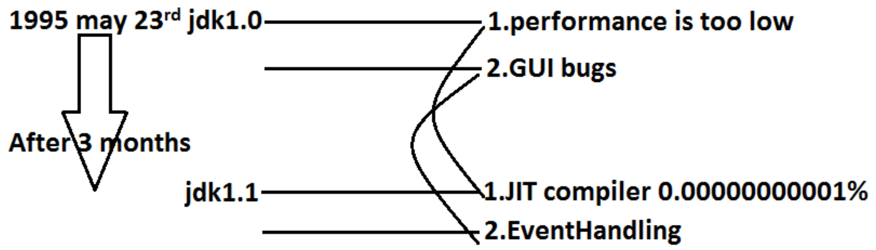
Inner Classes

- Sometimes we can declare a class inside another class such type of classes are called inner classes.

Diagram:



- Sun people introduced inner classes in 1.1 version as part of "**EventHandling**" to resolve GUI bugs.
- But because of powerful features and benefits of inner classes slowly the programmers starts 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.

Example: Without existing University object there is no chance of existing Department object hence we have to define Department class inside University class.

Example1:

```
class University————————Outer class
{
     class Department————inner class
     {
     }
}
```

Example 2: Without existing Bank object there is no chance of existing Account object hence we have to define Account class inside Bank class.

Example:

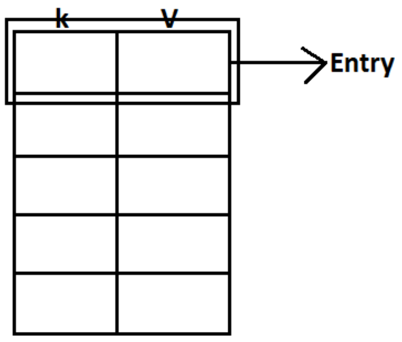
```
class Bank————————————Outer class
{
     class Account————inner class
     {
     }
}
```

Example 3: Without existing Map object there is no chance of existing Entry object hence Entry interface is define inside Map interface.

Example:

```
interface Map————outer interface
{
      interface Entry——inner interface
      {
      }
}
```

Diagram:



Note: The relationship between outer class and inner class is not IS-A relationship and it is Has-A relationship.

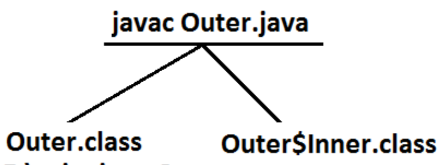
- Based on the purpose and position of declaration all inner classes are divided into 4 types. They are:
  1) **Normal or Regular inner classes**
  2) **Method Local inner classes**
  3) **Anonymous inner classes**
  4) **Static nested classes.**

1. Normal (or) Regular inner class: If we are declaring any named class inside another class directly without static modifier such type of inner classes are called normal or regular inner classes.

Example:

```
class Outer
{
        class Inner
        {
        }
}
```

Output:

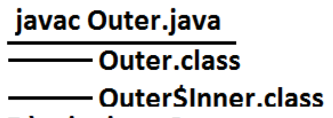


```
E:\scjp>java Outer
Exception in thread "main" java.lang.NoSuchMethodError: main
E:\scjp>java Outer$Inner
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Example:

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

Output:



```
E:\scjp>java Outer
outer class main method
E:\scjp>java Outer$Inner
Exception in thread "main" java.lang.NoSuchMethodError: main
```

- Inside inner class we can't declare static members. Hence it is not possible to declare main() method and we can't invoke inner class directly from the commend prompt.

Example:

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

Output:

```
E:\scjp>javac Outer.java
Outer.java:5: inner classes cannot have static declarations
        public static void main(String[] args)
```

Accessing inner class code from static area of outer class:

Example:

