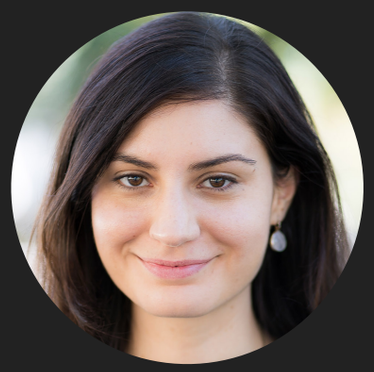
FUNDAMENTALS

CS62

DATA STRUCTURES AND ADVANCED PROGRAMMING

3: Classes and Objects



Alexandra Papoutsaki
she/her/hers

Lecture 3: Classes and Objects

- Classes and Objects

Some slides adopted from Princeton C0S226 course, Algorithms, 4th Edition, Oracle, and w3schools tutorials

A hypothetical scenario

- We want to write a program for the Office of Registrar to organize information about Pomona students.
- Let's think of what information we would need about a Pomona student. E.g.,:
  - Name
  - Email
  - Pomona ID
  - The year they entered Pomona
  - Academic standing
  - Classes they are currently enrolled in
  - How many credits they have taken so far
  - Have they graduated
  - Etc.

What can we do so far?

- Name -> String
- Email -> String
- Pomona ID -> int or String
- The year they entered Pomona -> int
- Academic standing -> String
- Classes they are currently enrolled in -> String[]
- How many credits they have taken so far -> int
- Have they graduated -> boolean

But this was for ONE student

- Would we need to make a variable for every single student at Pomona?
- And how can we logically organize them together so that it is clear which variables correspond to which student?
- What if we need to change information about a student?
- What if we want to distinguish between unique information (e.g., name) and shared information across all students (e.g., current semester)?
- Our code just doesn't scale up.

Object-oriented programming to the rescue

- Objects: logical bundles of software of related state (data) and behavior (procedures working on that data).
- State: the individual characteristics stored in variables (or fields).
  - e.g., name, ID, year entered Pomona, etc.
- Behavior: methods operate on internal state of objects and serve as the primary mechanism for object-to-object communication.
  - Determine academic standing based on student's credits and GPA, award them Latin Honors based on GPA, etc.

Class

- A blueprint or prototype from which objects are created.
- An object is an instance of a class and the process of creating it is called instantiation.
- In our example, a class would be a general recipe for what defines a Pomona student in general terms. An object would be an actual student whose information we specified based on that general recipe.

Declaring a class

```
public class ClassName {
    // variables (state)
    // methods (behavior)
}
```

- The class body is surrounded by curly braces.
- Class name is a noun and capitalized by convention.

Writing our first class

- Make a PomonaStudent.java file and within it write a PomonaStudent class:

```
public class PomonaStudent {

}
```

Writing our first class - variables we will need for every student

```
public class PomonaStudent {

    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    String[] enrolledClasses;
    boolean graduated;

}
```

PRACTICE TIME - Worksheet

- Assume you are writing a big application that an animal rescue will use to keep track of the pets it shelters.
- You have determined you want to make one class for each type of pet.
- Define a class `Dog` and declare variables that correspond to a dog's name, breed, age, days spent in rescue, vaccine names it has received so far, and whether it has been adopted.

ANSWER - Worksheet

```
public class Dog{

	String name;

	String breed;

	int age;

	int daysInRescue;

	String[] vaccines;

	boolean adopted;

}
```

Instantiating objects

- To instantiate a new object use the new keyword. E.g.,
  - `PomonaStudent student1 = new PomonaStudent();`
- Once you have instantiated an object, you can change its state through the dot operator. E.g.,
  - `student1.name = "Ravi Kumar";`
  - `student1.email = “rkjc2023@mypomona.edu";`

