Java Programming Language

By
Sunil Kumar(Master of Sc.)
Bangalore, India

Lesson 6

3. Agenda

- Classes
- Using Classes
- Classes as reference types
- Encapsulation and access modifier
- Basic Method
- Field accessors and mutators

Classes in Java

- Java is a n object-oriented language
- Objects encapsulate data, operations, and usage semantics
  - Allow storage and manipulation details to be hidden
  - Separates “WHAT” is to be done from “HOW” it is done
- Classes provide a structure for describing and creating objects

Classes in Java Contd..

- A class is a template for creating an object
  - Declared with the class keyword followed by class name
  - Java source file name normally has same name as the class
  - Body of the class is contained within brackets
  - A class is made up of both state and executable code
  - Fields
    - Store object state
  - Methods
    - Executable code that manipulates state and perform operations
  - Constructors
    - Executable code used during object creation to set initial state

Classes in Java Contd..

```
class flight{
        int passengers;
        int seats;
        Flight(){
                seats = 100;
                passengers = 0;
        }
        void add1Passenger(){
                if(passengers < seats)
                        passengers += 1;
        }
}
```

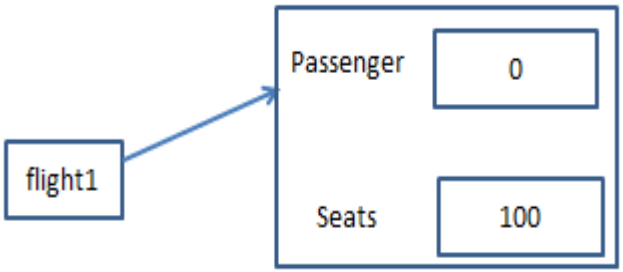
Using Classes

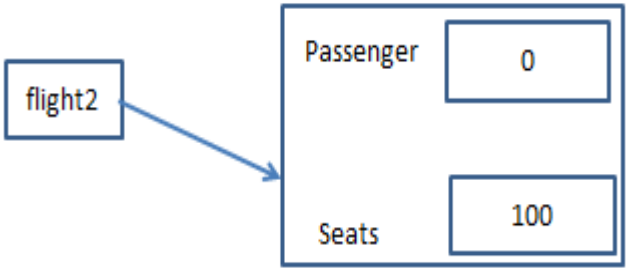
- Use the new keyword to create a class instance(a.k.a. an object)
  - Allocates the memory described by the class
  - Returns a reference to the allocated memory
- Syntax to declare

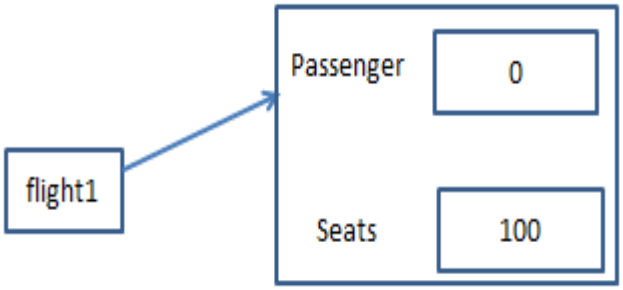
```
Flight nycToSf; //either we can declare in 2 line
nycToSf = new Flight(); //declare & create object
Flight slcToDallas = new Flight(); // or in 1 line
```

