Inner Classes

Inner Class

- Form of Inner Class
  - Regular Inner Class

    -- a class inside a class

    -- static nested class

  - Anonymous inner class
  - Method-Local inner class

Regular Inner class

- A inner class which are not static, method-local, and anonymous
- class MyOuter {

  class MyInner { }

  }
- javac MyOuter.java
  - MyOuter.class
  - MyOuter$MyInner.class

Why Use Nested Classes?

- It is a way of logically grouping classes that are only used in one place.
- It increases encapsulation.
- Nested classes can lead to more readable and maintainable code.

Regular Inner class

- Referencing the Inner or Outer Instance from within the Inner Class
  - class Outer {

```
    private int x = 7;


    class Inner {
public void seeOuter() {
    System.out.println("Outer x is " + x);
        } }
public static void main (String[] args) {
Outer.Inner inner = new Outer().new Inner();
inner.seeOuter(); } }
```

Points about inner class

- Inner class be private,protected,default or public.
- It can be final,static or abstract.
- It can be a sub class.
- It can be an implementation class of an interface

Analyse the code

- class A
- {
- int x=10;
- class B{
- int x=14;
- void m(int x){System.*out.println(x+" "+this.x+" "+A.this.x);*}

- }
- }
- public class ClassDemo {
- public static void main(String[] args) {
- new A().new B().m(15);

- }

- }

Static Nested Classes

- class BigOuter {

```
static class Nest {void go() { System.out.println("hi"); } }
}
class Broom {
static class B2 {void goB2() { System.out.println("hi 2"); } }
public static void main(String[] args) {
BigOuter.Nest n = new BigOuter.Nest(); // both class names
n.go();
B2 b2 = new B2(); // access the enclosed class
b2.goB2();
} }
```

- **Output:** hi

  hi 2

Points about static inner class

- As with class methods and variables, a static nested class is associated with its outer class. And 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.
- Can access static members of the outer class directly.
- Can be instantiated without the instance of the enclosing class.

Method-Local Inner Classes

- Define class inside method
- class MyOuter2 {

```
private String x = "Outer2";

void doStuff() {

class MyInner {

public void seeOuter() {

System.out.println("Outer x is " + x); } /* close inner class
    method  */   } // close inner class definition

MyInner mi = new MyInner(); // This line must come

// after the class

mi.seeOuter();

} // close outer class method doStuff()

} // close outer class
```


Method-Local Inner Classes

- class MyOuter2 {

```
private String x = "Outer2";
void doStuff() {
String z = "local variable";
class MyInner {
public void seeOuter() {
System.out.println("Outer x is " + x);
System.out.println("Local variable z is " + z); // Won't Compile!
} } } }
```

- Method-Local inner class can't be **public, private, protected, static, transient**
- It can be **abstract** and **final**
- **Cannot access non final local variables of the method**
- **Can access Outerclass variables.**

Anonymous Inner Classes

- Inner Class without name
- class Popcorn {

```
public void pop() {
System.out.println("popcorn");
} }
class Food {
Popcorn p = new Popcorn() {
public void sizzle() {
System.out.println("anonymous sizzling popcorn");  }
public void pop() {  System.out.println("anonymous popcorn");
} };
public void popIt() {
p.pop();  } }
```


Summary

- Inner Class
- Form of Inner Class
  - Regular Inner Class
  - Method-Local inner class
  - Anonymous inner class
  - Static nested class