Nested Classes

Nested class basics, static vs non-static, examples, UML

Nested Class basics

- You can declare a class inside another one:

```
public class Outer {
    // Outer class stuff
    private class Inner {
        // Inner class stuff
    }
}
```

- Any class declared inside another is called a **nested class**.
- Nested classes that are not static are called **inner classes**.
- Nested classes are typically private. This prevents other classes for breaking the encapsulation of the outer class. (They can also be declared public or protected or “package private.”)
- Nested classes usually have public methods so the outer class can use them.
- Inner classes can directly access all of the members of the outer class.

Rationale for nested classes

The Java Tutorial gives 3 reasons:

- Good for logical grouping of classes that are used in only one place.
- Better encapsulation.
- Easier to read and maintain code.

Static vs non-static

- Static nested classes cannot access instance variables or instance methods of the outer class.
- An inner class can access every member of its enclosing class.
- An inner class cannot be instantiated without first instantiating its outer class.
- Consider making a nested class static if it can (potentially) stand alone. (In that case, it would have no need to access the outer class at all.)

Example: standalone classes

A very simple Student class.

```
public class Student {
   private String name;
   private String major;

   public Student(String name, String major){
      this.name = name;
      this.major = major;
   }
   public String getName() { return name; }
   public String getMajor() { return major; }
}
```

Standalone example (cont)

And a very simple client.

```
public class StudentList {
  public final int MAX_LIST = 300;
  private Student list[] = new Student[MAX_LIST];
  private int size = 0;

  public void addToFront(String name, String major)
  {
    if (size <= MAX_LIST) {
      for (int k = size; k > 0; k--)
         list[k] = list[k - 1];
      size++;
      list[0] = new Student(name, major);
    }
  }
  public void removeFromFront() { // etc.
```

Observations

- Student is a simple utility class.
- The underlying purpose of Student is to put a name and major together in the same type.
- StudentList is the only class that uses Student.
- To encapsulate all of the list information, place Student inside StudentList.
- Since Student does not rely on StudentList for any data or operations, you can make the class a static nested class or an inner class.
  - If it's non-static, then it cannot be instantiated without an instance of StudentList.
  - Making it static and public would break the encapsulation.

Inner class example

Move Student inside StudentList, and declare it as private.

```
public class StudentList {
  public final int MAX_LIST = 300;
  private Student list[] = new Student[MAX_LIST];
  private int size = 0;

  // StudentList constructor and operations go here
  // ... etc.

  private class Student {
    public String name;
    public String major;
    public Student(String name, String major){
        this.name = name;
        this.major = major;
    }
  }
}
```

Public inner class members

All of Student's members are public.

```
private class Student { // inner class declaration
   public String name;
   public String major;
   public Student(String name, String major){
      this.name = name;
      this.major = major;
   }
 }
```

Public inner class members can be used directly in outer class methods. Here's an example StudentList method.

```
 public String firstStudentName() {
   if (size == 0)
     throw new IndexOutOfBoundsException();
   return list[0].name;
 }
```

UML diagram notation

Nested class rectangles go outside their outer class rectangles. The connector is a line from the nested class to the outer class, with hatched circle against the outer class.