Working with Classes and Interfaces in Java

UNDERSTANDING JAVA CLASSES AND OBJECTS

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim jwhh.com

Overview

Declaring classes

Class members

Working with objects

Encapsulation and access modifiers

Field accessors and mutators

Prerequisite course

Getting Started with Programming in Java

Java Is an 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

A Class Is a Template for Creating Objects

Flight.java

```
class Flight

  // class members

}
```

Declared with class keyword followed by the class name

Body of the class is contained within brackets

Java source file name normally has same name as class

Classes

A class is made up of both state and executable code

Fields

Store object state

Flight.java

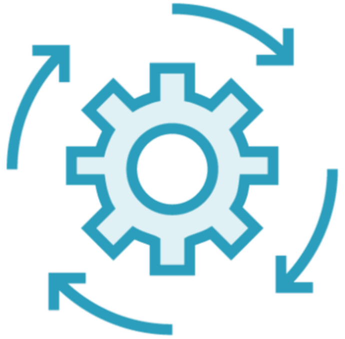
```java
class Flight {
  int passengers;
  int seats;









}
```

Classes

A class is made up of both state and executable code

Fields

Store object state



Methods

Executable code that manipulates state and performs operations

Flight.java

```java
class Flight {
  int passengers;
  int seats;





  void add1Passenger() {
    if(passengers < seats)
      passengers += 1;
  }
}
```

Classes

A class is made up of both state and executable code

Fields

Store object state

Methods

Executable code that manipulates state and performs operations

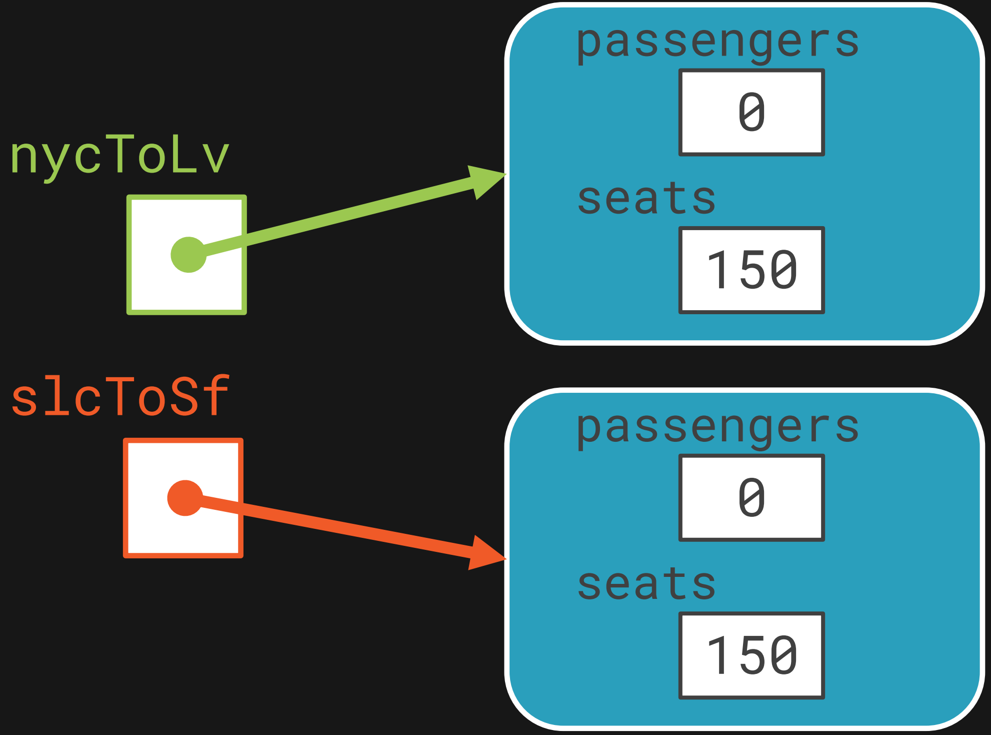
Constructors

Executable code used during object creation to set initial state

Flight.java

```java
class Flight {
  int passengers;
  int seats;
  Flight() {
    seats = 150;
    passengers = 0;
  }
  void add1Passenger() {
    if(passengers < seats)
      passengers += 1;
  }
}
```

```java
Flight nycToLv;

nycToLv = new Flight();

Flight slcToSf = new Flight();
```



