Object-Oriented Programming (OOP) Model

- Design problems/programs such that they correspond to real world entities
  - a.k.a Object
  - Something that has real existence
  - Examples: person, dog, car, student, bank account

Objects

- Objects have
  - Data/State: information about that object
  - Behaviors: describe how the object behaves

- Class in Java for OOP model
  - Is an abstract description of objects
    - Describe common features to all objects
    - Templates for creating objects
    - Hence object is an instance of a class

Example 1: Student Object

- Data: name, address, major, courseList
- Behavior: change address, change major, add/drop a course
**Example 2: Dog Objects**

**OOP in Java**
- A Java program is a *collection* of objects
  - Objects model the parts of a problem
- Classes contain
  - **Data Fields** that hold the data for each object
    - Data is stored in variables (*instance variables*)
  - **Methods** that describe the actions/behaviors the object can perform
  - **Constructor(s)** that tell how to *create a new object* of this class

**OOP Class Structure**
```java
class Classname {
    // Data Fields: data for each object
    ...
    // Constructor: create a new object of this class
    ..
    // Methods: describe the behaviors the object can perform
    ..
}
```

**Data Fields**
- Classes describe the data held by each of its objects
- Also known as *instance* variables

```java
class Dog {
    String name;
    int age;
    ..
    ..
    ..
}
```
Data Fields (contd..)

- Data can also describe the **relationship** of the object to other objects
- Example: a checkingAccount might have
  - An **accountNumber** (used as an ID number)
  - A **balance** (the internal state of the account)
  - An **owner** (some object representing a person)

Constructor

- A **constructor** is a piece of code that
  - Constructs or initializes a new object of that class
- If you don’t write a constructor
  - Java defines one for you (behind the scenes)
  - i.e. **default** constructor
- Usually a constructor is written to *initialize* an object’s data fields
  - Constructor can take input parameters
  - Default constructor initialize object’s state default value for that type. E.g. for type int, the default value is zero

Constructor: Initialize Data Example

**Example 1:**
```java
class Dog {
    String name;
    int age;
    //constructor with parameters
    Dog(String Name, int Age) {
        name = Name;
        age = Age;
    }
    //rest of the class
}
```

**Example 2:**
```java
class Dog {
    String name;
    int age;
    //constructor without parameters
    Dog() {
        name = “Unknown”;
        age = 1;
    }
    //rest of the class
}
```

Creating Objects

- **Declare** a variable of appropriate type to hold the object
  - The **type** of an object is the **class** that describes that object
    - E.g. For Dog object we a need a variable of **type Dog**
- To create an object:
  ```java
  Dog d1 = new Dog();
  Dog d2 = new Dog(“Fido”, 5);
  ```
  - make a new object
  - make a new object
- **new** allocates space for the object in computers memory
- Constructor initializes the data of the object
Methods

- A class may contain methods that describe the behavior of objects.
- Two kinds of Methods:
  - Query Methods: ask an object about its state
    - What’s your name? Age? Amount in Bank Account?
  - Command Methods: change an object’s state
    - Withdraw $100 from my bank account \(\Rightarrow\) my bank balance changes
- Methods just like how you are use to writing static methods
  - But written without the keyword “static”

Example of Method

Methods usually go after the data & constructor (style rule)

class Dog {
  ...
  void setDogAge(int dogAge){
    age = dogAge;
  }
  int getDogAge(){
    return age;
  }
}

Note: Methods have access to instance variables defined within class (outside of any method)

Calling/Invoking a Method on an Object

- Calling/Invoking a method on an Object
  - Is a programming terminology for asking or making an object to perform some behavior
- In general
  - `objectName.methodName(0 + parameters)`
  - Note: that in a method call:
    - the number, order, and type of arguments must match the corresponding parameters
  - Examples: `d1.getAge(); d1.setAge(5);`

Temporary /Local vs. Instance Variables

- Temporary/local variables are known
  - From the point of declaration until the end curly brace of the block in which they are declared
- In contrast, instance variables are
  - Declared outside of any method
  - Known to all methods in the class in which they are declared
Asking Object about its data directly

- It *may* possible to ask an object about its data without querying the object.
- `ObjectName.DataField;`
- But you can prevent such change by making object data `private`.
- E.g. `private int age;`

Example in Dr Java

```
Dog d1 = new Dog("Fido", 5);
>d1.age
5
>d1.age = 6;
>d1.age
6
```

Putting it all together

- When you want to create standalone Java application program one of the classes should contain the `main` method.
- Entry point into your program.
- Method `main` is not part of any object.
- Hence it is static.
- In main(), we create objects, ask objects to do something or interact with another object.

Main Method

```
public class DogTest {
    public static void main(String[] args) {
        Dog d1 = new Dog("Fido", 5); // creates a Dog
        System.out.println("Age = " + d1.getAge());
        d1.setAge(6); //change Fido’s age from 5 to 6
        System.out.println("Name = " + d1.getName());
        System.out.println("Age = " + d1.getAge());
    }
}
```

A Counter class example

```
Counter
int count: 3
incrementCount
Counter
int count: 4
```

```
Counter
int count: 4
reset
Counter
int count: 0
```
**Complete Counter class**

```java
public class Counter {
    private int count;  // Data field (instance variable)

    public Counter() { count = 0; }  // constructor

    public int currentCount() { return count; }  // query

    public void incrementCount() { count = count + 1; }  // command

    public void reset() { count = 0; }  // command
}
```

**Counter class declaration**

**Counter Object Behaviors**

- Counter c = new Counter();
- If you want to increment count value of object c
  - c.incrementCount();
- If you want see the count value of object c
  - c.currentCount();
- If you want to reset count value of object c
  - c.reset();

**Complete Counter Program**

```java
public class Counter {
    private int count;

    public Counter() { count = 0; }

    public int getCount() { return count; }

    public void incrementCount() {
        count = count + 1;
    }

    public void reset() {
        count = 0;
    }

    public static void main(String[] args) {
        Counter c = new Counter();
        int whatIsCount = 0;
        c.incrementCount();
        c.incrementCount();
        whatIsCount = c.currentCount();
        System.out.println(whatIsCount);
        c.reset();
        whatIsCount = c.currentCount();
        System.out.println(whatIsCount);
    }
}
```

**General OOP Java Program**

- A program consists of one or more classes
- Typically, each class is in a separate java file
Writing and Running Programs in OOP

- When you write a program
  - You are writing classes and all the things that go into classes
  - Your program typically contains commands to create objects and make them do something

- When you run a program
  - It creates objects, and those objects interact with one another and do whatever they do to cause something to happen

OOP Recap

- **Class**: a template for creating objects
- An object is an *instance* of a class
- Example:
  - One Dog class
  - Multiple Dog objects
  - Lord of the Rings Simulation
    - One Human class, multiple Human objects
    - One Elf class, multiple Elf objects
    - One Orc class, multiple Orc objects

Advantages of OOP

- OOP is conducive to good design and efficient redesign
- Most changes affect only a few classes