A class defined inside of another class can be static or non-static.

- Scope specifier: package, private, or protected.

A class defined inside of another class can be public, protected.

- Access specifier: public, protected, or private.

The rules for defining such classes are similar to fields and methods.

```java
{
  {
...
  }

class NestedClass
  ...

class EnclosingClass
}
```

Besides fields and methods, a Java class can also contain other classes.

**Nested Classes**
promote information hiding.

- Static nested classes are used mostly to avoid name clash and to

Usage:

- The enclosing class is called the outer class.

- The members (i.e., fields, methods, classes) of a static nested class

```java

{...

} class EnclosingClass

static class NestingClass

} class EnclosingClass

} when it is defined as static, it is called a static nested class.

Static Nested Classes
Its enclosing instance.

An inner class is a nested class whose instance exists within an instance of its enclosing class and has direct access to the instance members of that enclosing class.

```
class OuterClass {
    ...

    class AnInnerClass {
        ...
    }
}
```

When it is non-static, it is called an inner class.

**Inner Classes**
Inner Classes (cont.)
there is no need to have set and get methods for the state.

Since an inner object has access to its outer object (the context),
implemented as inner objects.

- In the state design pattern, the states of an object are often
  classes.
- Event listeners for Java GUI components are implemented as inner
  classes.

Usage:

Inner Classes (cont.)
Anonymous Inner Classes: An Example

```java
... {
    return head.toString(this);
}
}

public final String toString()
...

public class InnerClass
```
Anonymous Inner Classes: An Example (cont.)

```java
new IA170()
    .execute()
    return "" + host.getFirst() + host.getRest() + Object
()
{
    return "";
}

public Object emptyCase(IrStructure host, Object input)
    .execute()
    return (String)owner.

String toString(IrStructure owner)
    ...}

abstract class Anode
```
public Object emptyCase(LRStruct h, Object inp)
{
    return "";
}

public Object nonEmptyCase(LRStruct h, Object inp)
{
    return "" + h.getFirst() + h.getRest().execute(this, null);
}