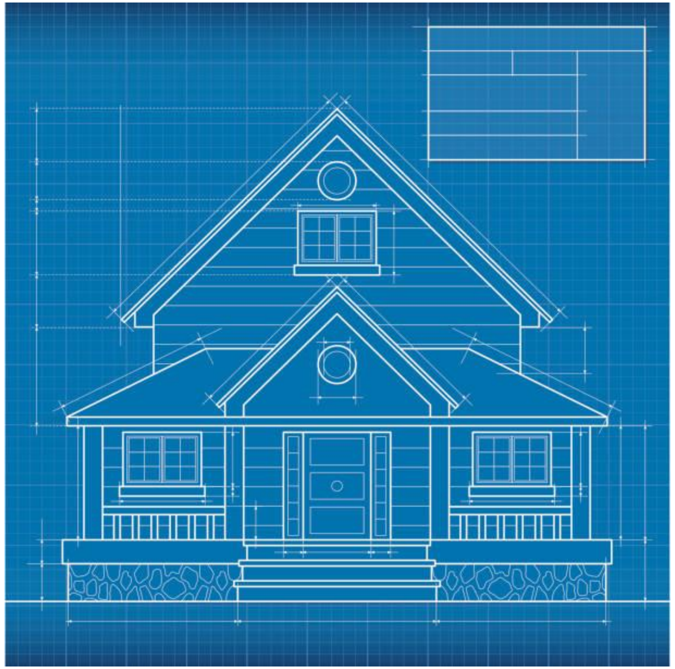
Creating an object

Objects of the same type are defined by a common class

To create an object, we need a **class**, which acts as a template or a blueprint to create the object

The object is an **instance** of a class

Class vs. Object


Class

Object

Creating an object : First we create a class

A class will have 3 things:

1. Instance Variables
   - Variables that describe the object
   - What differentiates one object of the same class from another?
2. Constructor
   - Creates a new object
   - What information is needed to create an object?
3. Methods
   - What can the object do?

The `Person` Object

A person could have these characteristics:

```
name (String)
address (String)
SSN (int or String)
income (double)
credit score (int)
medical record number (long)
GPA (double)
```

The list can go on forever

- You must choose which are necessary for your program

The `Person` Object

As a Leeward CC student:

```
name (String)
address (String)
SSN (int or String)
income (double)
credit score (int)
medical record number (long)
GPA (double)
```

The `Person` Object

As a taxpayer:

```
name (String)
address (String)
SSN (int or String)
income (double)
credit score (int)
medical record number (long)
GPA (double)
```

The `Person` Object

As a home-buyer:

```
name (String)
address (String)
SSN (int or String)
income (double)
credit score (int)
medical record number (long)
GPA (double)
```

The `Person` Object

As a hospital patient:

```
name (String)
address (String)
SSN (int or String)
income (double)
credit score (int)
medical record number (long)
GPA (double)
```

The `Person` Object : Selecting Characteristics

Select only those that are necessary for your program

If you try to store too much things, you waste memory and code that may not be used

Even after you define all the characteristics, you may still need to add/remove

Instance Variables

Instance variables are the characteristics of an object

Each object of the same class has its own set of instance variables

- – The Patient object
  - - Each patient has a set of name, address, and medical record number variables
  - - Each name, address, and medical record number are unique for each patient
- – The values of these variables differentiates one object of the same class from another

The Constructor

The **constructor** defines what information is needed to create an object.

- Some objects require additional information, while some objects don't:
  - `new String("Shamwow!")`

    Creates a String with value Shamwow!
  - `new Scanner(source)`

    Needs to know what/where to read
  - `new Random()`

    Doesn't require additional information

The Constructor

The constructor looks like a method, but with no return type

- NOT EVEN `void`

Used together with the `new` keyword when called

Must have exactly the same name as the object's class

We've used the constructor each time we create a new object

Calling a Method

Syntax: `object.method(arguments)`

`object`: Must have created the object first

`.` (dot): Associates the method with the object

- The method belonging to the object

`method`: The action you want to do

`arguments`: Extra info the method needs to work

- Can have zero or more

Creating a Class: Instance Variables Constructor

Let’s Create a Class!