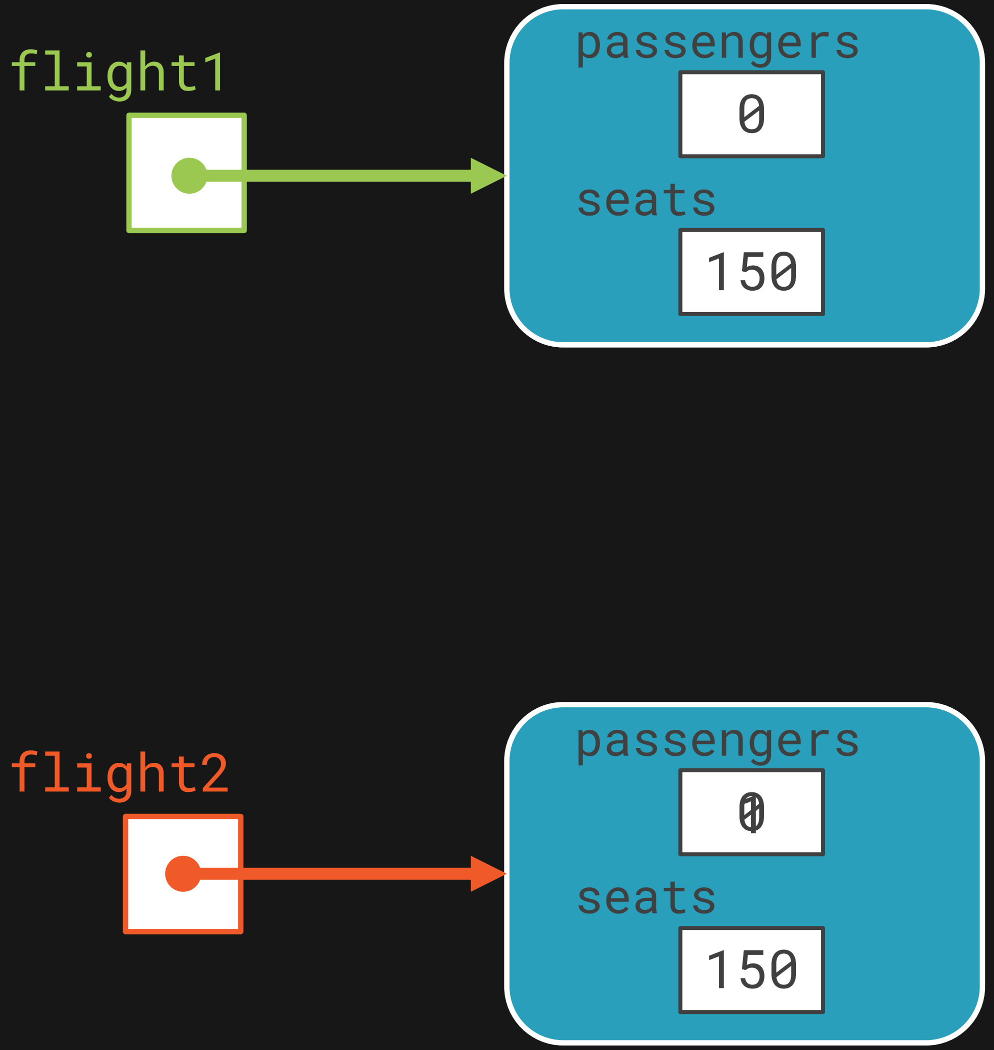
Using Classes

Use the new keyword to create a class instance (a.k.a. an object)

- Allocates the memory described by the class, and runs the constructor
- Returns a reference to the allocated memory

