**Inner classes**

1. A class declared inside another class is defined as inner class.
2. Introduced in 1.1v.
3. It was introduced to fix GUI bugs by Event Handling.

Without existing one type of object, if there is no chance of existing another type of Object then we should go for inner classes. Eg:

class University { class Car {  
  
 class Department { class Department {  
  
 } }  
} }

Java Eg:

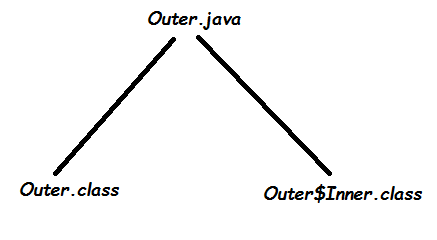
Entry is an inner interface inside Map interface.

interface Map {  
  
 interface Entry {  
  
 }  
}

Relationship between Outer class and inner class is **HAS-A** relationship and not **IS-A** relationship. So, it has **Composition or Aggregation**.

Categories:

1. Normal or Regular Inner classes.
2. Method Local Inner classes
3. Annonymous inner classes
4. **Static Nested class**.



***Note***: If we run any java program which is not having main method it will give NoSuchMethodFoundException.

class SomeClass {  
  
 class Inner {  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("Outer class main method");  
 }  
}

O/P:

Outer class main method

Inner classes can not contain static method including main method

class SomeClass {  
  
 class Inner {  
 public static void main(String[] args) {  
 System.*out*.println("Outer class main method");  
 }  
 }  
}

O/P

Compile time error: inner class cannot have static declarations.

class Outer {  
  
 class Inner {  
 public void m1() {  
 System.*out*.println("Inner class");  
 }  
 }  
  
 public static void main(String[] args) {  
 Outer outer = new Outer();  
 Outer.Inner inner = outer.new Inner();  
 inner.m1();  
  
 //OR  
  
 Outer.Inner inner1 = new Outer().new Inner();  
 inner1.m1();  
  
 //OR  
 new Outer().new Inner().m1();  
 }  
}

O/P

Inner class

Inner class

Inner class

class Outer {  
  
 class Inner {  
 public void m1() {  
 System.*out*.println("Inner class");  
 }  
 }  
  
 public void callInnerMethod() {  
 Inner inner = new Inner();  
 inner.m1();  
 }  
  
 public static void main(String[] args) {  
 Outer outer = new Outer();  
 outer.callInnerMethod();  
 }  
}

In the above we are directly creating the Object of Inner class in callInnerMethod() because Outer object is already created.

Hence

1. If we want to acces Inner class in instance area of Outer class we can directly use Inner inner = new Inner.
2. If we want to access Inner class in static area of outer class then we have to use Outer.Inner inner = new Outer().new Inner().

class Outer {

// static  
 public static void a() {  
 Inner inner = new Outer().new Inner();  
 inner.inner();  
 }  
  
 class Inner {  
 public void inner() {  
 System.*out*.println("This is inner class");  
 }  
 }  
  
 public static void main(String[] args) {  
 Outer outer = new Outer();  
 outer.*a*();  
 }  
}

**\*\*\*Inner class cannot have static declarations.**

But we can directly access static members of Outer class inside Instance method of inner class.

class Outer {  
  
 int a = 0;  
 static int *b* = 0;  
   
 class Inner {  
 public void m1() {  
 System.*out*.println(a);  
 System.*out*.println(*b*);  
 }  
 }  
}

**\*\*\*Important Example**

class Outer {  
  
 int a = 10; // Outer class a  
  
 class Inner {  
 int a = 100; // Inner class a  
  
 public void m1() {  
 int a = 1000; // Local method a  
  
 System.*out*.println(a); //1000  
 System.*out*.println(this.a); //100  
 System.*out*.println(Outer.this.a); //10  
 }  
 }  
}

