Java: Inner Classes

Inner Class

- Like variables and methods, class can also be defined inside a class.
- An inner class is a class defined inside the scope of another class.
- Classes that were covered so far were top-level classes.
- The class inside which the inner class is defined is called outer class.
- Inner class can access even the private members of the outer class. Similarly outer class can also access the private members of inner class.
- Similar to inner class, inner interface can also be created.

Inner class example

- Inside the Outer class scope, Inner class name is same as what is declared in Outer class.
- Outside the Outer class scope, Inner class name is combination of Outer class name and Inner class name and a dot (.) separating them. Inner class name can be same as outer class name.
- The name of the inner class's .class file name:

  OuterClass$InnerClass.class

```
class OuterClass{
private static  int k;
InnerClass I;

private InnerClass(){int j;}
…
}
```

```
class SomeClass{
OuterClass.InnerClass c;
…
}
```

Types of Inner Class

- Member class
  - Static Inner Class/ Top-Level nested classes
  - Non Static Inner Class

- Local Inner Class
- Anonymous Class

- Non Static Inner Class, Local Inner Class, Anonymous Class are generally called inner class.
- Static inner class are considered to be top-level class.

Non static inner class

- Structure:

```
public class OuterClass{
public class InnerClass{..}
}
```

- Non static inner class object cannot be created without a outer class instance.
- The **`private`** fields and methods of the member classes are available to the enclosing class and other member classes
- All the **`private`** fields and methods of the outer classes are also available to inner class.
- Non-static inner class cannot have **`static`** members.
- Other modifier applicable here are **`abstract, final, public, protected, private`**

Example: Non static inner class instance

```
class C{
private int i;
static private int k;
void m(){
B b= new B();
b.j=10;
}
class B{
private int j;
static private int l;
void m(){
i=10;
k=15;
j=12;
}
}
}
```

Can access inner class private members

Compilation Error

Can access outer class private members

Creating instance of non static inner class outside the outer class

- Outside outer class non-static inner class creation requires outer class instance also.
- There are 2 ways to do this.
  - If you don't need outer class instance , then create it like line 1,
  - If you need outer class instance or already have one, create it like line 2

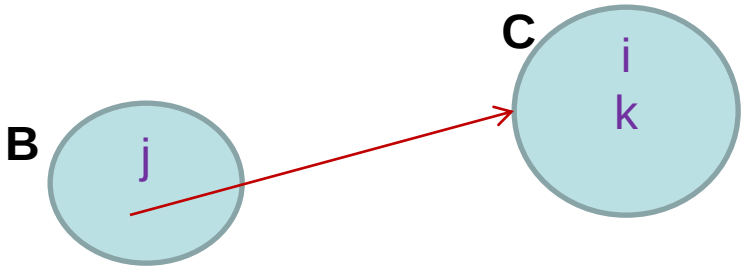
```
class A
{
C.B b= new C(). new B(); //line 1
C c= new C();
C.B b1= c. new B(); // line 2
}
```

  - If Inner class was defined in a package say p, then it can be created using the syntax:

```
new p.C(). new B();
```

Outer class implicit reference in inner class

- Non-static inner class instance cannot exist without Outer class instance.
- This inner class has implicit reference to the outer class object using which it is created.
- Therefore no explicit reference required in inner class for the outer class.
- However, if outer class needs a inner class reference it has to create it explicitly.



Tell me how?

- If inner class can be created only with outer class reference, then how is outer class able to create inner class?
  - Outer class creates inner class as instance member or locally inside instance method. In both of this context, the current object (this) is available. Hence the current object in context becomes the implicit reference for inner class instance.
  - Please note that the code below gives a compilation error because there is "No enclosing instance of type C is accessible".

```
class P{
static Q q= new Q(); //error
 Q q1= new Q(); //ok
 void f(){ Q q2= new Q(); } //ok
 static void g(){ Q q2= new Q(); } //error
class Q{}}
```

Name conflict

- If the name of the members in Outer class and inner class are same, then how to refer to the name of the outer class member in the inner class?
- This can be done using Outer class name dot (.) this dot (.) member name.

```
class P{
int i;
class Q{
int i;
void f(){
i=10;
P.this.i=9;
}
}}
```

Example 2: Static inner class in practice

- Here is a practical case where static inner class could be used.
- An AddressBook class encapsulating Address class.
- Address class encapsulates house address part – city, state etc. AddressBook encapsulates name, address and phone number. Please note that Address class is defined as private which means that only AddressBook will use this class.