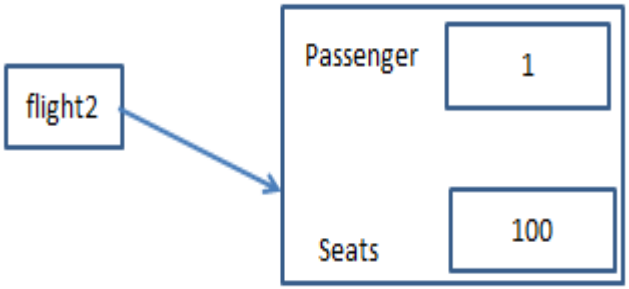
Classes are Reference Types

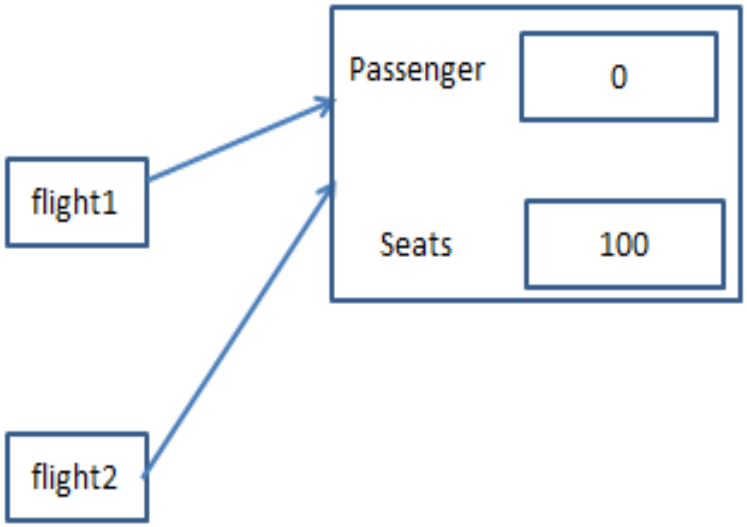
```
Flight flight1 = new Flight();
Flight flight2 = new Flight();

flight2.add1Passenger();
System.out.println(flight2.passengers);   // 1

flight2 = flight1;
System.out.println(flight2.passengers);   //0

flight1.add1Passenger();
flight1.add1Passenger();

System.out.println(flight2.passengers); //2
```

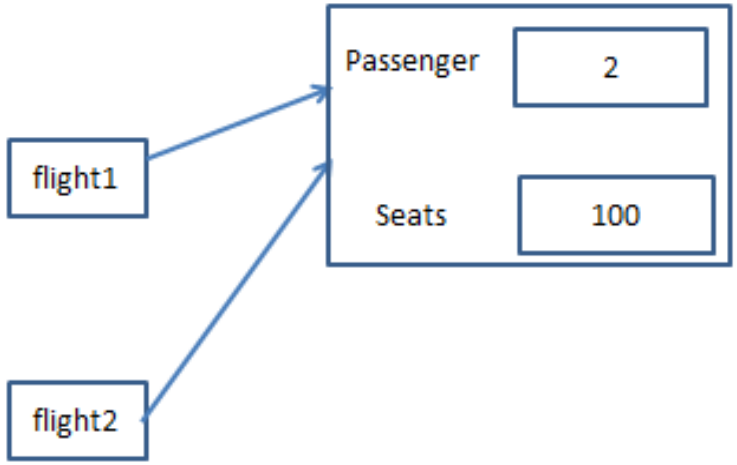

flight1
Passenger
0
Seats
100


flight2
Passenger
0
Seats
100


flight1
Passenger
0
Seats
100


flight2
Passenger
1
Seats
100


flight1
flight2
Passenger
0
Seats
100


flight1
flight2
Passenger
2
Seats
100

Points to be Noted

- When we assign one object to other, then we assign the references to the object and not the value.
- This is totally different with respect to the primitive data type where we copy the actual value to another, but here we just copy the reference of an object.

Encapsulation & Access Modifier

- Since Java is an Object Oriented language, we should use object oriented technique to built our program.
- The internal representation of an object is generally hidden. This means that external world(end-user) should be only familiar with what we are doing in the program not about how we are doing the program.
- This concept is called as encapsulation.
- Java uses access modifier to achieve encapsulation.

Basic Access Modifier

Modifier	Visibility	Usable on Classes	Usable on Members
No access modifier	Only within its own package(a.k.a. package private)	Y	Y
Public	Everywhere	Y	Y
Private	Only within its own class	N (as private applies to top-level classes, private is available to nested-classes)	Y

Applying Access Modifiers

```
public class flight{
    private int passengers;
    private int seats;
        public Flight(){
                        seats = 100;
                        passengers = 0;
            }
         public void add1Passenger(){
                        if(passengers < seats)
                                passengers += 1;
                        else
         private handleTooMany();
            }
            void handleTooMany(){
                    system.out.println(“Too Many”);
            }
}
```

```
Flight flight1= new Flight();
System.out.println(flight1.passengers);
/*not allowed as it will give compile time error as
access modifier used here is private */

flight1.add1Passenger();
/*No problem to call as it is public*/

flight1.handleTooMany();
/*not allowed as it will give compile time error as
access modifier used here is private */
```

Point to be Noted

- Normally class file has the same name as the name of the class.
- Once we mark the class as public, the source file name has to be the same name. This is mandatory, and so the previous class definition has to be in the file name called flight.java else it will throw an error.

Naming Classes

- Class name follow the same rules as variable name.
- Class name conventions are similar to variables with some differences.
  - Use only letters and numbers.
  - First character is always a letter
  - Follow the style often referred to as “Pascal Case”
    - Start of each word, including the first, is upper case
    - All other letters are lower case
  - Use simple, descriptive nouns
  - Avoid abbreviations unless abbreviation’s use is more common that full name.

