Inner Classes

* sometimes we can declare a class inside another class such type of classes are called inner classes
* inner classes concept introduced in 1.1 version to fix GUI bugs as part of event handling but because of powerful features and benefits of inner classes slowly programmer started using in regular coding also.
* without existing one type of object if there is no chance of existing another type of object then we should go for inner classes.
* university consist of several departments without existing university there is no chance of existing department hence we have to declare department class inside university class.

eg. class university {

class department

{

}

}

- without existing car object there is no chance of engine object hence we have to declare engine class inside car class.

eg. class car{

class engine{

}

}

eg. Map is a group of key value pair and each key value pair is call an entry without existing map object there is no chance of existing entry object hence interface entry is defined inside map interface

Interface Map{

Interface Entry{

}

}

* without existing outer class object there is no chance of existing inner class object.
* the relation between outer class and inner class is not "is a relation" and it is "has a relation"(composition or aggregation)
* based on position and behavior all inner classes divided into 4 types

1. Normal and regular inner classes
2. Method local inner classes
3. Anonymous inner classes
4. Static nested classes
5. **Normal and regular inner classes :**

* if we are any named class declaring inside a class without static modifier such type of inner class is call normal or regular inner class.

e. g 1 class outer {

class inner {

}

}

e. g 2 class outer {

class inner {

}

public Static void main(String[] args) {

sysout("outer class");

}

}

compile javac outer.java

compile successfully and o/p will be outer class

compile javac outer$inner.java

compile time error saying NoSuchMethod "main"

* inside inner class we can’t declare any static members hence we can’t declare main () and we can’t run inner class directly from command prompt

e.g class outer{

class inner {

public Static void main(String[] args) {

sysout("outter class");

}

}

}

* at the time of compilation: error will occur saying : inner class cannot have static declaration

**case 1** : accessing inner class code from static area of outer class.

e.g public class OuterClass {

public class InnerClass {

public void mi(){

System.out.println("inner class");

}

}

public static void main(String[] args) {

OuterClass out = new OuterClass();

InnerClass in = out.new InnerClass();

OuterClass.InnerClass i = new OuterClass().new InnerClass();

new OuterClass().new InnerClass().mi();//shortcut

}

}

**case 2 :** accessing inner class code from instance area of outer class

e.g public class OuterClass {

public class InnerClass {

public void mi()

{

System.out.println("inner class");

}

}

public void m1(){

InnerClass in = new InnerClass();

in.mi();

}

public static void main(String[] args) {

OuterClass out = new OuterClass();

InnerClass in = out.new InnerClass();

OuterClass.InnerClass i = new OuterClass().new InnerClass();

out.m1();

new OuterClass().new InnerClass().mi();//shortcut

}

}

**case 3 :** accessing inner class code outside of outer class

- same as case one but the main() will be in different class(anther outside class)

- from normal or regular class we can access both static or non-static member directly

e.g

public class OuterClass {

int a= 10;

static int b= 20;

public class InnerClass {

public void mi(){

System.out.println(a);

System.out.println(b);

System.out.println("inner class");

}

}

}

* within the inner class this always refer to current inner class object if we want to refer current outer class object we have to use outer class name .this

e.g public class OuterClass {

int a= 10;

public class InnerClass {

int a= 100;

public void mi(){

int a= 1000;

System.out.println(a);

System.out.println(this.a);

System.out.println(OuterClass.this.a);

}

}

public static void main(String[] args) {

OuterClass out = new OuterClass();

OuterClass.InnerClass in = out.new InnerClass();

in.mi();

}

}

**o/p** 1000

100

10

- for outer class default, public, abstract, final and strickfp modifier are allowed

- for inner class default, public, abstract, final, strickfp, private, protected and static modifier are allowed

* **nesting of inner class**
* inside a inner class we can declare inner class so nesting of inner class is possible.