Example 2

```
public class AddressBook{
    private class Address{
    private String name;
    private String houseNumber;
    private String street;
    private String city;
    private String state;
    private String pin;
    private Address(){}
    private Address(String a,String b,String c, String
    d,String e){
        this.name=AddressBook.this.name;
        houseNumber=a;street=b;
        city=c;state=d;pin=e;}
    public String toString(){
        return name+ ",\n"+houseNumber+ ",\n"+street+
        ",\n"+city+",\n"+state+ ",\nPIN:"+ pin;
    }}
```

name of the person is same as that of what was assigned in the Address object

```
String name;
Address addr;
String pno;

public AddressBook(String name,String pno, String ad)
{
    this.name=name;
    this.pno=pno;
    String s[]=ad.split(",");
    if(s.length>=5){
    String str="",":
    for(int i=1;i<s.length-3;i++)
    str=str+s[i]+",";  }
    addr=new Address(s[0],str,s[s.length-3],s[s.length-
    2], s[s.length-1]);
}
```

Address is considered valid only if it has 5 or more words separated by commas.

```
public String toString(){
   return addr.toString() + "\nPhone no.:"+pno;
}
public static void main(String str[]){
   AddressBook a=new
   AddressBook("Crossword","9008200021", "Icon Mall-
   2981,5th Cross,12th Main,Indira
   Nagar,Bangalore,560038");
   System.out.println(a);
}
}
```

Activity: Linked List

- Another situation where non-static inner class used is linked list.
- Can you figure down the skeleton of the code. (structure of the class, members etc.)

Hint:

A linked list is a chain of nodes. So you will have a Node class.

Where will you define this Node class?

To answer this, ask yourself if it is necessary for a LinkedList class to expose the Node class or is it just enough for it to provide public methods that returns the required data.

LinkedList Code outline

```
public class LinkedList{
private Node header = new Node(null, null, null);
public LinkedList(){...}
    public Object getFirst() {...}
    public Object getLast()  {...}
    public Object removeFirst() {...}
    public Object removeLast() {...}
    public void addFirst(Object Object) {...}
    public void addLast(Object Object) {...}
      …
    private static class Node {
      Object element;
      Node next;
      Node previous;
  Node(Object element, Node  next, Node previous) {
       this.element = element;
       this.next = next;
       this.previous = previous;      }     }}
```

Static Inner Class

- A static inner class is a class that's a **`static`** member of the outer class
- It can access only all **`static`** members of the outer class.
- But like **`main`** method, instances of outer class can be created inside **`static`** inner class and using this **`private`** members can be accessed.
- It is created without an instance of the outer class unlike the regular inner classes.
- That is why the **`static`** classes are sometimes called top-level nested classes.
- Other modifier applicable to member classes

```
abstract, final, public, protected, private
```

Syntax

- Structure:

```
public class OuterClass{
public static class InnerClass{}
}
```

- Creating instance:
  - Outside the outer class:

```
OuterClass.InnerClass sinner
=new OuterClass.InnerClass();
```

  - Inside the outer class:

```
InnerClass inner=new InnerClass();
```

Example: scenario for static inner class

- Let us say you have a class that maintains list of integers in sorted manner. How will you test this class (without a tool) ?
- May be you will do this.
- You will create at least 3 test cases – one that check for underflow, one for overflow and one for right number of inputs- as 3 methods of a class say TestSortedList class and a main method which calls these three methods.
- After you are satisfied, you submit SortedList.class and SortedList.java.
- Now let us say the code come back to for more additions, and alias you have lost the TestSortedList class somewhere… or you don't remember the name of the class!
- There is also another issue with this TestSortedList class, you cannot test the private methods, because they are not accessible!
- So the best way to do this would be to make TestSortedList a private static inner class.

```
public class SortedList{
private int size;
private int[] nums= new int[0];

private SortedList(int size)
{
    nums= new int[size];
}

private void insertInt(int temp) {
   int j = size++;
   while (j > 0 && nums[j-1] > temp){
      nums[j] = nums[j - 1];
      j--;}
   nums[j] = temp;
}
public void add(int i){insertInt(i);
}
```

Specify the size of the list during construction

Inserts numbers in sorted order

```
public String toString(){
    String s="";
    if (nums.length>0)
    for(int d:nums)
    s=s+d+ " ";
    return s;}
```

static inner class for unit testing

```
static class TestSortedList{
public static void main(String str[]){
    test0items();
    test3items();
    testoverflow();}

static void test0items(){
    SortedList list = new SortedList(0);
    list.insertInt(14);
    System.out.println(list);
    }
```

```
static void test3items(){
    SortedList list = new SortedList(3);
    list.add(11);
    list.add(4);
    list.insertInt(14);
    System.out.println(list);
}
static void testoverflow(){
    SortedList list = new SortedList(2);
    list.add(11);
    list.add(4);
    list.add(14);
    System.out.println(list);
}
}
}
```

Local Inner class defined

- An inner class that is defined inside a method is called local inner class (or method local inner class).
- A local inner class can be instantiated only by the method which defined it.
- Therefore no access specifier is applicable for the local inner class declaration. Only **`abstract`** and **`final`** modifiers are allowed.
- Also like other inner classes, local inner class can access all the members of the outer class including private members.
- Apart from the above, the local inner class can also access local variables which are **`final`**.
- ```
  class OuterClass {
   void someMethod(){
   class InnerClass{}
   }
  ```
  ```

Example

- In this example, we create a local inner class that will use sort method of Arrays class to sort student objects based on the name. (Recall **`NameSortStudent`** class that we created in the interface)
- In this case we create this class inside the sort method.

