# Lesson 5.01 — Object Oriented Programming

## Overview

**Objectives — *Students will be able to...***

- **Describe** the relationship between classes, objects, and client code.
- **Predict** the output of the code that uses objects.

**Assessments — *Students will...***

- **Complete** Practice questions

**Homework — *Students will...***

- **Read** BJP 8.2 up to "Mutators and Accessors"

## Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of WS 5.1.1
- **Classroom copies** of the textbook (or just section 8.1)
- **Bookmarks on student computers (or emailed links) to Bulbapedia**

If you decide to email or link to the Pokemon wiki page, the complete address is: http://bulbapedia.bulbagarden.net/wiki/Main_Page. Students can also easily search for the page by typing in "bulbapedia" or "pokemon wiki."

## Pacing Guide

| Section | Total Time |
|---|---|
| Bell-work and attendance | 5min |
| Introduction: Discussion | 10–30min |
| Introduction: Syntax Notes | 10–20min |
| Activity 1: Practice | 15min |
| Activity 2: Researching for a custom class | 15min |

Read through all of the Instructor's notes before you plan this lesson. **In some classrooms, it might be best if you extend this into a two-day lesson.** It is worth it to spend plenty of time discussing concepts of design and debating choices to drive home the idea that object and class construction are completely customizable. If you are expanding this lesson plan to a two-day lesson, use graphic organizer WS 5.1 to help students organize their thoughts from the class discussion during Day 1. A suggested stopping point for Day 1 syntax/notes is indicated by a **dotted line**.

On Day 2, start with the syntax notes below the dotted line, then invite students to complete Activity 1 and 2.

## Procedure

If you have the option to rearrange seating, set up student seats in a circle for class discussion. As students filter in, start small discussions with the following talking points: To write personalized programs that handle real-life data, they'll need to know how to design and create their own models of real-life events, phenomena, or processes. Rather than learning new structural code, they're going to start focusing on making design decisions.

Up until now we have used pre-made objects and classes that are given to us by importing java classes (bonus points if students can name some examples). Moving forward, they'll be using customized classes that they've built themselves.

**Bell-work and Attendance [5 minutes]**

**Introduction: Discussion [10-30 minutes]**

1. Give students permission to put down their pencils to participate in a discussion (you can review important definitions as part of a recap before class practice). **It's important to get students used to critiquing and debating design decisions before getting into the nuts-and-bolts of objects.**
    - What do we mean by "models of real-life events, phenomena, or processes?" In programming, objects are models of something else:
        - The object forecast is a model of a future weather event.
        - The object student1 is a model of an actual student that goes to this school.
        - The object myDog is a model of your pet dog.
    - What do we mean by making "design decisions?" To model a forecast, student, or dog, you need to make certain decisions about what data and actions are important to your model. Ask students to offer some design decisions for a forecast object:
        1. What types of data are important for a local forecast?
        2. What sorts of data might be important for a student writing the forecast object in Alaska? Arizona? Oklahoma?
        3. What behavior (methods) might we want our forecast object to have?
    - Discussion points to bring up/guide students to:
        - If you're interested in forecasting a tornado, you might choose to model the weather with finer granularity than you might opt for if predicting rainfall or cloud cover.
        - Different model components will be appropriate in different situations.
        - Does a forecast written by a student in Arizona need the same inputs and methods as a forecast in Oklahoma? What might be different? The same?
2. Work through another design discussion about the student1 or myDog object. How might you design the student1 object in a music school? A martial arts school? A high school or college? What behaviors (methods) and data (states) might they have in common? Which would be different? (Both the music school and martial arts school might include fields for billing information, but only the martial arts school would have fields with emergency medical information.)
    - If students are suggesting overly-complex models of student1 or myDog, use it as an opportunity to discuss **complexity**. Is it always a good idea to add states and behaviors (data and methods)? When is it appropriate to make a model more complex? When do you want to keep it simple?
    - The rule of thumb is to only include the complexity you need. If this language works for your class, tell them to always design around principles of completeness, robustness, and simplicity.
        - Completeness: Does this model do everything I need it to do? Does it contain all the data I need it to contain?
        - Robustness: Is this model sufficiently flexible for everything I need it to do? Can I use it in different contexts (this isn't important in this unit, but will become important later.)
        - Simplicity: Can my model be simpler? Extra complexity can lead to coding mistakes or errors down the road. I want my code to be easy for other programmers to read/interpret.