Instantiating objects

- We typically (but not always) instantiate objects in the `main` method of a class. E.g.,

```
public static void main(String args[]){
    PomonaStudent student1 = new PomonaStudent();
    student1.name = "Ravi Kumar";
    student1.email = "rkjc2023@mypomona.edu";
    student1.id = 1234;
}
```

Constructors

- We can also initialize fields during instantiation.
- To do, we will need a special type of method, a constructor.
- Constructors are methods that have the same name with the class and can take 0 or more parameters that typically correspond to all or a subset of the variables. E.g.,

```
public PomonaStudent(String studentName, String studentEmail, int studentId){
      name = studentName;
      email = studentEmail;
      id = studentId;
}
```

parameters

arguments

- We can now instead write:

```
PomonaStudent student2 = new PomonaStudent("Ravi Kumar", "rkjc2023@mypomona.edu", 1234);
```

Constructors

- If we don't specify a constructor, Java makes implicitly one for us, the zero-argument constructor.
  - All variables are initialized to their default value, i.e.,
  - `int->0`
  - `double ->0.0`
  - `boolean -> false`
  - and any object reference (e.g., `String` or an array) is set to null.
- The no-argument constructor is what we invoked before:
  - `PomonaStudent student1 = new PomonaStudent();`
- Once we specify a constructor, we HAVE to explicitly create a no-argument constructor; our code above would stop working otherwise.

Overloading constructors

- We can have more than one constructors that specify different ways that an object of our class can be instantiated.
  - E.g., a different constructor could only initialize a student's name upon instantiation. i.e.:

```
public PomonaStudent(String studentName) {
   name = studentName;
}
```

- This is known as overloading. Java knows which constructor you mean to use by matching the number, type, and order of arguments you are passing to the equivalent parameters.

Instance variables

- Once we have instantiated an object, we can access its instance (or member) variables using the dot operator. E.g.,

```
public static void main(String args[]){
    PomonaStudent student2 = new PomonaStudent("Ravi Kumar”, "rkjc2023@mypomona.edu", 1234);
    System.out.println(student2.name); //prints Ravi Kumar
    student2.name = "Alexandra Papoutsaki”;
    System.out.println(student2.name); //prints Alexandra Papoutsaki
```

- We cannot access instance variables without specifying the object. For example:

```
public static void main(String args[]){
    System.out.println(name); //won’t compile, WHOSE name???
```

this keyword

```
public PomonaStudent(String studentName, String studentEmail, int studentId){

    name = studentName;

    email = studentEmail;

    id = studentId;

}
```

Instance variables

parameters

- The keyword this refers to the current object. We can use it to differentiate between instance variables and parameters.

```
public PomonaStudent(String name, String email, int id){

    this.name = name;

    this.email = email;

    this.id = id;

}
```

Instance variables

parameters

Initializing arrays

- When initializing an array, e.g., `String[] enrolledClasses;` we have two options:
  - We could initialize them using the curly braces, e.g.,
    - `enrolledClasses = {"CSCI062", "PHYS051", "ANTH124", "PE050", “HIST032"};`
- Or, we could just determine the storing capacity and reserve a *fixed* space in memory, e.g.,
  - `enrolledClasses = new String[6];`
  - This will reserve 6 spots in memory (counting at 0…5)
  - `enrolledClasses.length` will be 6.
  - Until we specify what values each index will hold, they will all have the default value of the type the array holds, e.g.,
    - `enrolledClasses` will hold `[null,null,null,null,null,null]`

PomonaStudent class so far

```
public class PomonaStudent {

    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    String[] enrolledClasses;
    boolean graduated;

    public PomonaStudent(String name, String email, int id){
        this.name = name;
        this.email = email;
        this.id = id;
        enrolledClasses = new String[6];
    }

}
```

PRACTICE TIME - Worksheet

- Add a constructor to your Dog class so that you initialize its name, breed, and age. You can assume that dogs receive 6 vaccines maximum. Use the keyword `this`.

