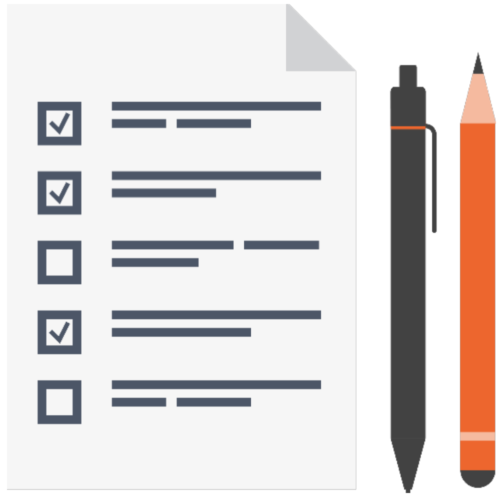
Static Members, Nested Types, and Anonymous Classes

Jim Wilson

@hedgehogjim | blog.jwhh.com | jimw@jwhh.com

What to Expect in This Module



Static members

Static initialization blocks

Nested types

Inner classes

Anonymous classes

Static Members

Static members are shared class-wide

Not associated with an individual instance

Declared using the static keyword

Accessible using the class name

Field

- A value not associated with a specific instance
- All instances access the same value

Method

- Performs an action not tied to a specific instance
- Can access static fields only

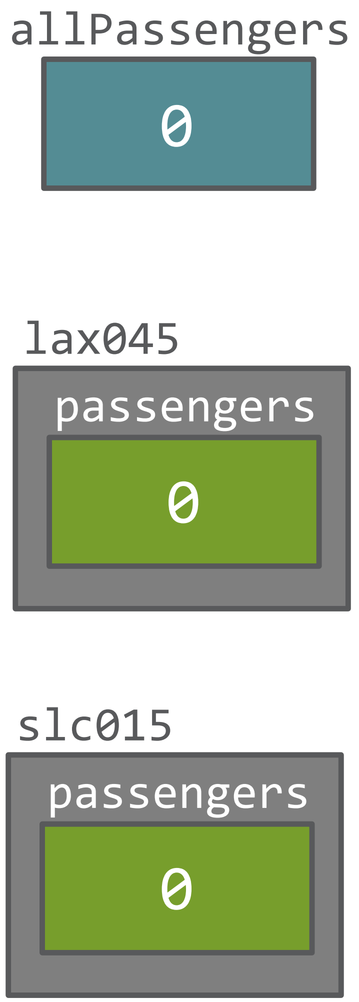
Static Members

```
  Flight.resetAllPassengers();
  System.out.println(
0   Flight.getAllPassengers());

  Flight lax045 = new Flight();
  lax045.add1Passenger();
  lax045.add1Passenger();

  Flight slc015 = new Flight();
  slc015.add1Passenger();

  System.out.println(
3   Flight.getAllPassengers());
```



```
class Flight {
   // other members elided for clarity
   int passengers;
   void add1Passenger() {
     if(hasSeating()) {
        passengers += 1;
        allPassengers += 1;
     } else
        handleTooMany();
   }
   static int allPassengers;
   static int getAllPassengers() {
      return allPassengers;
   }
   static int resetAllPassengers() {
      allPassengers = 0;
   }
}
```

Static Members

Static members are shared class-wide

Not associated with an individual instance

Declared using the static keyword

Accessible using the class name

Field

- A value not associated with a specific instance
- All instances access the same value

Method

- Performs an action not tied to a specific instance
- Can access static fields only

Static import

Provides short hand for accessing static members

Static Members

```
import static com.pluralsight.travel.Flight.resetAllPassengers;
import static com.pluralsight.travel.Flight.getAllPassengers;

Flight.resetAllPassengers();
System.out.println(
  Flight.getAllPassengers());

Flight lax045 = new Flight();
lax045.add1Passenger();
lax045.add1Passenger();

Flight slc015 = new Flight();
slc015.add1Passenger();

System.out.println(
  Flight.getAllPassengers());
```