eg public class OuterClass {

public class InnerClass {

public class InnerClassOfInner {

public void mi(){

System.out.println("InnerClassOfInner class ");

}

}

}

public static void main(String[] args) {

OuterClass out = new OuterClass();

OuterClass.InnerClass in = out.new InnerClass();

OuterClass.InnerClass.InnerClassOfInner inin = in.new InnerClassOfInner();

inin.mi();

}

}

1. **method local inner classes:**

* sometimes we can declare a class inside a method such type of inner classes are 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 is best suitable to meet nested method requirement.
* we can access method local inner classes only within a method where we declare outside of the method we can’t access because of its less scope
* method local inner classes are most really used type of inner classes

e.g

public class OuterClass {

public void mi(){

class InnerClass {

public int sum(int a, int b){

return a+b;

}

}

InnerClass i = new InnerClass();

System.out.println(i.sum(10, 20));

System.out.println(i.sum(20, 30));

}

public static void main(String[] args) {

OuterClass out = new OuterClass();

out.mi();

}

}

**o/p** 30

20

* we can declare method local inner class inside both instance and static methods.
* if we declare inner class inside instance method then form that method local inner class we can access both static and non-static member of outer class directly
* if we declare inner class inside static method the we can access only static members of outer class directly from that method local inner class.

e.g public class OuterClass {

int a = 10;

static int b =20;

public static void mi(){

class InnerClass {

public void sum(){

System.out.println(a); // line 1

System.out.println(b);

}

}

InnerClass i = new InnerClass();

i.sum();

}

public static void main(String[] args) {

OuterClass out = new OuterClass();

out.mi();

}

}

* if declare m1() as static then at line 1 we will get compile time error saying non-Static Variable cannot be referenced from a static context
* from method local inner class we can’t access local variables of the method in which we declare inner class if the local variable declared as final then we can access
* the only applicable modifier for method inner local classes are final ,abstract and strictfp.
* if we are trying to apply any other modifier then we will get compile time error

consider following code

class test{

int i =10;

static int j =20;

public void m1(){

int k =30;

final int m =40;

class inner{

line 1.

}

}

}

question 1 :

at line 1 which of the variable we can access directly among i,j,k and m?

Ans: i,j and m

question 2 :

if we declare m1() as static the at line 1 which variables we can access directly among i,j and k, and m?

Ans : we will get compile time error because we can’t declare Static members inside inner classes

1. **Anonymous inner classes :**

* sometimes we can declare inner class without name such type of inner classes are called Anonymous inner classes.
* the main purpose of Anonymousinner classes is just for instance use (one time usage)
* based on declaration and behavior there are three types of anonymous inner class

1. anonymous inner class that extends a class
2. anonymous inner class that implements an interface
3. anonymous inner class that define inside arguments
4. **anonymous inner class that extends a class**

public class AnonymousClass {

public static void main(String[] args) {

Anonymous2 an = new Anonymous2(){

public void m1(){

System.out.println("spicy");

}

};

System.out.println(an.getClass().getName());

an.m1();

Anonymous2 an1 = new Anonymous2();

an1.m1();

}

}

* generated .class file are AnonymousClass.class ,AnonymousClass2.class and AnonymousClass$1.class
* **analysis**

1) popcorn p = new porpcorn();

just we are creating popcorn object

2) popcorn p = new porpcorn(){};

- we are declaring a class that extends popcorn class without name (Anonymous inner class).

in that child class we are overriding m().