ANSWER - Worksheet

```
public class Dog{

    String name;
    String breed;
    int age;
    int daysInRescue;
    String[] vaccines;
    boolean adopted;

    public Dog(String name, String breed, int age){
        this.name = name;
        this.breed = breed;
        this.age = age;
        vaccines = new String[6];
    }
}
```

PRACTICE TIME - Worksheet

- Define a `main` method and within it instantiate two objects of type `Dog`. Initialize their name, age, and breed to whatever you choose.
- Once you instantiate the two `Dog` objects, initialize their days in rescue to whatever number you want.

ANSWER - Worksheet

```
public class Dog{

    String name;
    String breed;
    int age;
    int daysInRescue;
    String[] vaccines;
    boolean adopted;

    public Dog(String name, String breed, int age){
        this.name = name;
        this.breed = breed;
        this.age = age;
        vaccines = new String[6];
    }

    public static void main(String[] args){
        Dog dog1 = new Dog("Rex", "German Shepherd", 3);
        Dog dog2 = new Dog("Lassie", "Rough Collie", 7);
        dog1.daysInRescue = 3;
        dog2.daysInRescue = 47;
    }

}
```

Instance methods

- A collection of grouped statements that perform a logical operation and control the behavior of objects.
- By convention method names should be a verb (+ noun) in lowercase.
- Syntax: access modifier returnType methodName(type parameter-name,…){…}. E.g.,
  - `public int getYearEntered(){return yearEntered;}`
- Method signature: method name and the number, type, and order of its parameters.
- Control goes back to the calling program as soon as a `return` statement is reached. If it does not return anything it is `void`. E.g.,
  - `public void printName(){System.out.println(name);}`
- Can be overloaded (same name, different number, type, or order of parameters). This is common for constructors.
- Invoked using the dot operator, e.g.,
  - `student1.printName();`

PomonaStudent class so far

```
public class PomonaStudent {

    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    String[] enrolledClasses;
    boolean graduated;

    public PomonaStudent(String name, String email, int id){
        this.name = name;
        this.email = email;
        this.id = id;
        enrolledClasses = new String[6];
    }
    public int getYearEntered(){
        return yearEntered;
    }

    public void setYearEntered(int yearEntered){
        this.yearEntered = yearEntered;
    }

}
```

Static variables and methods

- Static (or class) variables are variables shared across all objects. E.g.,
  - `static int studentCounter;`
- Can be accessed through the class name, without needing to instantiate an object. E.g.,
  - `System.out.println(PomonaStudent.studentCounter);`
- When a method only accesses static variables then it can be defined as static. E.g.,

```
static void graduateAllStudents(){
   studentCounter = 0;
}
```

PomonaStudent class so far

```
public class PomonaStudent {

    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    String[] enrolledClasses;
    boolean graduated;

    static int studentCounter;

    public PomonaStudent(String name, String email, int id){
        this.name = name;
        this.email = email;
        this.id = id;
        enrolledClasses = new String[6];
        studentCounter++;
    }
    public int getYearEntered(){
        return yearEntered;
    }

    public void setYearEntered(int yearEntered){
        this.yearEntered = yearEntered;
    }

}
```

PRACTICE TIME - Worksheet

- Add a dog counter in your `Dog` class.
- Update its constructor to increase the counter by one every time a new `Dog` object is created.
- Write an `adopt` method that updated the dog's adoption status and decreases the counter of dogs.

ANSWER - Worksheet

```
public class Dog{

    String name;
    String breed;
    int age;
    int daysInRescue;
    String[] vaccines;
    boolean adopted;

    static int dogCounter;

    public Dog(String name, String breed, int age){
        this.name = name;
        this.breed = breed;
        this.age = age;
        vaccines = new String[6];
        dogCounter++;
    }

    public void adopt(){
        adopted = true;
        dogCounter--;
    }
```

Data Hiding

- Core concept in Object-Oriented Programming.
- We encapsulate data and related methods in one class and we restrict who can see and modify data.
  - For example, FERPA protects the privacy of students so the Registrar cannot share their academic record freely, even if its their parents who request it.
- Java uses access modifiers to set the access level for classes, variables, methods and constructors.

