Nested Types and Anonymous Classes

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim jwhh.com

Overview

Declaring one type within another type

Nesting types for naming scope

Inner classes

Anonymous classes

Nested types

- A type declared within another type

Are members of the enclosing type

- Nested type can access private members of enclosing class

Nested types support all access modifiers

- No modifier (a.k.a. package private)
- public
- private
- protected

Two categories of nested types

- One provides only naming scope
- The other links type instances

Similar but different

- Syntax is similar
- Behavior is very different

Nesting types for naming scope

- Type name scoped within enclosing type
- No relationship between nested type and enclosing type instances

Applies to the following nested types

- Static classes nested within classes
- All classes nested within interfaces
- All nested interfaces

Passenger.java

```
public class Passenger implements Comparable<Passenger> {

  private int memberLevel; // 3 (1st priority), 2, 1
  private int memberDays;



  public Passenger(String name, int memberLevel, int memberDays) {
    this.name = name;
    memberLevel = memberLevel;
    memberDays = memberDays;
  }
} // other members elided
```

Passenger.java

```
public class Passenger implements Comparable<Passenger> {
  public static class RewardProgram {
  private int memberLevel; // 3 (1st priority), 2, 1
  private int memberDays;
  }


  public Passenger(String name, int memberLevel, int memberDays) {
    this.name = name;
    memberLevel = memberLevel;
    memberDays = memberDays;
  }
} // other members elided
```

Passenger.java

```
public class Passenger implements Comparable<Passenger> {
  public static class RewardProgram {
    private int memberLevel; // 3 (1st priority), 2, 1
    private int memberDays;
  }


  public Passenger(String name, int memberLevel, int memberDays) {
    this.name = name;
    memberLevel = memberLevel;
    memberDays = memberDays;
  }
} // other members elided
```

Passenger.java

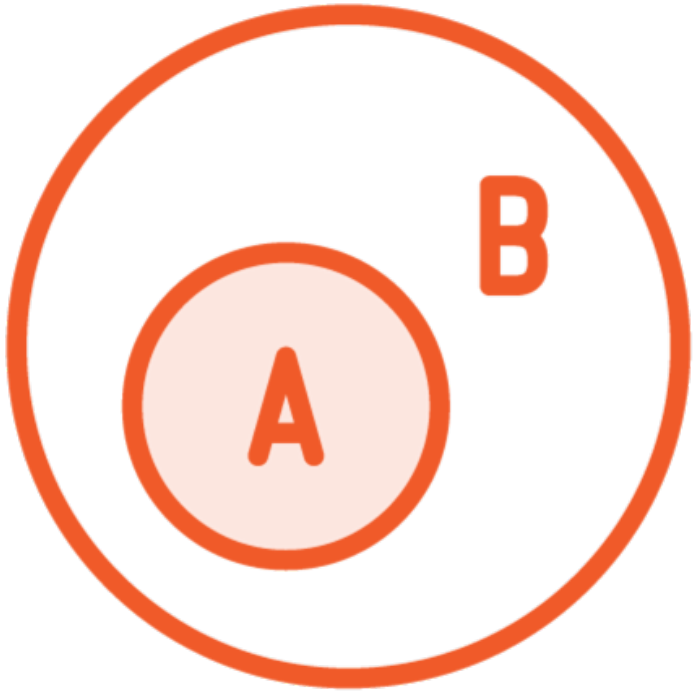
```
public class Passenger implements Comparable<Passenger> {
  public static class RewardProgram {
    private int memberLevel; // 3 (1st priority), 2, 1
    private int memberDays;
  } // getters and setters elided
  private RewardProgram rewardProgram
  public Passenger(String name, int memberLevel, int memberDays) {
    this.name = name;
    memberLevel = memberLevel;
    memberDays = memberDays;
  }
} // other members elided
```

Passenger.java

```
public class Passenger implements Comparable<Passenger> {
  public static class RewardProgram {
    private int memberLevel; // 3 (1st priority), 2, 1
    private int memberDays;
  } // getters and setters elided

  private RewardProgram rewardProgram = new RewardProgram();

  public Passenger(String name, int memberLevel, int memberDays) {
    this.name = name;
    rewardProgram.memberLevel = memberLevel;
    rewardProgram.memberDays = memberDays;
  }
} // other members elided
```

