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

Object Characteristics

- A Object has
  - **Data/State**: information about that object
  - **Behaviors**: describe how the object behaves

  ![Student Object](image)

  **Data/State**: name, address, major, courseList
  **Behavior**: change address, change major, add/drop a course

More Examples

- **Radio**
  - **Data/State**: on/off, current volume, current station
  - **Behavior**: turn on/off, increase/decrease volume, seek station

- **Dog**
  - **Data/State**: name, age, color, breed
  - **Behavior**: change age, bark, hungry
### OOP in Java

- A Java program is a **collection** of objects.
  - Objects model the parts of a problem.
- **Class** in Java for OOP model:
  - Is an abstract description of **objects**
    - Describe common features to all objects
    - Templates for creating objects
  - Hence we say that object is an instance of a class
    - Each object has its own unique data.

### Anatomy of Class in OOP

- **Classes** contain:
  - **Data Fields** that hold the data for each object
    - Data is stored in variables
  - **Methods** that describe the actions/behaviors the object can perform
  - **Constructor(s)** to initialize object with some information

### 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
    ..
}
```

### Example 2: Dog Objects

- Different types of dogs represented as objects.
Data Fields

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

```java
class Dog {
    String name;
    int age;
    ...
    // rest of the class...
}
```

Constructor

- A constructor is a piece of code that initializes a new object
- Used to initialize an object's data fields
  - Constructor can initialize data to set value or taken external values
  - If you don't write a constructor
    - Java defines one for you (behind the scenes) i.e. default constructor
      - The data will initialize to the 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
}
```

Important: Constructor name is same as ClassName

A class can have more than one constructor

Creating Objects

- Class is just an abstract description
- In order to use objects we need to create them

**Step 1**
- 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
    - Dog d1;
    - Dog Fido;
Creating Objects (contd..)

Step 2
- Create object with keyword `new` and call to the constructor

```java
Dog d1;
d1 = new Dog();
```

- The keyword `new` allocates space for the object in computers memory
- Constructor initializes the data of the object

```java
Dog d2;
d2 = new Dog("Fido", 5);
```

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 ⇒ my bank balance changes
- Methods just like writing static methods
  - But written *without* the keyword “static”

Example of Method

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

```java
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)

Sending messages to objects

- We don’t perform operations on objects, we “talk” to them
  - This is called *sending a message* to the object

- A message looks like this:
  ```java
  object.method(extra information)
  ```
  - The `object` is the thing we are talking to
  - The `method` is a name of the action we want the object to take
  - The `extra information` is anything required by the method in order to do its job
Sending Messages to Objects (contd..)

- `objectName.methodName(0 + parameters)`
  - Examples: `d1.getAge();  d1.setAge(5);`
  - Note: the number, order, and type of arguments must match the corresponding parameters as in the methods header description

- Is just like *Calling/Invoking a static method but in OOP we asking or making a particular object to perform some behavior*

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
  - Cannot use modifier private or public with these

- **In contrast, instance** variables are
  - Declared outside of any method
  - Known to all methods in the class in which they are declared
  - Can use modifier private or public with these

Asking Object about its data directly

- It *may* possible to ask a object about its data without querying the object
  - public or no modifier
  - `ObjectName.DataField;`

- But you can prevent such change by making object data *private*
  - E.g. `private int age;`

Encapsulation or Information Hiding

- One of the advantages of OOP is that object need not reveal all of its attributes (data/state) and behavior

- We can hide details of one object from another

- Use modifiers (private/public) to hide information
  - Ideally we make all instance variable(s) private

- Provide methods (query/command) if you want to allow the data to read or written
  - Getter methods to read e.g. `getAge()`
  - Setter methods to modify e.g. `setAge()` -> not necessary to provide

Example in Dr Java

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

Example in Dr Java

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

- **Class**: a template for creating objects
  - Variables – data
  - Methods – behavior
  - Constructor – initialize data

- An object is an *instance* of a class
  - Dog d = new Dog("Lassie",5); -> d is an object of class Dog

- A Java program is collection of co-operating objects
  - E.g. Lord of the Rings Simulation
    - One Human class, multiple Human objects
    - One Elf class, multiple Elf objects
    - One Orc class, multiple Orc objects
    - One weapon class, multiple weapon