I want to model/represent students in code : create the **`Student`** object

I’ll use the following characteristics:

- **`String name`**
- **`int idNum`**
- **`double studentGPA`**

These characteristics will become instance variables

First, we’ll create the class

- We create objects from classes

The `Student` class

```
public class Student {

    public Student(String name, int idNum,
                   double studentGPA) {

    }

}
```

The `Student` class : Class/Constructor Name

```
public class Student {

    public Student(String name, int idNum,
                                    double studentGPA) {

    }

}
```

The name of the class and the constructor must be the same.

The `Student` class : Parameters

```
public class Student {

    public Student(String name, int idNum,
                             double studentGPA) {

    }

}
```

These are the constructor's parameters. This information is needed to create a Student object.

The `Student` class : Instance Variables

```
public class Student {

    String name = "";
    int idNum = 0;
    double studentGPA = -1;

    public Student(String name, int idNum,
                                         double studentGPA) {
        // constructor code goes in here
    }
}
```

These are the instance variables

We need to assign the information from the constructor to the corresponding instance variables. Remember variable scoping.

What is `this` ?

We will use the `this` keyword to help us distinguish between the information in the constructor vs. the instance variables

- Since they have the same name

`this` will refer to the instance variables

Let’s see how it’s used!

Parameters to Instance Variables

```java
public class Student {

    String name = "";
    int idNum = 0;
    double studentGPA = -1;

    public Student(String name, int idNum, double studentGPA) {
        this.name = name;
        this.idNum = idNum;
        this.studentGPA = studentGPA;
    }
}
```

Parameters to Instance Variables

```
public class Student {

    String name = "";
    int idNum = 0;
    double studentGPA = -1;

    public Student(String name, int idNum, double studentGPA) {
        this.name = name;
        this.idNum = idNum;
        this.studentGPA = studentGPA;
    }
}
```

Parameters to Instance Variables

```
public class Student {

    String name = "";
    int idNum = 0;
    double studentGPA = -1;

    public Student(String name, int idNum, double studentGPA) {
        this.name = name;
        this.idNum = idNum;
        this.studentGPA = studentGPA;
    }
}
```

Parameters to Instance Variables

```java
public class Student {

    String name = "";
    int idNum = 0;
    double studentGPA = -1;

    public Student(String name, int idNum, double studentGPA) {
        this.name = name;
        this.idNum = idNum;
        this.studentGPA = studentGPA;
    }
}
```

The `Student` class

Mainly used to represent a student -- that's it!

It is not meant to be run

- There is no main method

- Just like the Scanner
  - The Scanner class is not run,
    but you create Scanner objects

We will need a "driver" class to run a program

A “driver” class

A class that IS meant to be run

- There is a main method

Example: I want to make an object and print its information

```
public class School {
    public static void main(String[] args) {
        Student ashK = new Student("Ash K.", 1234, 1.0);
        System.out.println(ashK);
```

Arguments vs. Parameters

arguments : information passed in a method call

```
Student ashK = new Student("Ash K.", 1234, 1.0);
```

– Ash K., 1234, and 1.0 are arguments

parameters : exist in the class definition

```
public Student(String name, int idNum, double
               studentGPA) {
}
```

The `toString()` Method

Returns a String that is meant to be printed
- You want to return a meaningful message that is meant to be printed

Must be called toString and must return a String
- When you print an object, Java will automatically look for this method

Defined in the object class you are making

Called in the "driver" class

The toString() Method

```
public String toString() {
    String output = "";

    output += "Student name:" + this.name;
    output += "\nID: " + this.idNum;
    output += "\nGPA: " + this.studentGPA;

    return output;
}
```

Creating a Class: Methods

Methods

A group of instructions that accomplish a task

- Allows you to reuse code
- Breaks up your program into pieces

Named after the action they perform

- Typically a verb

Be sure to comment what each method does

- Similar to commenting a class

Commenting a Method

Every method should be preceded with a descriptive comment using Javadoc documentation comments.