Main.java

```java
Passenger steve = new Passenger("Steve", 3, 180);

Passenger.RewardProgram platinum = new Passenger.RewardProgram();
platinum.setMemberLevel(3);

if(steve.getRewardProgram().getMemberLevel() == platinum.getMemberLevel())
    System.out.println("Steve is platinum");
```



Inner classes

- Type name scoped within enclosing type
- Creates instance relationship
- Each instance of nested class associated with an instance of enclosing class

Applies to the following nested type

- Non-static classes nested within classes

Flight.java

```
public class Flight implements Comparable<Flight>, Iterable<Passenger> {
  private ArrayList<Passenger> passengerList = new ArrayList<>();
  public Iterator<Passenger> iterator() { return passengerList.iterator(); }

  private class FlightIterable implements Iterable<Passenger> {
    @Override
    public Iterator<Passenger> iterator() {
        Passenger[] passengers = new Passenger[passengerList.size()];
        passengerList.toArray(passengers);
        Arrays.sort(passengers);
        return Arrays.asList(passengers).iterator();
    }
  }
```

this

Flight.this

Flight.java

```
  public Iterable<Passenger> getOrderedPassengers() {
    FlightIterable orderedPassengers
    return orderedPassengers;
  }
} // other members elided
```

Main.java

```java
Flight f175 = new Flight(175);
f175.add1Passenger(new Passenger("Luisa", 1, 180));
f175.add1Passenger(new Passenger("Jack", 1, 90));
f175.add1Passenger(new Passenger("Ashanti", 3, 730));
f175.add1Passenger(new Passenger("Harish", 2, 150));

// Returns Flight class' Iterable<Passenger> implementation
for(Passenger p : f175)
  System.out.println(p.getName());

// Luisa
// Jack
// Ashanti
// Harish
```

Main.java

```
Flight f175 = new Flight(175);
f175.add1Passenger(new Passenger("Luisa", 1, 180));
f175.add1Passenger(new Passenger("Jack", 1, 90));
f175.add1Passenger(new Passenger("Ashanti", 3, 730));
f175.add1Passenger(new Passenger("Harish", 2, 150));

// Returns Flight.FlightIterable class' Iterable<Passenger> implementation
for(Passenger p : f175.getOrderedPassengers())
  System.out.println(p.getName());

// Ashanti
// Harish
// Luisa
// Jack
```

Anonymous classes

- Declared as part of their creation
- Use as simple interface implementations
- Use as simple class extensions

Anonymous classes are inner classes

- Instance is associated with enclosing class instance

Creating anonymous class instance

- Use new keyword with base class or interface name
- Place parenthesis after name

Adding code

- Placed within brackets
- Can implement methods
- Can override methods

Flight.java

```
public class Flight implements Comparable<Flight>, Iterable<Passenger> {
  private ArrayList<Passenger> passengerList = new ArrayList<>();
  public Iterator<Passenger> iterator() { return passengerList.iterator(); }

  private class FlightIterable implements Iterable<Passenger> {
    @Override
    public Iterator<Passenger> iterator() {
        Passenger[] passengers = new Passenger[passengerList.size()];
        passengerList.toArray(passengers);
        Arrays.sort(passengers);
        return Arrays.asList(passengers).iterator();
    }
  }
```

Flight.java

```
  public Iterable<Passenger> getOrderedPassengers() {
    FlightIterable orderedPassengers = new FlightIterable();
    return orderedPassengers;
  }
} // other members elided
```

Flight.java

```
  public Iterable<Passenger> getOrderedPassengers() {
    return new Iterable<Passenger>
      @Override
      public Iterator<Passenger> iterator() {
        Passenger[] passengers = new Passenger[passengerList.size()];
        passengerList.toArray(passengers);
        Arrays.sort(passengers);
        return Arrays.asList(passengers).iterator();
      }
    } ;
  }
} // other members elided
```

Summary

Nested types

- Members of enclosing type
- Can access private members of enclosing type

Summary

Nesting types for naming scope

- Scopes name within enclosing type
- No relationship between instances of nested type and enclosing type

Inner class

- Close relationship with enclosing class
- An Instance of nested class associated with an instance of enclosing class

Summary

Anonymous classes

- Declared as part of their creation
- Can implement methods
- Can override methods

Anonymous classes are inner classes

- Associated with containing class instance

Remember

Slide and demo code for all modules is in the course exercise files