Static Initialization Blocks

Static initialization blocks perform one-time type initialization

Executed before type's first use

Statements enclosed in brackets outside of any method or constructor

Precede with static keyword

Cannot access instance members

Must handle all checked exceptions

Static Initialization Blocks

```
public class CrewManager {
  private final static String FILENAME = "...";

  private static CrewMember[] pool;
  public static CrewMember
   FindAvailable(FlightCrewJob job) {
    CrewMember cm = null;
    for(int i=0; i < pool.length; i++) {
      if(pool[i] != null && pool[i].job == job) {
        cm = pool[i];
        pool[i] = null;
        break;
      }
    }
    return cm;
  }

  // other members temporarily elided

} // class CrewManager
```

```
CrewMember p =
  CrewManager.FindAvailable(FlightCrewJob.Pilot);
```

```
Pilot,Patty
Pilot,Paul
CoPilot,Karl
CoPilot,Karen
FlightAttendant,Fred
FlightAttendant,Phyllis
FlightAttendant,Frank
FlightAttendant,Fiona
AirMarshal,Ann
AirMarshal,Alan
```

Static Initialization Blocks

```
public class CrewManager {
  private final static String FILENAME = "...";

  private static CrewMember[] pool;
  public static CrewMember
   FindAvailable(FlightCrewJob job) {
    CrewMember cm = null;
    for(int i=0; i < pool.length; i++) {
      if(pool[i] != null && pool[i].job == job) {
        cm = pool[i];
        pool[i] = null;
        break;
      }
    }
    return cm;
  }
```

```
  static {
    BufferedReader reader = null;
    try {
      reader = new BufferedReader(. . .);
      String line = null;
      int idx = 0;
      pool = new CrewMember[10];
      while ((line = reader.readLine()) != null) {
        String[] parts = line.split(",");
        FlightCrewJob job =
          FlightCrewJob.valueOf(parts[0]);

        pool[idx] = new CrewMember(job);
        pool[idx].setName(parts[1]);
        idx++;
      }
    } catch(IOException e) {
      // handle error
    }
  }
} // class CrewManager
```

Static Initialization Blocks

```
CrewMember p =
  CrewManager.FindAvailable(FlightCrewJob.Pilot);

CrewMember c =
  CrewManager.FindAvailable(FlightCrewJob.CoPilot);

CrewMember a =
  CrewManager.FindAvailable(FlightCrewJob.AirMarshal);
```

Nested Types

A nested type is a type declared within another type

Classes can be declared within classes and interfaces

Interfaces can be declared within classes and interfaces

Nested types are members of the enclosing type

Private members of the enclosing type are visible to the nested type

Nested types support all member access modifiers

public

package private

protected

private

Nested Types

Nested types serve differing purposes

Structure and scoping

No relationship between instances
of nested and enclosing type

Static classes nested within classes

All classes nested within interfaces

All nested interfaces

Nested Types

```
public class Passenger implements  Comparable {
  // others members elided for clarity
  public static class RewardProgram {
    private int memberLevel;
    private int memberDays;

    public int getLevel() { return level; }
    public void setLevel(int level) { this.level = level; }

    public int getMemberDays() { return memberDays; }
    public void setMemberDays(int days) { this.memberDays = days; }
  }
  private RewardProgram rewardProgram = new RewardProgram();
  public RewardProgram getRewardProgram() {
    return rewardProgram;
  }
}
```

Nested Types

```
Passenger steve = new Passenger();
steve.setName("Steve");
steve.getRewardProgram().setLevel(3);
steve.getRewardProgram().setMemberDays(180);

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

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

Nested Types

