Nested Classes

Nested Classes and Interfaces

- Classes and interfaces can be declared inside other classes and interfaces, either as members or within blocks of code.

Some Definitions

- Nested = a class or interface definition is somewhere inside another one
- Top-level class or interface = an instance of a type **does not** need to be instantiated with a reference to any enclosing instance
- Inner class = an instance of a class **does** need to be instantiated with an enclosing instance
- Inner Member class = defined inside another class, but not inside any methods.
- Inner Local class = defined inside a method
- Named Inner Local class = has a class name
- Anonymous Inner Local class = does not have a class name

Nested Classes

```
class TheEnclosingClass{
      ...
     class ANestedClass {
        ...
      }
}
```

- **Definition**: A **nested class** is a class that is a member of another class.
- **Reason for making nested classes**:
  - the nested class **makes sense only in the context of its enclosing class**
  - the nested class **needs the enclosing class to have the right functionality**.

Nested Static Classes

- A **nested class** that is **declared static** **is attached to the enclosing class** and not to objects of the enclosing class. Instance fields and methods can not be directly accessed.

Example: *Linked List*

```
public class LinkedList {
   private Node first;
   .....
   public static class Node{
      public Node next;
      public Object data;
   }
   .....
}
```

Static Nested Classes/Interfaces — Overview

- A nested class/interface which is declared as `static` acts just like any non-nested class/interface, except that its name and accessibility are defined by its enclosing type.
- Static nested types are members of their enclosing type
  - They can access all other members of the enclosing type including the private ones.
  - Inside a class, the static nested classes/interfaces can have private, package, protected or public access; while inside an interface, all the static nested classes/ interfaces are implicitly public.
  - They serve as a structuring and scoping mechanism for logically related types

Static Inner Classes

- Since a static inner class has no connection to an object of the outer class, within an inner class method
  - Instance variables of the outer class cannot be referenced
  - Nonstatic methods of the outer class cannot be invoked
- To invoke a static method or to name a static variable of a static inner class within the outer class, preface each with the name of the inner class and a dot

Static Nested Classes/Interfaces (cont.)

- Static nested classes
  - If a class is nested in an interface, it's always static (omitted by convention)
  - It can extend any other class, implement any interface and itself be extended by any other class to which it's accessible
  - Static nested classes serve as a mechanism for defining logically related types within a context where that type makes sense.

Static Nested Classes/Interfaces (cont.)

- Nested interfaces
  - Nested interfaces are always static (omitted by convention) since they don’t provide implementation

Static Nested Classes/Interfaces (cont.)

```
public class BankAccount {
      private long number;     //account number
      private long balance;    //current balance

      public static class Permissions {
            public boolean canDeposit, canWithdraw,
  canClose;
      }
//  .  .  .
}
```

- Code outside the `BankAccount` class must use `BankAccount.Permissions` to refer to this class

```
BankAccount.Permissions perm =
  acct.permissionsFor(owner);
```

Inner Classes

- **A nested class that is not static is called an inner class.**
- An **inner class is associated with an object** of its enclosing class and it has **direct and unlimited access to that object's instance variables and methods**.
- A **nested class** can be **declared at** the **top level** inside a class **or inside any block of code**.

Inner classes

```
class Outer {
    int n;

    class Inner {
        int ten = 10;
        void setNToTen( ) { n = ten; }
    }

    void setN ( ) {
        new Inner( ).setNToTen( );
    }
}
```

Inner Class

- Name Reference
  - OuterClass inside : Use InnerClass Simple name
  - OuterClass outside : OuterClass.InnerClass

```
public static void main(String[] args) {
        OuterClass  outObj = new  OuterClass();
       OuterClass.InnerClass   inObj = outObj.new  InnerClass();
}
```

- Access Modifier
  - public, private, protected

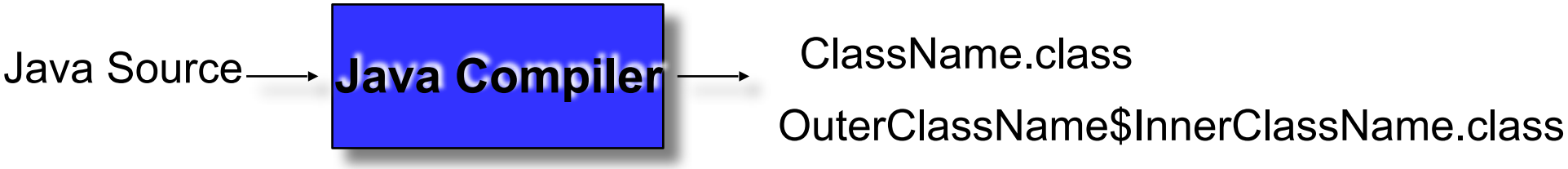
Inner class cannot have static variable

Nesting Inner Classes

- It is legal to nest inner classes within inner classes
  - The rules are the same as before, but the names get longer
  - Given class **`A`**, which has public inner class **`B`**, which has public inner class **`C`**, then the following is valid:

```
A aObject = new A();
A.B bObject = aObject.new B();
A.B.C cObject = bObject.new C();
```

Name of Inner Class




```
class Outer {
        class Inner1 {
            class Inner2 {    // …       }
            // ...
        }
        // ...
}
```

Outer.class
Outer$Inner1.class
Outer$Inner1$Inner2.class

Simple inner class example

```
class Outer{
  private int x1;

 Outer(int x1){
    this.x1 = x1;
  }

 public void foo(){ System.out.println("fooing");}

  public class Inner{
    private int x1 = 0;
    void foo(){
      System.out.println("Outer value of x1: " + Outer.this.x1);
      System.out.println("Inner value of x1: " + this.x1);
    }
  }
```

Simple example, cont -- driver

- Rules for instantiation

```
public class TestDrive{

  public static void main(String[] args){
     Outer outer = new Outer();
     Outer.Inner inner = outer.new Inner(); //must call new through
                                            //outer object handle
      inner.foo();

      // note that this can only be done if inner is visible
      // according to the regular scoping rules
  }
}
```

Non-static Classes — Inner classes

- *Inner classes* are associated with instances of its enclosing class.

```
public class BankAccount {
    private long number;      // account number
    private long balance;     // current balance
    private Action lastAct;   //last action performed

    public class Action {
        private String act;
        private long amount;

        Action(String act, long amount) {
            this.act = act;
            this.amount = amount;
        }
    }
```

Non-static Classes — Inner classes

```
    public void deposit(long amount) {
        balance += amount;
        lastAct = new Action("deposit", amount);
    }

    public void withdraw(long amount) {
        balance -= amount;
        lastAct = new Action("withdraw", amount);
    }
    // . . .
}
```