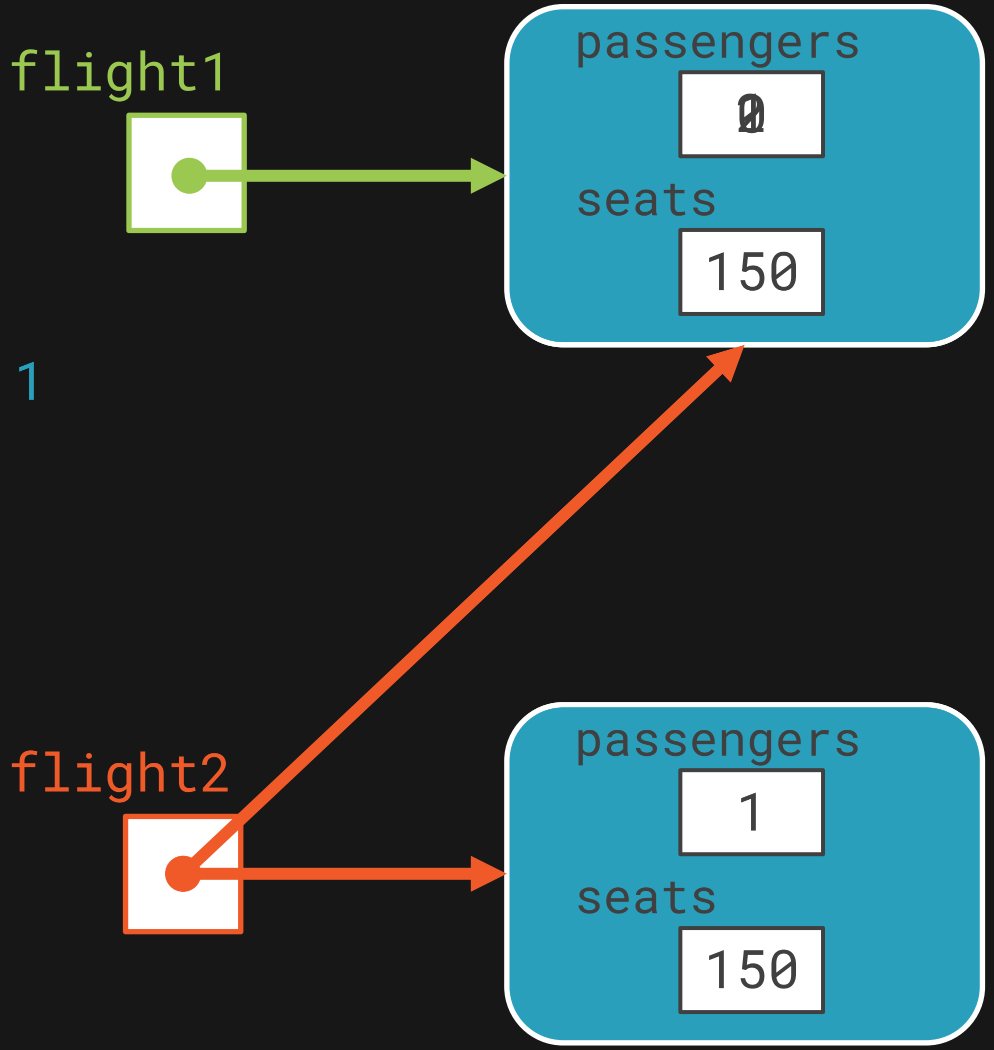
Understanding Classes as Reference Types

```
Flight flight1 = new Flight();

Flight flight2 = new Flight();

flight2.add1Passenger();

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



Understanding Classes as 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);

flight1.add1Passenger();

flight1.add1Passenger();

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



Encapsulation and Access Modifiers

The implementation details of a class are generally hidden

This concept is known as encapsulation

Java uses access modifiers to achieve encapsulation

Basic Access Modifiers

Modifier	Visibility	Usable on Classes	Usable on Members
		*	

* As private applies to top-level classes; private is available to nested-classes

Applying Access Modifiers

Main.java

```
Flight flight1

System.out.println(
    flight1.passengers);

flight1.add1Passenger();
```

Flight.java

```
class Flight {
    int passengers;
    int seats;
  Flight(){...}
  void add1Passenger(){...}
}
```

Flight.java

```
public class Flight {
  // other members elided for clarity
  public void add1Passenger() {
    if(passengers < seats)
      passengers += 1;
    else
      handleTooMany();
  }
  private void handleTooMany() {
    System.out.println(“Too many”);
  }
}
```

Applying Access Modifiers

Main.java

```
Flight flight2 = new Flight();
flight2.~~handleTooMany()~~;
flight2.add1Passenger();
```

Flight.java

```
public class Flight {
  private int passengers;
  private int seats;
  public Flight(){...}
  public void add1Passenger(){...}
  private void handleTooMany(){...}
}
```

Special References

this

Implicit reference to current object

Useful for reducing ambiguity

Allows an object to pass itself as a parameter

null

Represents an uncreated object

Can be assigned to any reference variable

Special Reference: this

```
public class Flight {
  private int passengers;
  private int seats;
  // other members elided for clarity
  public boolean hasRoom(Flight f2) {
    int total =      passengers + f2.passengers;
    return total <= seats;
  }
}
```

Special Reference: null

```
Flight lax1 = new Flight();
Flight lax2 = new Flight();
// add passengers to both flights
Flight lax3
if(lax1.hasRoom(lax2))
  lax3 = lax1.createNewWithBoth(lax2);
// do some other work
if(lax3 != null)
    System.out.println(“Flights combined”);
```

Field Encapsulation

The specific fields a class uses to manage data values is generally considered to be an implementation detail

In most cases these details should not be directly accessible outside the class

Use methods to control field access

Accessors and Mutators

Use the accessor/mutator pattern to control field access

Accessor retrieves field value

Also called getter

Method name: get*FieldName*

Mutator modifies field value

Also called setter

Method name: set*FieldName*

Flight.java

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

Accessors and Mutators

Main.java

```
Flight slcToNyc = new Flight();

slcToNyc.setSeats(200);

System.out.println(slcToNyc.getSeats());
```

```
200
```

Summary

Class

- Template for creating objects

Object

- Instance of a class

Classes are reference types

- Class variables simply hold a reference
- Instances created with new keyword
- Multiple variables can reference the same instance

Summary

Fields

- Store object state

Methods

- Executable code
- Manipulate state
- Perform operations

Constructors

- Executable code
- Runs during object creation
- Sets initial state

Summary

this

- Implicit reference to current object

null

- Represents an uncreated object

Summary

Access modifiers

- Control class visibility
- Control member visibility
- Enable encapsulation

Fields not normally directly accessible

- Use methods to provide access
- Accessors retrieve field values
- Mutators modify field values