```
class Outer
{
        class Inner
        {
                public void methodOne(){
                        System.out.println("inner class method");
                }
        }
        public static void main(String[] args)
        {
                Outer o=new Outer();
                Outer.Inner i=o.new Inner();     (or)
                i.methodOne();
                Outer.Inner i=new Outer().new Inner();
                i.methodOne();                            (or)

                new Outer().new Inner().methodOne();
        }
}
```

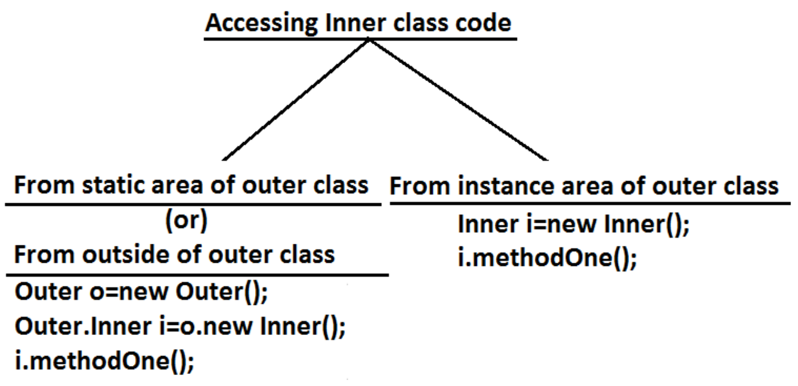
Accessing inner class code from instance area of outer class:

Example:

```
class Outer
{
        class Inner
        {
                public void methodOne()
                {
                        System.out.println("inner class method");
                }
        }
        public void methodTwo()
        {
                Inner i=new Inner();
```

```
		i.methodOne();
	}
	public static void main(String[] args)
	{
		Outer o=new Outer();
		o.methodTwo();
	}
}
```
Output:
```
E:\scjp>javac Outer.java
E:\scjp>java Outer
Inner class method
```
Accessing inner class code from outside of outer class:

Example:
```
class Outer
{
	class Inner
	{
		public void methodOne()
		{
			System.out.println("inner class method");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		new Outer().new Inner().methodOne();
	}
}
```
Output:
```
Inner class method
```


Accessing Inner class code
From static area of outer class
(or)
From outside of outer class
Outer o=new Outer();
Outer.Inner i=o.new Inner();
i.methodOne();
From instance area of outer class
Inner i=new Inner();
i.methodOne();

- From inner class we can access all members of outer class (both static and non-static, private and non private methods and variables) directly.

Example:

```
class Outer
{
        int x=10;
        static int y=20;
        class Inner{
                public void methodOne()
                {
                        System.out.println(x);//10
                        System.out.println(y);//20
                }
        }
        public static void main(String[] args)
        {
                new Outer().new Inner().methodOne();
        }
}
```

- Within the inner class ”**this**” always refers current inner class object. To refer current outer class object we have to use “**outer class name.this**”.

Example:

```
class Outer
{
        int x=10;
        class Inner
        {
                int x=100;
                public void methodOne()
```

```
            {
                int x=1000;
                System.out.println(x);//1000
                System.out.println(this.x);//100
                System.out.println(Outer.this.x);//10
            }
        }
        public static void main(String[] args)
        {
            new Outer().new Inner().methodOne();
        }
}
```

- The applicable modifiers for outer classes are:
  1) **public**
  2) **default**
  3) **final**
  4) **abstract**
  5) **strictfp**
- But for the inner classes in addition to this the following modifiers also allowed.

<u>Diagram:</u>

public		**private**
default		**protected**
final	**+**	**static[static nested classes]**
abstract		
strictfp		

<u>Method local inner classes:</u>

- Sometimes we can declare a class inside a method such type of inner classes are called method local inner classes.
- The main objective of method local inner class is to define method specific repeatedly required functionality.
- We can access method local inner class only within the method where we declared it. That is from outside of the method we can't access. As the scope of method local inner classes is very less, this type of inner classes are most rarely used type of inner classes.

<u>Example:</u>

```
class Test
{
        public void methodOne()
```

```
	{
		class Inner
		{
			public void sum(int i,int j)
			{
				System.out.println("The sum:"+(i+j));
			}
		}
		Inner i=new Inner();
		i.sum(10,20);
		;;;;;;;;;;;;
		i.sum(100,200);
		;;;;;;;;;;;;;;
		i.sum(1000,2000);
		;;;;;;;;;;;;;;;;
	}
	public static void main(String[] args)
	{
		new Test().methodOne();
	}
}
```

Output:

```
The sum: 30
The sum: 300
The sum: 3000
```

- If we are declaring inner class inside instance method then we can access both static and non static members of outer class directly.
- But if we are declaring inner class inside static method then we can access only static members of outer class directly and we can’t access instance members directly.

Example:

```
class Test
{
	int x=10;
	static int y=20;
	public void methodOne()
	{
		class Inner
		{
```

```
                public void methodTwo()
                {
                        System.out.println(x);//10
                        System.out.println(y);//20
                }
        }
        Inner i=new Inner();
        i.methodTwo();
    }
    public static void main(String[] args)
    {
        new Test().methodOne();
    }
}
```

- If we declare methodOne() method as static then we will get compile time error saying "non-static variable x cannot be referenced from a static context".
- From method local inner class we can't access local variables of the method in which we declared it. But if that local variable is declared as final then we won't get any compile time error.

Example:

```
class Test
{
    int x=10;
    public void methodOne()
    {
        int y=20;
        class Inner
        {
                public void methodTwo()
                {
                        System.out.println(x);//10
                        System.out.println(y); //C.E: local variable y is accessed from
within inner class; needs to be declared final.
                }
        }
        Inner i=new Inner();
        i.methodTwo();
    }
```

```
        public static void main(String[] args)
        {
                new Test().methodOne();
        }
}
```

- If we declared y as final then we won't get any compile time error.
- Consider the following declaration.

```
class Test
{
        int i=10;
        static int j=20;
        public void methodOne()
        {
                int k=30;
                final int l=40;
                class Inner
                {
                        public void methodTwo()
                        {
                                System.out.println(i);  \
                                System.out.println(j);   } line 1
                                System.out.println(k);   |
                                System.out.println(l);  /
                        }
                }
                Inner i=new Inner();
                i.methodTwo();
        }
        public static void main(String[] args)
        {
                new Test().methodOne();
        }
}
```

- **At line 1 which of the following variables we can access?**
  [✓ 1)] i   [✓ 2)] j   [✗ 3)] k   [✓ 4)] l
- **If we declare methodOne() method as static then which variables we can access at line 1?**

~~1)~~ i ✓2) j ~~3)~~ k ✓4) l

- If we declare methodTwo() as static then we will get compile time error because we can't declare static members inside inner classes.
- The only applicable modifiers for method local inner classes are:
  1) **final**
  2) **abstract**
  3) **strictfp**
- By mistake if we are declaring any other modifier we will get compile time error.

<u>Anonymous inner classes:</u>

- Sometimes we can declare inner class without name such type of inner classes are called anonymous inner classes.
- The main objective of anonymous inner classes is "**just for instant use**".
- There are 3 types of anonymous inner classes

1) **Anonymous inner class that extends a class.**
2) **Anonymous inner class that implements an interface.**
3) **Anonymous inner class that defined inside method arguments.**

<u>Anonymous inner class that extends a class:</u>

```
class PopCorn
{
	public void taste()
	{
		System.out.println("spicy");
	}
}
class Test
{
	public static void main(String[] args)
	{
		PopCorn p=new PopCorn()
		{
			public void taste()
			{
				System.out.println("salty");
			}
		};
		p.taste();//salty
		PopCorn p1=new PopCorn();
```

```
                p1.taste();//spicy
        }
}
```
Analysis:

1) **PopCorn p=new PopCorn();**

- We are just creating a PopCorn object.

2) **PopCorn p=new PopCorn()**
```
{
};
```
- We are creating child class without name for the PopCorn class and for that child class we are creating an object with Parent PopCorn reference.

3) **PopCorn p=new PopCorn()**
```
{
        public void taste()
        {
        System.out.println("salty");
        }
};
```
1) We are creating child class for PopCorn without name.
2) We are overriding taste() method.
3) We are creating object for that child class with parent reference.

Note: Inside Anonymous inner classes we can take or declare new methods but outside of anonymous inner classes we can't call these methods directly because we are depending on parent reference.[parent reference can be used to hold child class object but by using that reference we can't call child specific methods]. These methods just for internal purpose only.

Example 1:
```
class PopCorn
{
        public void taste()
        {
                System.out.println("spicy");
        }
}
class Test
{
        public static void main(String[] args)
        {
                PopCorn p=new PopCorn()
                {
                        public void taste()
                        {
                                methodOne();//valid call(internal purpose)
                                System.out.println("salty");
                        }
                        public void methodOne()
                        {
                                System.out.println("child specific method");
                        }
                };
                //p.methodOne();//here we can not call(outside inner class)
                p.taste();//salty
                PopCorn p1=new PopCorn();
                p1.taste();//spicy
        }
}
```

Output:

