Java: Classes and Objects

Instructor: Nihshanka Debroy
Java objects

- Java is an *object-oriented* programming language
  - use *objects* to define both the data type and the operations that can be applied to the data

- Objects have *attributes and functionality*
  - attributes describe the state of the object
  - the functionality of an object is the set of actions the object can perform

- In Java, we define an object’s attributes using variables and its functionality using methods
Real-world objects

- Suppose we want to describe a car in terms of its attributes and functionality

- Attributes:
  - int year;
  - int mileage;
  - String make;
  - String model;
  - boolean manual_transmission;

- Methods:
  - void brake()
  - int getMileage()
  - boolean needsGas()
  - void shift(int gear)
Java classes

- Java objects are created using *classes*
- Encapsulation
  - combining elements to create a new entity
- A class encapsulates the variables and methods that define an object
- Instantiation
  - the act of creating an object
  - objects are called class instances
- Java provides many predefined classes
- You can also define your own classes
Primitive Data Types and Objects

- **Primitive data types**: for basic values like a number or letter character
- **Objects** can be much more complicated and can hold lots of data as well as methods.
- Variable holds either a primitive data type value or a reference to an object.

  ✓ **Reference** is an address (location) in the computer's memory.
  ✓ Primitive data type variable holds an actual value,
  ✓ Variable of an object data type holds memory address (reference) of object.

- **Class**: definition of a particular type of object, blueprint for creating objects of that type.
- Class defines what variables and methods that any object of its type will have.
Abstract View of a Class

• Create objects to keep track of the shooting percentage of a basketball player.
• Write a class called Shooter to define a type of object.
• Objects made from this class, of the type Shooter - instances of the Shooter class.

Outline (abstraction) for the class looks like this:

class name:
    Shooter

variables:
    int shotsAttempted // records how many shots attempted
    int shotsMade      // records how many shots made

methods:
    Shooter()          // constructor method
    madeShot()         // add another made shot
    missedShot()       // add another missed shot
    getShotPercentage() // return the current shooting percentage as a 'double' value
 Constructor

• Name of constructor method - same as its class name.

• **Constructor** of a class: method used to create an instance of that class
  (Shooter() constructor creates an instance of the Shooter class -> a Shooter object)

  ➔ Using this class as a "blueprint", can create Shooter objects

• To create 2 variables holding references to Shooter objects, player1 and player2:

  ```java
  Shooter player1 = new Shooter();
  Shooter player2 = new Shooter();
  ```

• **new** operator to create a new object (instantiation) - by calling constructor method

  ➔ Objects referenced by player1 and player2 -> 2 different objects, even though same "blueprint" class used to create them.
Dot Operator

- To use player1's madeShot() method use: `player1.madeShot();`
- **dot operator** (.) - operators we've seen were +, -, *, /, %
  - name of the variable we want on left side
  - name of that variable's method that we want to use on right side
  - left and right parentheses after the method name: b/w these parentheses, place whatever data (variables, literals, etc.) the method requires be passed to it
  - dot operator also used to access variables or **fields** within an object - place the field's name to the right of the dot operator (instead of a method name)