```
public class Passenger implements  Comparable {
  // others members elided for clarity
  public static class RewardProgram {
    private int memberLevel;
    private int memberDays;

    public int getLevel() { return level; }
    public void setLevel(int level) { this.level = level; }

    public int getMemberDays() { return memberDays; }
    public void setMemberDays(int days) { this.memberDays = days; }
  }
  private RewardProgram rewardProgram = new RewardProgram();

  public RewardProgram getRewardProgram() {
    return rewardProgram;
  }
}
```

Nested Types

Nested types serve differing purposes

Structure and scoping

No relationship between instances
of nested and enclosing type

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

Inner classes

Each instance of the nested class is associated
with an instance of the enclosing class

- Non-static classes nested within classes

Inner Classes

```
public class FlightIterator
              implements Iterator<Person> {
  private CrewMember[] crew;
  private Passenger[] roster;
  private int index = 0;
  public FlightIterator(
   CrewMember[] crew, Passenger[] roster) {
    this.crew = crew;
    this.roster = roster;
  }
  public boolean hasNext() {
   return index < (crew.length + roster.length);
  }
  public Person next() {
   Person p = (index < crew.length) ?
     crew[index] : roster[index - crew.length];
   index++;
   return p;
  }
}
```

```
public class Flight
  implements Comparable<Flight>, Iterable<Person> {
  // others members elided for clarity
  private CrewMember[] crew;
  private Passenger[] roster;
  public Iterator<Person> iterator() {
    return new FlightIterator(crew, roster);
  }
  private class FlightIterator
                implements Iterator<Person> {
    private int index = 0;
    public boolean hasNext() {
      return index < (crew.length + roster.length);
    }
    public Person next() {
      Person p = (index < crew.length) ?
        crew[index] : roster[index - crew.length];
      index++;
      return p;
    }
  }
}
```

Inner Classes

```
public class Flight
  implements Comparable<Flight>, Iterable<Person> {
  // others members elided for clarity
  private CrewMember[] crew;
  private Passenger[] roster;
  public Iterator<Person> iterator() {
    return new FlightIterator();
  }
 private class FlightIterator
              implements Iterator<Person> {
    private int index = 0;
    public boolean hasNext() {
      return index < (crew.length + roster.length);
    }
    public Person next() {
      Person p = (index < crew.length) ?
       crew[index] : roster[index - crew.length];
      index++;
      return p;
    }
  }
}
```

Flight.this

this

Anonymous Classes

Anonymous classes are declared as part of their creation

Useful for simple interface implementations or class extensions

Anonymous classes are inner classes

Anonymous instance is associated with the containing class instance

Create as if you are constructing an instance of the interface or base class

Place opening & closing brackets after the interface or base class

Place implementation code within the brackets

Anonymous Classes

```java
public class Flight
  implements Comparable<Flight>, Iterable<Person> {
  // others members elided for clarity
  private CrewMember[] crew;
  private Passenger[] roster;
  public Iterator<Person> iterator() {
    return new FlightIterator();
  }
  private class FlightIterator
                implements Iterator<Person> {
    private int index = 0;
    public boolean hasNext() {
      return index < (crew.length + roster.length);
    }
    public Person next() {
      Person p = (index < crew.length) ?
        crew[index] : roster[index - crew.length];
      index++;
      return p;
  } }
}
```

Anonymous Classes

```
public class Flight
  implements Comparable<Flight>, Iterable<Person> {
  // others members elided for clarity
  private CrewMember[] crew;
  private Passenger[] roster;
  public Iterator<Person> iterator() {
    return new Iterator<Person>() {
      private int index = 0;
      public boolean hasNext() {
        return index < (crew.length + roster.length);
      }
      public Person next() {
        Person p = (index < crew.length) ?
        crew[index] : roster[index - crew.length];
        index++;
        return p;
      }
    }
  }
}
```

Summary

- Static methods and fields are shared class-wide
  - Not associated with an individual instance
- Static initialization blocks provide one-time type initialization
- A nested type is a type declared within another type
  - Can be used to provide structure and scoping
  - Inner classes create an association between nested and enclosing instances
- Anonymous classes are declared as part of their creation
  - Useful for simple interface implementations and class extensions