Nested Classes

Sangeeta Joshi

Agenda

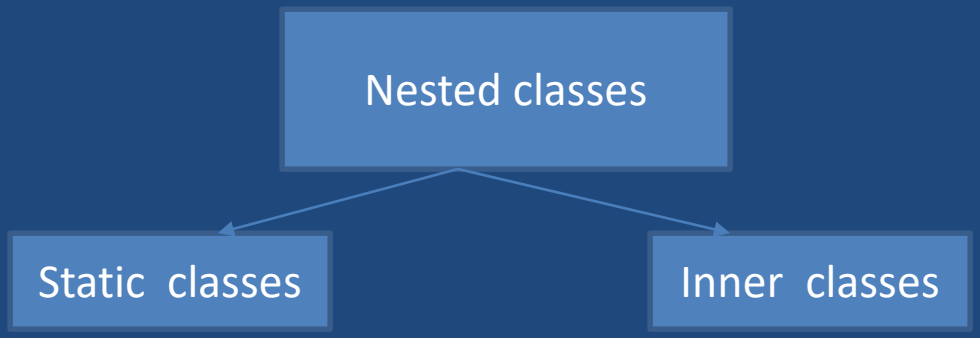
- What are Nested classes
- Why and where to use them
- Static classes - Usage
- Inner classes - Usage
- Anonymous Inner classes

Nested Class

- In Java, we can define a class within another class. Such a class is called a nested class:

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

Nested Classes




```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Why Nested Classes

- Compelling reasons for using nested classes :
- For logically grouping classes that are ***only used in one place***: If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.
- It increases ***encapsulation***: Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.
- It can lead to more ***readable and maintainable*** code: Nesting small classes within top-level classes places the code closer to where it is used.

Why Nested Classes

Advantages of Nested classes:

- one time use
- supports and improves encapsulation
- readability
- private field access (if non static i.e.inner class)

Why Nested Classes

Without existing of outer class inner class will not exist :

```
class car{
    class wheel {

        }
}
```

Static Nested Classes

- a static nested class is associated with its outer class.
- Like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class.
- it can use them only through an object reference.

OuterClass.StaticNestedClass nestedObject =
new OuterClass.StaticNestedClass();

Static Nested Classes

Using a static nested class may save spaces in some cases.

For example: implementing a Comparator inside a class, say Student.

```
public class Student {
  public static final Comparator<Student> BY_NAME = new ByName();
  private final String name;
  ...
  private static class ByName implements Comparator<Student> {
    public int compare() {...}
  }
}
```

Then the static ensures that the Student class *has only one Comparator*, rather than instantiate a new one every time a new student instance is created.

Inner Classes

- an inner class is associated with an instance of its enclosing class (As with instance methods and variables)
- It has direct access to that object's methods and fields.
- An inner class is associated with an instance, it cannot define any static members itself
- Objects that are instances of an inner class exist within an instance of the outer class.

Inner Classes

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
}
```

- To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

OuterClass.InnerClass innerObject = outerObject.new InnerClass();

Inner Classes

two special kinds of inner classes:

- *local classes*

- *anonymous classes*

Anonymous Inner Classes

- Anonymous classes enable you to make your code more concise.
- They enable you to declare and instantiate a class at the same time.
- They are like local classes except that they do not have a name.

Use them if you need to use a local class only once.