Java Inner Classes

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

To access the inner class, create an object of the outer class, and then create an object of the inner class:

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}

// Outputs 15 (5 + 10)
```

Private Inner Class

Unlike a "regular" class, an inner class can be `private` or `protected`. If you don't want outside objects to access the inner class, declare the class as `private`:

```
class OuterClass {
  int x = 10;

  private class InnerClass {
    int y = 5;
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

If you try to access a private inner class from an outside class, an error occurs:

Static Inner Class

An inner class can also be `static`, which means that you can access it without creating an object of the outer class:

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
  }
}

// Outputs 5
```

Access Outer Class From Inner Class

One advantage of inner classes, is that they can access attributes and methods of the outer class:

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}

// Outputs 10
```