Access Modifiers

- You are already familiar with the `public` keyword. E.g., `public class PomonaStudent`.
- For classes, you can either use public or *default*:
  - `public`: The class is accessible by any other class. E.g.,
    - `public class PomonaStudent`
  - *default*: The class is only accessible by classes in the same package (think of it as in the same folder. More later). This is used when you don't specify a modifier. E.g.,
    - `class PomonaStudent`
- For variables, methods, and constructors, you can use any of the following:
  - `public`: the code is accessible by any other class
  - `private`: The code is only accessible within the declared class
  - `default`: The code is only accessible in the same package. This is used when you don't specify a modifier
  - `protected`: The code is accessible in the same package and subclasses (More later).

Data Hiding

- To follow the concept of data hiding, we define variables as `private`.
- We provide more lax (i.e. *default*, `protected`, or `public`) getter and setter methods to access and update the value of a `private` variable.

PomonaStudent class so far

```
public class PomonaStudent {

    private String name;
    private String email;
    private int id;
    private int yearEntered;
    private String academicStanding;
    private String[] enrolledClasses;
    private boolean graduated;
    private static int studentCounter;

    String getName() {
        return name;
    }

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

    String getEmail() {
        return email;
    }


    void setEmail(String email) {
        this.email = email;
    }
…
```

PRACTICE TIME - Worksheet

- Update all of the variables in Dog to private.
- Define a getter method that returns the days spent in rescue, and a setter method that updates the days spent in rescue. What access modifier do you want to provide?
- Use them to update the days spent in rescue for the two objects of type `Dog` you instantiated.

ANSWER - Worksheet

```
public class Dog{

    private String name;
    private String breed;
    private int age;
    private int daysInRescue;
    private String[] vaccines;
    private boolean adopted;

    private static int dogCounter;

    public Dog(String name, String breed, int age){
        this.name = name;
        this.breed = breed;
        this.age = age;
        vaccines = new String[6];
        dogCounter++;
    }

    public int getDaysInRescue(){
        return daysInRescue;
    }

    protected void setDaysInRescue(int daysInRescue){
        this.daysInRescue = daysInRescue;
    }
```

String representation of an object

- If we want to print an object, we must override the method toString. e.g.,

```
public String toString(){
    return "Name: " + name + "\nemail: " + email + "\nid: " + id;
}
public static void main(String args[]){
    PomonaStudent student1 = new PomonaStudent("Ravi Kumar", "rkjc2023@mypomona.edu", 1234);
    System.out.println(student1);
}
```

- Will print:
  - Name: Ravi Kumar
  - email: rkjc2023@mypomona.edu
  - id: 1234

PRACTICE TIME - Worksheet

- Add a toString method to your `Dog` class and return whatever string representation you think is appropriate for a `Dog` object.

ANSWER - Worksheet

```
public String toString(){
    return "Name: " + name + "\nBreed: " + breed + "\nAge: " + age;
}
```

Constant variables

- If you want a variable to be constant, that is its value to remain unchanged once it is initialized, you can use the keyword `final`. E.g.,
  - `final int LEVELS = 5;`
- It is conventional to capitalize the variable name to convey it is a constant.
- It is common for a final variable to also be static. E.g.,
  - `static final double PI = 3.141592653589793;`

Lecture 3: Classes and Objects

- Classes and Objects

Readings:

- Oracle's guide: What Is an Object? What Is a Class? https://docs.oracle.com/javase/tutorial/java/concepts/index.html

- Classes and Objects: https://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Code

- Lecture 3 code

Worksheet

- Lecture 3 worksheet

Practice Problems:

- Make a class Cat for the animal rescue program you are building. Consider what variables (instance or static), methods (instance or static), and constructors you would need. Make sure to hide any sensitive data.