**Introduction: Syntax Notes [10-20 minutes]**

1. Distribute graphic organizers WS 5.1 to the students that need extra structure for their notes. Start by showing students the difference between a program that is a set of actions (commands), and a program that contains data and behavior (data and methods).
    - This object digits is a (very simple, and somewhat boring) model of a collection of integers. This model contains state (data) and actions (methods):

```
int[] digits = {1,2,3,4,5,6,7,8,9,10};          // The data is stored in the array.

System.out.println(Arrays.toString(digits));    // The method dictates actions
                                                // to be done with the data.
```

Depending on your class' culture and level of understanding, you might consider a brief side-discussion on other ways we could get the program to print out the array. Ask for students to volunteer some other code, and ask students to argue/debate whether it is easier to write the code from scratch or ask the array to format itself (as above).

2. An **object** is a combination of data AND methods. The book refers to these as **state** (content, or data) and **behavior** (methods, or what is to be done with the data).
   - The behavior can modify or report the data contained by the object.
   - The book refers to the data as the "state" of the object
   - Ask students to describe how we use the word "state" in daily life, and ask them to compare to how you use "state" in computer science.
   - Ask students to explain how the word "behavior" applies to an object.
   - By contrast, this is a program that is not an object/model:

     ```
     while (guess != number) {
         System.out.println("Incorrect.");
         System.out.println("Your guess? ");
         guess = console.nextInt();
         numGuesses++;
     }
     ```

     Ask students why this isn't an object/model of something. (This program only contains actions, no behavior.)

3. The code that uses the objects is called **client code**. You'd never create a model of something (create an object) if you weren't going to use it with other programs (client code).
   - To pull from our earlier example, what sort of program/client code might make use of student1? (An attendance program, a grade records program)
   - What program (client code) might need to access the data and methods (state and behavior) stored in the myDog object? (A veterinarian's digital medical charts, a dog show's registration program)

4. Using whichever example is most engaging to your students, have them write 3 objects with proper syntax as a Think-Pair-Share. Before you list the objects, have a brief design conversation as a whole group, so students can decide what data and methods should be included in each object. Some suggested objects:
   - myDog, teachersDog, sistersDog
   - student1, student2, student3
   - forecastNY, forecastAZ, forecastOK

   Ask students if they wrote the same code over and over again, how long it took them, if they can think of another way to make the task easier (some of them might have read about classes the night before).

5. A **class** is a blueprint (or outline) that tells Java how to make a particular set of objects. We could save ourselves a lot of time by writing a class Student, which will make sure that every student object has <whatever fields your students decided student should have>.
   - Each object is called an instance of that class.
     - The object myDog is an instance of the Dog class. So is the object teachersDog and sistersDog. What is another instance of the Dog class? (Any individual dog is correct—categories of dog, such as seeingEyeDog are not objects, but probably classes in a hierarchy—more on that later!)
   - Ask students to give instances of the Student and Forecast classes.

- Finally, ask for students to provide examples of their own classes and instances. Some examples:
  - *Class*: `ClassroomChair` *Instances*: Student 1's Seat, Student 2's seat, *etc.*
  - *Class:* `Pens` *Instances*: My pen, your pen, the pen on the desk
- If you use these examples, walk around the classroom, physically touching or picking up the instances of each class.
- As you work through these examples, be careful not to generate an example that illustrates a superclass with classes. This will be coming up in the next unit, so it's important not to confuse students.

  If a student gives an example that is overly general, you can redirect them towards a more specific example:

  **Incorrect Student Example:** Class = Car, Instances = Jetta, Prius, Model T **Correction:** Class = Car, Instances = myCar, yourCar, thatCarOverThere
- A class contains several key components:
  - **Fields** — which outline what data (state) the object will hold
  - **Methods** — which determine the behavior of each object
  - **Constructors** — code that initializes each object as its being constructed with the new keyword
- A class uses encapsulation to protect the object's data from outside access (by the client code). You do this by making each field private.

  If you need additional examples, work through the book example of the Point Class, driving home the idea that a class can contain whatever they want/need. If your students are easily grasping these concepts, have the students help you create a boutique/bespoke class Pokémon. The idea here is to give them a design problem that they can work through, making choices about content and behavior that result in a model of the Pokémon game.