The comment should describe the method's purpose and use Javadoc tags to comment parameters, return types, and any exceptions.

Use a blank * to separate the description from the Javadoc tags.

The @return is omitted if there is no return value.

Commenting a Method – Example

```java
/**
 * Adds all the numbers within a given range.
 *
 * @param num1 the beginning of the range
 * @param num2 the end of the range
 * @return the sum of the numbers in the given range
 * @exception none
 */
public static int exampleSumMethod(int num1, int num2) {}
```

Method Header Syntax – Access Modifier

```
public static void main(String[] args)
```

public : Can be called from anywhere

– Within the same, or from other, programs

private : Used only within the class file it was defined in

– A public method may call a private method to do an additional task

protected : Via inheritance

Method Header Syntax – Requires an Object?

```
public static void main(String[] args)
```

static : Can call the method w/out creating an object of the class

– The method is associated with the class, not a specific instance (object) of that class

non-static : Requires an object to use it

– Non-static is not a keyword! If it's non-static, then simply do not write the word **`static`**

Method Header Syntax – Return Value Type

```
public static void main(String[] args)
```

void : Doesn't return anything

non-void : The method will return something when called

– Can only return ONE thing: object, array, char

– Non-void is not a keyword! You need to specify the data type of whatever is being returned by the method

Method Header Syntax – Method Name

```
public static void main(String[] args)
```

Methods begin with lowercase letters

Do NOT use the word main other than for running your program

– It is a special name Java looks for to run your program

Use a verb, or a combination of words that implies an action

Method Header Syntax – Parameters

```
public static void main(String[] args)
```

Every method name is followed by parenthesis ( )

- args are the program arguments
  - Also called the Command Line Arguments
  - jGRASP > Build > Run Arguments

Parameter vs. Argument

- Parameters are used when defining methods
- Arguments are used when calling methods

The `Student` Class

Used to create Student objects

Instance variables:

```
String name
int idNum
double studentGPA
```

Methods:

```
toString()
```

Instance Variables + Constructor

```
public class Student {

    String name = "";
    int idNum = 0;
    double studentGPA = -1;

    public Student(String name, int idNum, double studentGPA) {
        this.name = name;
        this.idNum = idNum;
        this.studentGPA = studentGPA;
    }
}
```

Instance Variables + Constructor

```
public class Student {

    private String name = "";
    private int idNum = 0;
    private double studentGPA = -1;

    public Student(String name, int idNum, double studentGPA) {
        this.name = name;
        this.idNum = idNum;
        this.studentGPA = studentGPA;
    }
}
```

Using `private` for Instance Variables

Enforce data encapsulation

- Data hiding

To create more secure code

- Use the private keyword on all instance variables

You will not be able to change instance variables unless:

- Within the class itself
- Create a helper method to do so

Rule of thumb : Always have instance variables as private

Accessor Methods aka “Get” Methods

Accessor methods simply return the instance variable

Since we use `private` on the instance variables we cannot just call them by `objectName.variable`

But we still need a way to get that information from outside of the class

Adding Accessor Methods to `Student` Class

```
public String getName() {
    return this.name;
}

public int getId() {
    return this.idNum;
}

public double getGpa() {
    return this.studentGPA;
}
```

Mutator Methods aka “Set” Methods

Allows us to modify instance variables

- Changes the instance variable to what is passed to the method
- BUT, you may have to do some validation before changing the instance variable

Again, since we are using **`private`**, we cannot easily access/modify them

Adding Accessor Methods to `Student` Class

```
public void setName(String newName) {
    this.name = newName;
}

public void setGpa(double newGpa) {
    this.studentGPA = newGpa;
}

public void setId(int newId) {
    this.idNum = newId;
}
```

The `Student` Class skeleton so far...

```
public class ClassName {

    /** Instance variables */

    /** Constructor */

    /** Mutator method(s) */

    /** Accessor method(s) */

    /** toString() method */
}
```