```
package student;
import java.util.*;
class ArraySortStudent{
public static void sortStudents(Student[] s){

class NameSort implements Comparator<Student>{
   public int compare(Student s1, Student s2){
   return s1.getName().compareTo(s2.getName());
}}
```

```
Arrays.sort(s,new NameSort());
    for(Student s1:s){
    System.out.println(s1);
}}
public static void main(String str[]){
    Student s[]= { new Student("Ram"), new
    Student("Bharat"), new Student("Lakshman")};
    sortStudents(s);
}}
```

How many objects of **`NameSort()`** *are we going to create?*
Only one in this cases. And we don't seem to require a named object.
We just created an unnamed object on the fly.
Also the only purpose that we created this class was to create this one-time object that will carry the implementation of interface method.

Java provides better syntax to do the same thing as above in the form of anonymous classes. And the cool thing is this syntax can be used any where not just inside the method!

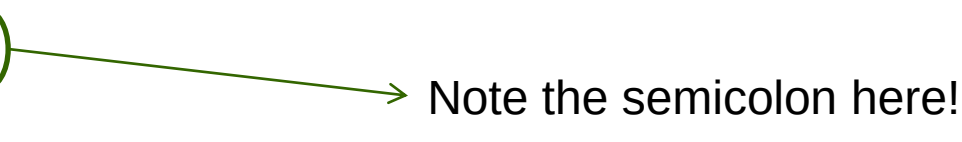
Anonymous Inner Classes

- Inner class without a class name is an anonymous inner class.
- Allows creation of one time use object !
- Anonymous inner class can be created either inside a method or outside a method. It is implicitly **`final`**.
- No modifier is allowed anywhere in the class declaration
- Also declaration cannot have an **`implements`** or **`extends`** clause.
- No constructors can be defined.
- An anonymous inner class is either inherited from an interface or from a class and so polymorphism is applicable. It cannot inherit from more than one class directly.

Syntax

- General way to create an anonymous inner class:

```
class OuterClass{
…
SomeClassOrInterface s
= new SomeClassOrInterface(){
//  overridden methods
};
}
```



Example1: Anonymous Inner Classes

Let us change the sort method of the previous example to use an anonymous inner class

```
public static void sort(Student[] s){
class NameSort implements Comparator{
Arrays.sort(
s,new Comparator<Student>(){
public int compare(Student s1, Student s2){
    return s1.getName().compareTo(s2.getName());
    }}
    );

    for(Student s1:s){
    System.out.println(s1);
}
}
```

Example2: Anonymous Inner Classes

Overriding a method of concrete class.
Note the way the this is created. It is important to make sure that the right constructors are called.

```
class Test{
static {
   HOD h=new HOD("Rana",""){
   public void display(){
   System.out.println("Name "+getName()); }

   };
}
}
```

Test your understanding

```
class E {
E() {
System.out.print("E");}
static class Z {
Z(){
System.out.print("Z");}}
public static void main(String args[]){
new E.Z();}}
```

What is the result of attempting to compile and run the program?

Test your understanding

```
class B {
  private static String s1 = "s1";
  final String s2 = "s2";
  B () {new Z("s5","s6");}
  static class Z {
    final String s3 = "s3";
    static String s4 = "s4";
    Z (final String s5, String s6) {
      System.out.print(???);
  }}
  public static void main(String args[]) {new B();}}
```

Which variable (in red) cannot be substituted for “???” without causing a compile-time error?

Test your understanding

```
class F {
public void m1() {Z.m1();}
private static class Y {
    private static void m1() {
      System.out.print("Y.m1 ");}
}
private static class Z {
    private static void m1(){
      System.out.print("Z.m1 ");
      Y.m1();
}}
public static void main(String[] args) {
new F().m1();}}
```

What is the result of attempting to compile and run the program?

Test your understanding

```
class Outer {
  static class StaticNested {
    static final int a = 25; // 1
    static final int b;      // 2
    static int c;            // 3
    int d;                   // 4
    static {b = 42;}         // 5
  }
  class NonStaticInner {
    static final int e = 25; // 6
    static final int f;      // 7
    static int g;            // 8
    int h;                   // 9
    static {f = 42;}         // 10
}}
```

Compile-time errors are generated at which lines?

Inner class in interface and vice versa

- A class can be nested inside an interface. Though this is allowed in java, it is a bad practice to include implementation inside abstraction.
- A interface can be nested inside a class (or an interface). This is a very rarely used feature.