* for that child class we creating a object with parent reference
* we are declaring a class that extends popcorn

public static void main(String[] args) {

Runnable r = new Runnable() {

@Override

public void run() {

for(int i=0;i<10;i++)

System.out.println("Anonymous");

}

};

Thread t = new Thread(r);

t.start();

for(int i=0;i<10;i++)

System.out.println("main");

}

}

2) **anonymous inner class that implements an interface**

**public class AnonymousClass {**

**public static void main(String[] args) {**

**Runnable r = new Runnable() {**

**@Override**

**public void run() {**

**for(int i=0;i<10;i++)**

**System.out.println("Anonymous");**

**}**

**};**

**Thread t = new Thread(r);**

**t.start();**

**for(int i=0;i<10;i++)**

**System.out.println("main");**

**}**

**}**

**defining a thread by extending thread class**

* anonymous inner class that implements an interface
* defining a thread by implementing runnable interface

**3) anonymous inner class that defins inside of arragument**

public class AnonymousClass {

public static void main(String[] args) {

new Thread(new Runnable() {

@Override

public void run() {

for(int i=0;i<10;i++)

System.out.println("Anonymous");

}

}).start();

for(int i=0;i<10;i++)

System.out.println("main");

}

}

* **where anonymous inner class are best suitable**
* we can use anonymous inner classes frequently in GUI based applications to implement event handling
* difference between Normal java class and anonymous inner class

|  |  |
| --- | --- |
| **Normal java class** | **anonymous inner class** |
| A normal java class can extend only one class at time | A anonymous inner class also can extend only one class at time |
| Normal java class can implement any number of interface simultaneously | anonymous inner class can implement only one interface at a time |
| A normal java class can extend a class and can implement any number of interface simultaneously | But anonymous inner class can extend a class or can implement an interface but not both simultaneously |
| In normal java class we can create any number of constructors simultaneously. | But in anonymous inner classes we cant write any constructor explicitly because name of the class and name of the constructor must be same but anonymous inner classes not having any name |
| If the requirement is standard and required several times then we should go for normal java class | If the requirement is temporary and the required only once(instance use) then we anonymous inner class |

1. **Static nested classes**

* sometimes we can declare inner class with static modifier such type of inner classes are called static nested classes
* in the case of normal or regular class without existing outer class object there is no chance of existing inner class object i.e inner class object is strongly associated with outer class object.
* but in the case of static nested classes without existing outer class object there may be a chance of existing nested class object hence static nested class object is not strongly associated with outer class object.

e.g1 public class AnonymousClass {

static class inner{

public void m1()

{

System.out.println("Static class method");

}

}

public static void main(String[] args) {

inner in = new inner();

in.m1();

}

}

* if you want to create nested class object from outside of outer class then we can create as follows.
* OuterClass.InnerClass in = new OuterClass.InnerClass(); or InnerClass in1 = new InnerClass();
* in normal or reguler classes we can’t declare any static members but in static nested classes we can declare static members including main method hence we can invoke static nested class directly from console

public class AnonymousClass {

static class inner{

public void m1(){

System.out.println("Static class method");

}

}

public static void main(String[] args) {

inner in = new inner();

in.m1();

System.out.println("outer class method");

}

}

o/p

javac AnonymousClass.java

java AnonymousClass

o/p : outer class method

java AnonymousClass$inner

o/p : Static class method

* from normal or regular inner classes we can access both static and nonstate of outer class directly but from static nested classes we can access static members of outer class directly and we can’t access nonstate members.

public class OuterClass {

int a =10;

static int b =20;

static class inner{

public void m1(){

System.out.println(a);

System.out.println(b);

}

}

}

exception : Cannot make a static reference to the non-static field a

* **difference between normal or regular class and static nested class**

|  |  |
| --- | --- |
| Normal inner java class | Static nested class |
| Without existing outer class object there is no chance of existing inner class object i.e inner class object strongly associated outer class object | Without existing outer class object there may be a chance of existing static nested class object i.e i.e static nested class is not strongly associated with outer class object |
| In normal inner class we cant create static members | In static nested class we can create static members |
| In normal inner class We cant declare main method hence we cant invoke inner class directly from command prompt | In static nested classes We can declare main method hence we can invoke nested class directly from command prompt |
| From normal inner classes we can access both non-static and static members of outer class directly in inner class | From static nested classes We can access only static members of outer class directly in inner class |