**Applicable modifiers for Outer class.**

1. public
2. <default>
3. final
4. abstract
5. strictfp

**Applicable modifiers for Inner classes.**

1. public
2. <default>
3. final
4. abstract
5. strictfp
6. private
7. protected
8. static

**Nesting of Inner classes.**

Inside Inner class we can declare another Inner class.

class Outer {  
  
 class Inner {  
  
 class InnerMost {  
 public void m1() {  
 System.*out*.println("Innermost method");  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 Outer outer = new Outer();  
 Outer.Inner inner = outer.new Inner();  
 Outer.Inner.InnerMost innerMost = inner.new InnerMost();  
 innerMost.m1();  
 }  
}

**Method Local Inner class**

We cannot declare method inside method in java. We can use class inside method to achieve that. Mostly in cases where repeatations are required we use this concept.

class Outer {  
  
 public void m1() {  
  
 class MethodLocal {  
 public void sum(int a, int b) {  
 }  
 } // End of class MethodLocal  
  
 MethodLocal outer = new MethodLocal();  
 outer.sum(10, 10);  
 outer.sum(10, 60);  
 }// End of m1  
}

**valid**

class Outer {  
  
 int a =10;  
 static int *b* =10;  
 public void m1() {  
  
 class MethodLocal {  
 public void m2() {  
 System.*out*.println(a);  
 System.*out*.println(*b*);  
 }  
  
 } // End of class MethodLocal  
  
 MethodLocal outer = new MethodLocal();  
 outer.m2();  
 outer.m2();  
 }// End of m1  
}

\*\*We **can** access static as well as no static inside class which is inside **instance** method.

\*\*We **can** access only static inside class which is inside **static** method.

* public **static** void m1()

**Memory level issue**:

class Outer {

// Local Method  
 public void m1() {  
 int a = 10;  
  
 class MethodLocal {  
 public void m2() {  
 System.*out*.println(a);  
 }  
  
 } // End of class MethodLocal  
  
 MethodLocal methodLocal = new MethodLocal();  
 methodLocal.m2();  
 methodLocal.m2();  
 }// End of m1  
  
 public static void main(String[] args) {  
 Outer outer = new Outer();  
 outer.m1();  
 }  
}

From MethodLocal inner class we can’t access local variables of m1() method. When local method completes local variable will be gone. In the above **a=10** will be created inside stack and as soon as the local method completed local variable is destroyed.

Now, MethodLocal class’ Object is created in heap area. Whenever we are calling m1() method:

1. Local variable is created.
2. Inner class Object is created.

After method m1() executes first time we get the output and then after the method gets complete local variable a gets destroyed. Inner class is present in Heap area so it does not gets destroyed immediately and hence there may be a chance of calling m2() method of Inner class directly but the variable **‘a’** already got destroyed so value of a won’t be present.

In this case we can use final keyword as with final a will get replaced with the actual value.

Static Nested class

class Outer {  
  
 static class Inner{  
   
 public void a(){  
   
 }  
 }  
  
 public static void main(String[] args) {  
 Inner inner = new Inner();  
 inner.a();  
 }  
  
}

We do not need Outer class to use a() method of Inner class. We can directly use Inner class.

class Outer {  
  
 static class Inner {  
  
 public static void main(String[] args) {  
 System.*out*.println("Inner main");  
 }  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("outer");  
 }  
}

javac Outer.java

java Outer

O/P outer

javac Outer$Inner.java

java Outer$Inner

O/P inner

So, we can declare main method inside static nested class.

Questions:

1. Can we declare class inside interface
2. Can we interface inside interface
3. Can we declare interface inside class.

interface Outer {  
  
 void outer();  
  
 interface Inner {  
 void inner();  
 }  
}  
  
class TestOuter implements Outer {  
  
 @Override  
 public void outer() {  
 }  
}  
  
class TestInner implements Outer.Inner {  
  
 @Override  
 public void inner() {  
 }  
}

If we are implementing Inner interface we do not need to provide the implementations of Outer interface and vice versa.