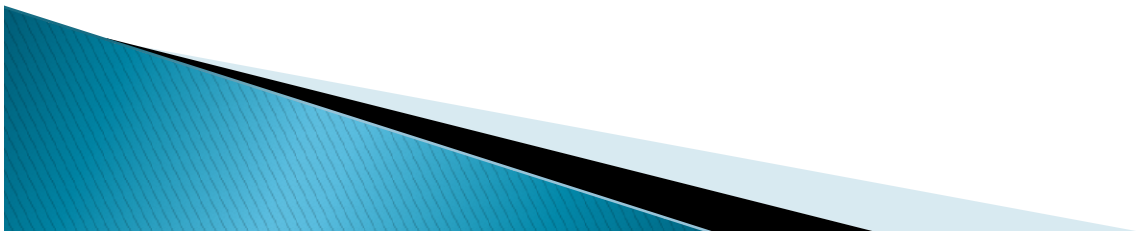
Static Members, Nested Types and Anonymous Classes

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

1. Agenda

- Static Member
- Static initialization Block
- Nested Types
- Inner Types
- Anonymous classes



Static Members

- Sometime we want a class member that's available class-wide.
  - Static members are shared class-wide.
  - Not associated with an individual instance.
- Declared using the static keyword.
  - Since they are not associated with individual instance, it can be accessed using the class name.

Static Members

```
Class Flight{
//other member eided for clarity
 int passengers;
 void add1Passeneger(){
      if(hasSeating()){
            passenegers +=1;
            allPassengers +=1;
      } else
            handleTooMany();
      }
      static int allPassengers;
      static int getAllPasseneger(){
            return allPassenegers;
      }
      static int resetAllPassengers(){
            allPassengers = 0;
      }
}
```

```
Flight.resetAllPassengers();
System.out.println(Flight.getAllPassenegers());  // 0
Flight lax045 = new Flight();
lax045.add1Passeneger();
lax045.add1Passeneger();          // 2

Flight slc015 = new Flight();
slc015.add1passenger();
System.out.println(Flight.getAllPassengers(); // 3

//Here we are not doing extra work to keep the list of
      all passenger, but simply declared it as static so
      that it will keep on increasing automatically
```

Static Members

- Static Field
  - A value not associated with a specific instance
  - All instance access the same value
- Static Method
  - Perform an action not tied to a specific instance
  - Can access static fields only

Static Members

- Static import
  - Provide short hand for accessing static members without even using the class name.
  - Thus We can access static members by using the class name. But Java also has a shorthand for accessing static members called static import.

Static Members

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

resetAllPassengers();  //Now flight qualifier is not required
System.out.println(getAllPassenegers());  // 0; Here also flight qualifier is not required
Flight lax045 = new Flight();
lax045.add1Passeneger();
lax045.add1Passeneger();            // 2

Flight slc015 = new Flight();
slc015.add1passenger();
System.out.println(getAllPassengers(); // 3; Here also flight qualifier is not required
```

Static Initialization Block

- Static initialization blocks perform one-time type initialization.
  - Execute automatically before type's first use.
- Please make a note, we have discussed earlier regarding instance initialization block. This is little different from Static initialization block.
  - Instance initialization block execute before any of constructors in our code, while static initialization block execute before we actually start using the type.
- Thus static initialization block can be treated as type Initializers.

Static Initialization Block

- Statements enclosed in brackets outside of any method or constructor and precede with static keyword.
  - Being static they can't access instance members.
  - Also since they are called automatically, we have to handle any checked exceptions within the initialization block. There is no concept of throws keyword here and we have to make sure that exception is handled within the initialization block.

Static Initialization Block Example

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

```
CrewMember p=
      CrewManager.FindAvailable(FlightCrewJob.Pilot);
CrewMember c=
      CrewManager.FindAvailable(FlightCrewJob.CoPilot);
CrewMember a=
  CrewManager.FindAvailable(FlightCrewJob.AirMarshal);
```

```
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 member temporarily elided
}   // class CrewManager
```

Static Initialization Block

- How does pool get initialized from the file. Now we have to write another static method called initialize, that loaded up the file contents. But in complex applications, we can't make sure always that the required initialize block will always run 1st. It would be nice to be able to know that pool field would automatically get loaded prior to its 1st use. And that is where static initialization block comes in.
- Static initialization blocks are really powerful capability that takes care of initializing our classes prior to their 1st use with no special calls on the part of the user.

Static Initialization Block Ex.

```
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
```