Lesson 6
Nested and Inner Class

Wholeness of the Lesson

Inner and nested classes allow classes to play the roles of instance variable, static variable and local variable, providing more expressive power to the Java language. Likewise, it is the hidden, unmanifest dynamics of consciousness that are responsible for the huge variety of expressions in the manifest world.

Nested Classes

The Java programming language allows you to define a class within another class.

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

Nested Class

- A nested class is a member of its enclosing class.
- Nested classes are divided into two categories: non-static and static. Non-static nested classes are called inner classes. Nested classes that are declared static are called static nested classes.

  Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private.

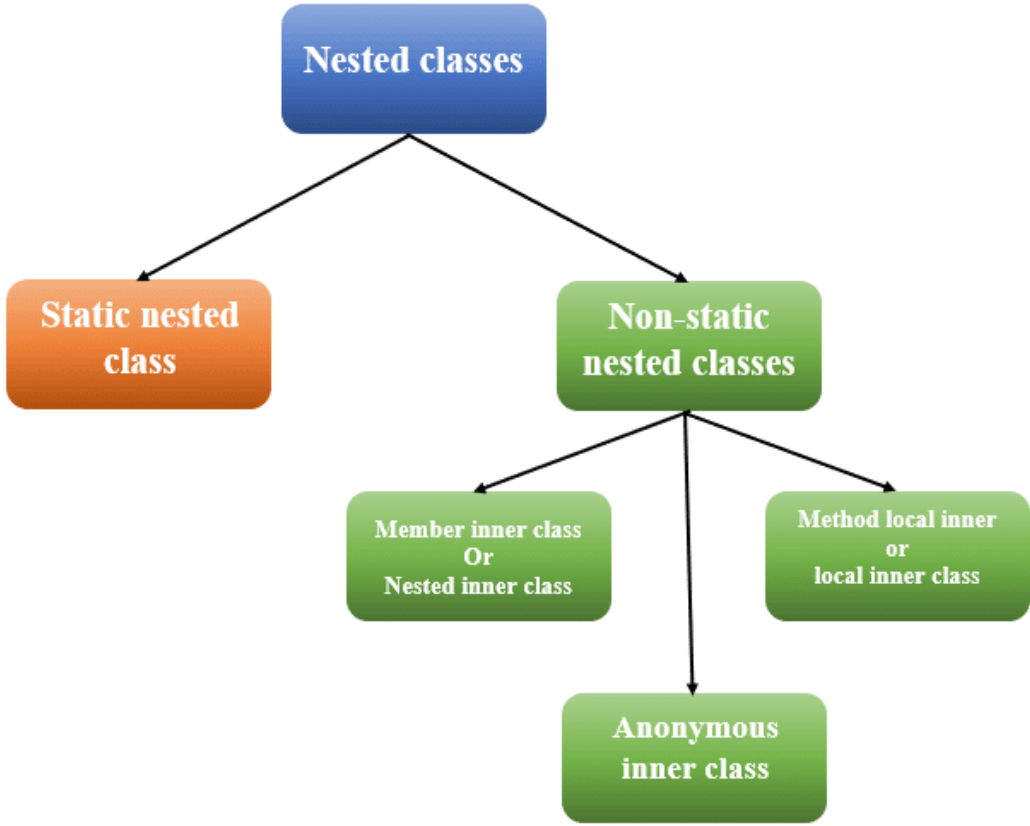
  Static nested classes do not have access to other members of the enclosing class.
- Advantages of nested class
- By use of nested classes, the inner class can access all variables and methods of outer class. It can access private variables or methods.
- It provides good maintains and readability of code.
- It requires less code that lead to code optimization.

Nested Class Types

```
class OuterClass{

     // Body of OuterClass
    static class StaticNestedClass{
        // Body of StaticNestedClass
    }

    class InnerClass{
        // Body of OuterClass
    }
}
```



Why use nested classes?