**Activity 1: Practice [15 minutes]**

1. Students will be working in groups for much of the week, so have them work independently today. If students are really having a rough time, work through the first Practice question together as a whole group.
2. Have students read through the Point Class example before moving on to the Practice questions.
3. Have students log in to back of the chapter to complete the following self-check questions:
   a. Self-Check 8.1: whatIsOOP
   b. Self-Check 8.2: whatIsAnObject
   c. Self-Check 8.3: StringObject
   d. Self-Check 8.4: ReferenceMystery3
   e. Self-Check 8.5: CalculatorObject
4. If more 25% or more of the class is struggling, return to whole group with the stipulation that students who get it may continue working independently.

**Activity 2: Researching For a Custom Class [5 minutes]**

1. Ask students to take a few minutes to research the Pokémon game in earnest. An ongoing design challenge will be for them to construct a model of the Pokémon game that resembles the one they play at home. If they are already familiar with the game, they should visit Bulbapedia to learn how some of the stats are calculated. If they are not familiar with the game, they should watch game examples on YouTube, read the rules and steps on Nintendo's website, or navigate through the intro pages on Bulbapedia. This can be extended as a homework assignment.

   If your students have trouble reading, direct them to the following webpages instead of having them search at their own discretion:

   1. http://www.pokemon.com/us/parents-guide/ (Basic overview of the game)

2. https://youtu.be/DlEbXH8eUTk?t=1m26s (this is a 30 minute YouTube video of gameplay—students should either watch it at home or only watch the first 5 - 10 minutes in class)
3. http://www.pokemon.com/us/pokedex/ (types of Pokemon)
4. http://tinyurl.com/no4mzic (Pokemon with stats)
5. http://en.wikipedia.org/wiki/Gameplay_of_Pok%C3%A9mon (Wikipedia entry)

2. As students research, have them jot down ideas for what type of data and behaviors they would want to include in a Pokémon class. What design features do they feel are most important to their model? Encourage students to justify their answers to each other, you, and the class at large.
3. If students show interest, let them read ahead in the textbook to figure out exactly what fields, methods, and constructors they might use in the next class. Ask students to reflect on their current model and think of ways they could improve/change it.

---

**Final Project** In your final project for this course you will be designing and programming a game similar to Pokemon. Each character will be implemented as an object so you will have to develop a good understanding of how these work.

---

## Accommodation and Differentiation

If you have students who are speeding through this lesson, invite them to create a mind map of the concepts introduced today using key vocabulary words. If the mind map is thorough, give the student materials to turn the map into a large-format poster for the classroom.

For students struggling with the vocabulary, ask them to bring in physical objects that can all be classified as the same type. (Perhaps they bring in drink bottles, or types of snacks, or different writing implements.) Using those objects, you should have them create an in-class display that models the relationship between classes and instances of the class. Have students label the physical objects with:

- Class name
- Object
- Instance of [name of class]
- Sample code or pseudocode (on index cards or pieces of paper) for:
  - Fields
  - Constructors
  - Methods

If students need additional anchoring for the "object" concept, ask them if they can guess what object they've worked with before. (String objects, Scanner objects, etc.) Have a brief discussion where you:

- Ask students to provide examples of data stored in String objects (Data includes the characters and their locations, information about the length of the string.)
- Ask student to provide examples of behavior (methods) associated with Strings. (Methods include anything used in the .dot notation, such as `s.length()`.)

If you need additional discussion, ask students to discuss the behavior and state of array objects.

## Teacher Prior CS Knowledge

Up to this point, students have been consumers of objects. They have used the `String`, `Scanner`, and `ArrayList` classes. As we move into object oriented programming concepts, students will be able to create classes and objects. This is like being able to read a language to being able to write a language. Both require some knowledge and skill in addition to lots of practice. The knowledge and skill are related for reading and writing, but not necessarily the same.

## Misconceptions

Students think `class` is a collection of objects, rather than a template for creating objects.

## Videos

- BJP 8-1, *Defining a Class* http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoPlayer.php?id=c8-1
- CSE 142, *Intro Object Oriented Programming* (11:26–19:53) https://www.youtube.com/watch?v=0IGWknpGPhM&start=686
- CSE 142, *Class* vs *Object* (26:36–31:43) https://www.youtube.com/watch?v=0IGWknpGPhM&start=1596

## Forum discussion

Lesson 5.01 Object Oriented Programming (TEALS Discourse account required)