```
Child specific method
Salty
Spicy
```

Example 2:

```
class Test
{
        public static void main(String[] args)
        {
                Thread t=new Thread()
                {
                        public void run()
                        {
                                for(int i=0;i<10;i++)
                                {
                                        System.out.println("child thread");
                                }
                        }
                };
                t.start();
                for(int i=0;i<10;i++)
```

```
				{
					System.out.println("main thread");
				}
		}
}
```
Anonymous Inner Class that implements an interface:

Example:

```
class InnerClassesDemo
{
		public static void main(String[] args)
		{
				Runnable r=new Runnable()//here we are not creating for Runnable interface,
we are creating implements class object.
				{
					public void run()
					{
						for(int i=0;i<10;i++)
						{
							System.out.println("Child thread");
						}
					}
				};
				Thread t=new Thread(r);
				t.start();
				for(int i=0;i<10;i++)
				{
					System.out.println("Main thread");
				}
		}
}
```
Anonymous Inner Class that define inside method arguments:

Example:

```
class Test
{
		public static void main(String[] args)
		{
				new Thread(
					new Runnable()
		{
			public void run()
			{
				for(int i=0;i<10;i++)
				{
					System.out.println("child thread");
				}
			}
		}).start();
		for(int i=0;i<10;i++)
		{
			System.out.println("main thread");
		}
	}
}
```

Output:

- This output belongs to example 2, anonymous inner class that implements an interface example and anonymous inner class that define inside method arguments example.

```
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
```

Difference between general class and anonymous inner classes:

General Class	Anonymous Inner Class
1) A general class can extends only one class at a time.	1) Ofcource anonymous inner class also can extends only one class at a time.
2) A general class can implement any no. Of interfaces at a time.	2) But anonymous inner class can implement only one interface at a time.
3) A general class can extends a class and can implement an interface simultaneously.	3) But anonymous inner class can extends a class or can implements an interface but not both simultaneously.

Static nested classes:

- Sometimes we can declare inner classes with static modifier such type of inner classes are called static nested classes.
- In the case of normal or regular inner classes without existing outer class object there is no chance of existing inner class object.
- But in the case of static nested class without existing outer class object there may be a chance of existing static nested class object.

Example:

```
class Test
{
	static class Nested
	{
		public void methodOne()
		{
			System.out.println("nested class method");
		}
	}
	public static void main(String[] args)
	{
		Test.Nested t=new Test.Nested();
		t.methodOne();
	}
}
```

- Inside static nested classes we can declare static members including main() method also. Hence it is possible to invoke static nested class directly from the command prompt.

Example:

```
class Test
```

```
{
        static class Nested
        {
                public static void main(String[] args)
                {
                        System.out.println("nested class main method");
                }
        }
        public static void main(String[] args)
        {
                System.out.println("outer class main method");
        }
}
```

Output:

```
E:\SCJP>javac Test.java
E:\SCJP>java Test
Outer class main method
E:\SCJP>java Test$Nested
Nested class main method
```

- From the normal inner class we can access both static and non static members of outer class but from static nested class we can access only static members of outer class.

Example:

```
class Test
{
        int x=10;
        static int y=20;
        static class Nested
        {
                public void methodOne()
                {
                        System.out.println(x);//C.E:non-static variable x cannot be referenced
from a static context
                        System.out.println(y);
                }
        }
}
```

Compression between normal or regular class and static nested class?

Normal /regular inner class	Static nested class

1) Without existing outer class object there is no chance of existing inner class object. That is inner class object is always associated with outer class object.	1) Without existing outer class object there may be a chance of existing static nested class object. That is static nested class object is not associated with outer class object.
2) Inside normal or regular inner class we can’t declare static members.	2) Inside static nested class we can declare static members.
3) Inside normal inner class we can’t declare main() method and hence we can’t invoke regular inner class directly from the command prompt.	3) Inside static nested class we can declare main() method and hence we can invoke static nested class directly from the command prompt.
4) From the normal or regular inner class we can access both static and non static members of outer class directly.	4) From static nested class we can access only static members of outer class directly.