- It is a way of logically grouping classes that are only used in one place
- It increases encapsulation
- It can lead to more readable and maintainable code

Static nested Classes

- As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference. Inner Class and Nested Static Class Example demonstrates this.

- A static nested class interacts with the instance members of its outer class (and other classes) just like any other top-level class. In effect, a static nested class is behaviorally a top-level class that has been nested in another top-level class for packaging convenience.

```
You instantiate a static nested class the same way as a top-level class:
StaticNestedClass staticNestedObject = new StaticNestedClass();
```

How to access static nested class

```
class OuterClassName
{
   // variables and method of outer class
   static class className
   {
    // variables and method of static class
   }
}

OuterClassName.StaticNestedClassName;
```

```
class College {

// variables and method of College class
    static class Student {
    // variables and method of     Studen
class
    }
}

College.Student
```

How to use static nested class

- You can create an object of the static nested class by use of an outer class.

OuterClassName.StaticNestedClassName objectName = new outerClassName.StaticNestedClassName();

```
College.Student object = new College.Student();
```

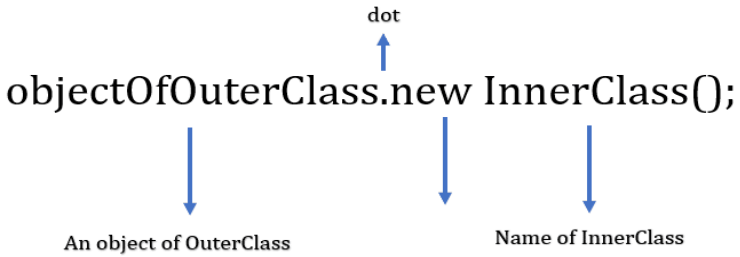
Static nested class rules

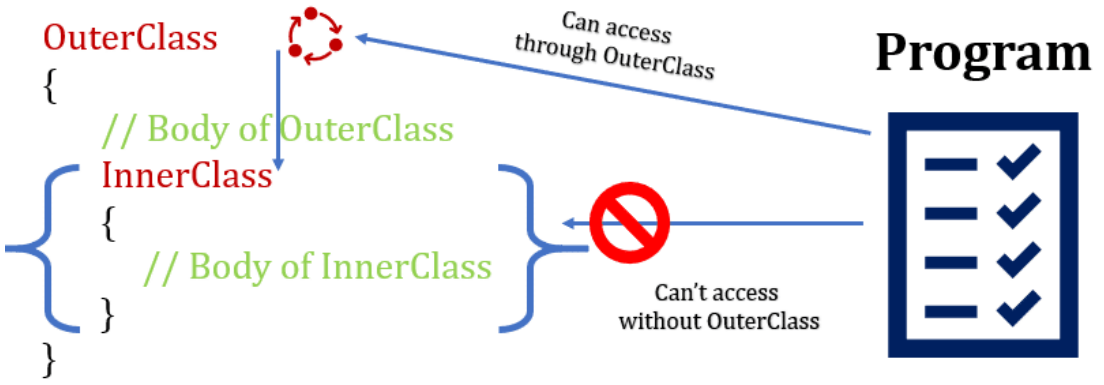
- In Java, static class can’t access non-static variables and methods.
- It can be accessed by an outer class name.
- It can access the static variables and static methods of outer class including private.

Inner Classes(Non-static nested class)

- As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.
- To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

