**Inner Classes – Part-02**

* **Case\_04:**

From normal or regular inner-class we can access both static and non-static members of outer class directly.

class Outer{

int x = 10;

static int y = 20;

class Inner{

public void m1(){

System.out.println(x);

System.out.println(y);

}

}

public static void main(String[] args){

new Outer().new Inner().m1();

}

}

**Output:**

**10**

**20**

* **Case\_05:**

Within the inner-class this always refers current inner-class object.

If we want to refer current outer class object, we have to use

OuterClassName.this

class Outer{

int x =10;

class Inner{

int x = 100;

public void m1(){

int x = 1000;

System.out.println(x); // 1000

System.out.println(this.x); //100 Line-01

System.out.println(Outer.this.x);

}

}

public static void main(String[] args){

new Outer().new Inner().m1();

}

}

Note: Line-01 can be done in this way also.

Inner.this.x

* **Allowed Modifiers:**

The only applicable modifiers for outer classes are:

public

default

abstract

final

strictfp

But for inner classes applicable modifiers are

public

default

abstract

final

strictfp

private

protected

static

* **Nesting of Inner-Classes:**

Inside inner-class, we can declare another inner class that is nesting of inner-classes is possible

class A{

class B{

class C{

public void m1(){

System.out.println(“Innermost class method”);

}

}

}

}

class Test{

public static void main(String[] args){

A a = new A();

A.B b = a.new B();

A.B.C b = b.new C();

}

}

* **Method Local Inner Classes:**

Sometimes we can declare a class inside a method, such type of inner classes is called “Method Local Inner Classes”.

The main purpose of method local inner-class is to define method specific repeatedly required functionality.

Method local inner-classes are best suitable to meet nested method requirements.

We can access method local inner-classes only within the method where we declared. Outside of the method we can’t access. Because of its less scope, method local inner classes are most rarely used type of inner-classes.

Example:

class Test{

public void m1(){

class Inner{

public void sum(int x, int y){

System.out.println(“The sum:”+(x+y));

}

}

Inner i = new Inner();

i.sum(10, 20);

i.sum(100, 200);

i.sum(1000, 2000);

}

public static void main(String[] args){

Test t = new Test();

t.m1();

}

}

Output:

The sum: 30

The sum: 300

The sum: 3000

We can declare method local inner class inside both instance and static methods.

If we declare inner class inside instance method, then from that method local inner class, we can access both static and non-static members of outer class directly.

If we declare inner class inside static method, then we can access only static members of outer class directly from that method local inner-class.

class Test{

int x;

static in y = 20;

public void m1(){

class Inner{

public void m2(){

System.out.println(x); // Line-01

System.out.println(y);

}

}

Inner i = new Inner();

i.m2();

}

}

Output:

10

20

Note:

If we declare m1() method as static, then at Line-01, we will get compile time error saying

Non-static variable x cannot be referenced from a static context.

* **Accessing Local Variable of Method Local Inner Class:**

From method local inner class we can’t access local variables of the method in which we declared inner class.

If the local variable as declared as final, then we can access.

class Test{

public void m1(){

int x = 10; // Line-01

class Inner{

public void m2(){

System.out.println(x);

}

}

Inner i = new Inner();

i.m2();

}

public static void main(String[] args){

Test t = new Test();

t.m1();

}

}

**Output:**

CE: local variable x is accessed from within inner class; needs to be declared final

Note:

If we declare x as final then we won’t get any compile time error.

Reason:

Local variables will be created in the method stack when the method starts its execution, and it will be deleted once the method completes.

Objects will be created on the heap, there may be a chance even though the method which has this inner classes is destroyed, but still the object is there on the heap. If we are calling the m2() method directly without calling m1() method, m2() method depends on the local variable x, which will not be there in this case. So, this is not legal.

When we declare the any variable as final, all the values will be replaced at compile time itself, in the above case, instead of x, the SOP will have the value 10, so even if we call method m2() directly from inner object, there is no dependency on variable x.

* **Question\_01:**

Consider the following code

class Test{

int i = 10;

static int j = 20;

public void m1(){

int k = 30;

final int m = 40;

class Inner{

public void m2(){

Line-01

}

}

}

}

At Line-01 which of the following variables we can access directly?

i, j, k, m

Answer: I, j, m

* **Question\_02:**

If we declare m1() as static then at Line-01 which variables we can access directly?

i, j, k, m

Answer: j, m

* **Question\_03:**

If we declare m2() method as static, then at Line-01, which variables we can access directly?

i, j, k, m

Answer: We will get compile time error, because we can’t declare static members inside inner classes.

* **Applicable Class Level Modifiers:**

The only applicable modifiers for method local inner classes are

final

abstract

strictfp

If we are trying to apply any other modifier. Then we will get compile time error.