* **various combinations of nested classes and interface**

**case 1 : class inside a class**

* without existing one type of object if there is no chance of existing another type of object then we can declare class inside a class

**e.g** : university consist of several department without existing university there is no chance of existing department hence we have to declare department class inside university class.

class university{

class department{

}

}

**case 2 interface inside a class:**

* inside a class if we require multiple implementations of an interface and all these implementations are related to a particular class then we can define interface inside a class.

e.g

class vehicleType{

interface vehicleType{

public int getNumberOfWhilees();

}

class bus impliments vehicleType{

public int getNumberOfWhilees(){

return 6;

}

}

class auto impliments vehicleType{

public int getNumberOfWhilees(){

return 3;

}

}

}

**case 3: interface inside interface**

* we can declare interface inside interface.

e.g a map is group of key value pairs and each key value pair is called an entry. without existing map object there is no chance of existing entry object hence interface entry is defined inside map interface

- every interface present inside interface is always public and static whether we are declaring or not hence we can implement inner interface directly without implementing outer interface.

- similarly whenever we are implementing outer interface we are not required to implement inner interface i.e. we can implement outer and inner interface independently

e.g

interface outterInterface {

public void m1();

interface InnerInterface {

public void m2();

}

}

public class ImlimentationOfOutterInterface implements outterInterface{

@Override

public void m1() {

System.out.println("imlimenting outter interface");

}

}

public class ImlimentationOfInnerInterface implements outterInterface.InnerInterface{

@Override

public void m2() {

System.out.println("imlimenting inner interface");

}

}

public class MainClass {

public static void main(String[] args) {

ImlimentationOfOutterInterface out = new ImlimentationOfOutterInterface();

out.m1();

ImlimentationOfInnerInterface in = new ImlimentationOfInnerInterface();

in.m2();

}

}

**case 4: class inside interface :**

* if functionality of a class closely associated with interface then it is highly recomanded to declair a class inside interface

e.g

interface email {

public void mail();

class emailDetail{

String to;

String from;

String cc;

String subject;

String Body;

}

}

* in the above example email detail is require only for email service and we are not using anywhere else hence emailDetails class recommended to declare inside email interface
* we can also implement a class inside interface to provide default implementation for that interface.

e.g :

public interface Vehicles {

public int getWheels();

class defaultVehicle implements Vehicles{

@Override

public int getWheels() {

return 2;

}

}

}

public class Bus implements Vehicles{

@Override

public int getWheels() {

return 6;

}

}

public class ThesVehicles {

public static void main(String[] args) {

Vehicles.defaultVehicle dv = new Vehicles.defaultVehicle();

System.out.println(dv.getWheels());

Bus bus = new Bus();

System.out.println(bus.getWheels());

}

}

o/p : 2 6

* in the above example default vehicle is the default implementation of vehicle interface whereas bus is customized implementation of vehicle interface.

**Note :**

* the class which is declared inside interface is always public static whether we are declaring or not hence we can create class object directly without having outer interface type object.

**Conclusions:**

* among classes and interfaces we can declare anything inside anything
* the interface which is declare inside interface is always public and static whether we are declaring or not
* the class which is declare inside interface is always public and static whether we are declaring or not
* the interface which is declare inside class is always static but need not be public whether we are declaring or not

**Anonymous class vs Lambda Expression**

* An anonymous class object generates a separate class file after compilation that increases the size of a jar file while a lambda expression is converted into a private method. It uses invokedynamic bytecode instruction to bind this method dynamically, which saves time and memory.
* We use this keyword to represent the current class in lambda expression while in the case of an anonymous class, this keyword can represent that particular anonymous class.
* Anonymous classes can be used in case of more than one abstract method while a lambda expression specifically used for functional interfaces.
* We need to provide the function body only in lambda expression while in the case of an anonymous class, we need to write the redundant class definition.