Naming Classes

- Example:

  Class BankAccount{……}

  Class Person{……}

  Class TrainingVideo{……..}

  Class URL{…..}

Method Basics

- Executable code that manipulates state and perform operations
  - Name
    - Same rules and conventions as variables
    - Should be a verb or action
  - Return type
    - Mandatory to mention return type.
    - Use void when no value returned
  - Typed parameter list
    - Can be empty
  - Body contained with brackets

Method Basics

- Syntax

```
return-type name(typed-parameter-list){
        statements;
}

public class MyClass{
public void showSum(float x, float y, int count){
        float sum = x+y;
        for(int i =0; i <count; i++)
                system.out.println(sum);
        }
}

MyClass m = new MyClass();
m.showSum(7.5, 1.5, 2);
```

Exiting from a Method

- A method exits for one of three reasons
  - The end of the method is reached
  - A return statement is encountered
  - An error occurs
- Unless there's an error, control returns to the method caller.
- Example

Exiting from a method

```
public class MyClass{
    public void showSum(float x, float y, int count){
        if(count < 1)
                return;
        float sum = x+y;
        for(int i =0; i <count; i++)
                system.out.println(sum);
           return;
        }
}

    MyClass m = new MyClass();
    m.showSum(7.5, 1.5, 0);
    System.out.println("I am back");
```

Method Return Values

- A method returns a single values
  - A primitive value
  - A reference to an object
  - A reference to an array
    - Array are objects

Method Return Values

```
public class Flight{
 private int passenger;
 private int seats;
 //constructor and other method for more clarity
 public boolean hasRoom(Flight f2){
                int total = passenger + f2.passengers;
                if(total <= seats)
                                return true;
                else
                                return false;
                }
 public Flight createNewWithBoth(Flight F2){
                Flight newFlight = new Flight();
                newFlight.seats = seats;
                newFlight.passengers = passengers + f2.passengers;
                                return newFlight;
                }
}
```

```
Flight lax1 = new Flight();
Flight lax2 = new Flight();

/*add passenger to both flight*/

Flight lax3;
If(lax1.hasRoom(lax2))
Lax3=
lax1.createNewWithboth(lax2);
```

Special References: This

- Java provides special references with predefined meanings
  - this is an implicit reference to the current object
    - Useful for reducing ambiguity
    - Allows an object to pass itself as a parameter

```
public class Flight{
private int passenger;
private int seats;
//constructor and other method for more clarity
public boolean hasRoom(Flight f2){
        int total = this.passenger + f2.passengers;
        if(total <= seats)
                return true;
        else
                return false;
 }
```

Special Reference: null

–NULL is a reference literal

- Represents an uncreated object
- Can be assigned to any reference variable

```
Flight lax1 = new Flight();
Flight lax2 = new Flight();
 /*add passenger to both flight*/
 Flight lax3 = null;
 If(lax1.hasRoom(lax2))
 Lax3= lax1.createNewWithboth(lax2);
 //do some other work
 If(lax3 != null)
 System.out.println(“Flights combined”);
```

Field Encapsulation

- In most cases, a class field should not be directly accessible outside of the class.
  - Helps to hide implementation details
  - Use method to control field access

Accessors and Mutators

- Use the accessors/mutators pattern to control field access.
  - Accessors retrieves field value
    - Also called getter
    - Method name: getFieldName
  - Mutator modifies field value
    - Also called setter
    - Method name: setFieldName

Accessors and Mutators

```
public class Flight{
    private int passengers;
    private int seats;
    //other members elided for clarity
    public int getSeats(){
        return seats;
    }
    public void setSeats(int seats){
        this.seats = seats;
    }
}
```

Using Accessors and Mutators

```
public class flight{
    private int passengers;
    private int seats;
        public Flight(){
                        seats = 100;
                        passengers = 0;
            }
         public void add1Passenger(){
                        if(passengers < seats)
                                passengers += 1;
                        else
         private handleTooMany();
            }
            void handleTooMany(){
                    system.out.println(“Too Many”);
            }
}
```

```
Flight flight1 = new Flight();
flight1.setSeats(100);
System.out.println(flight1.getSeats());  //100
```

Summary

- A class is a template for creating an object
  - Declared with class keyword
  - Class instances (a.k.a. objects) allocated with new keyword
- Classes are references types
- Use access modifier to control encapsulation
- Methods manipulate state and perform operations
  - Use return keyword to exit and/or return a value
- Fields store object state
  - Interaction normally controlled through accessors(getter) and mutators (setters).