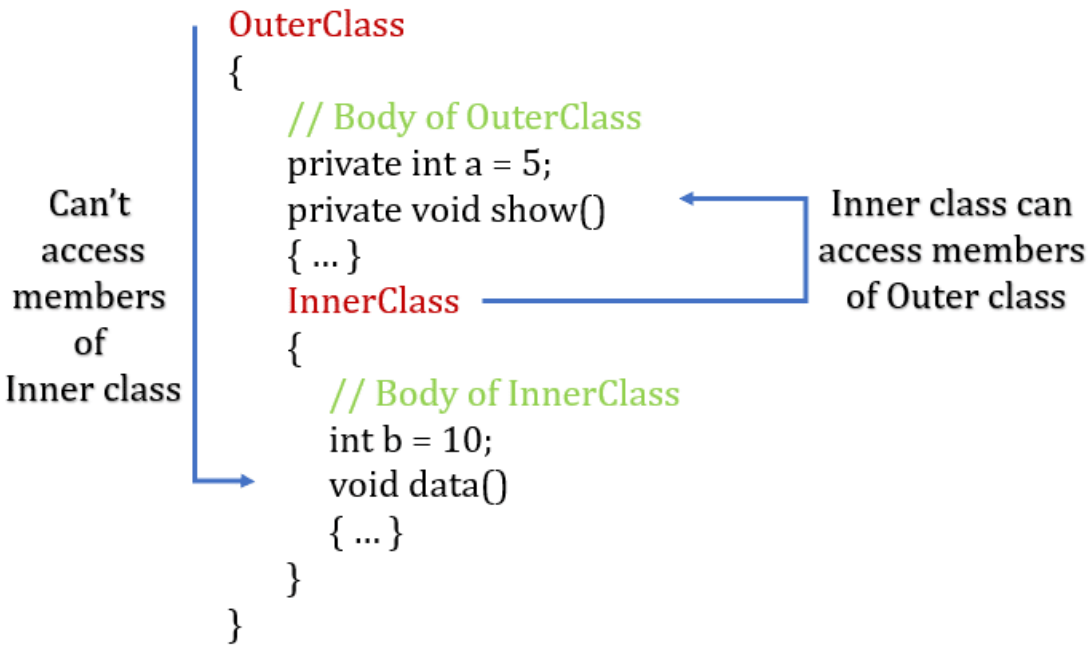
```
OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```





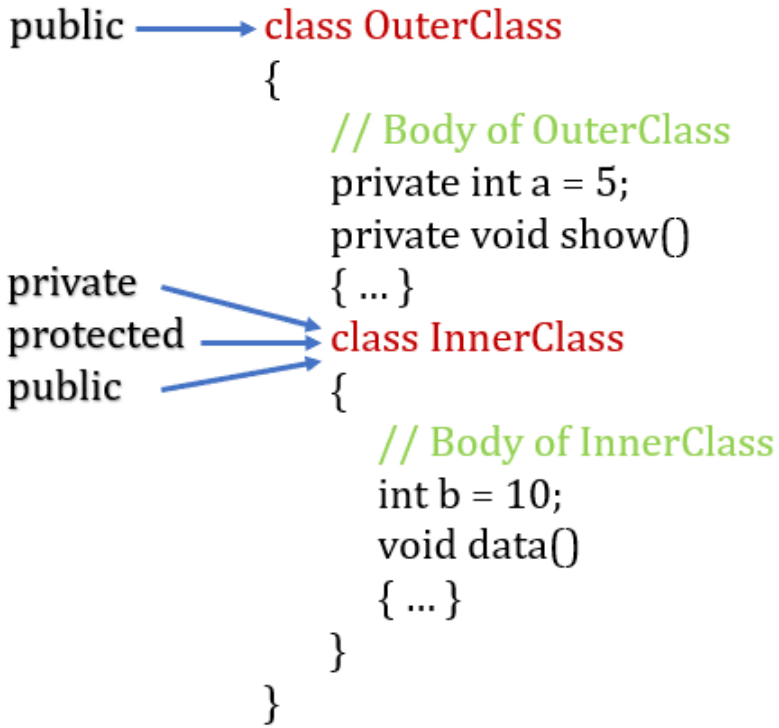
Inner Classes(Non-static nested class)

- An instance of InnerClass can exist only within an instance of OuterClass and has direct access to the methods and fields of its enclosing instance.



Inner Class modifier

- As a member of the OuterClass, a nested class can be declared private, public, protected, or package private. (Recall that outer classes can only be declared public or package level aka default.)



How to access members between Outer class and Inner Class

- We can access the member of the Inner class in Outer class by the object of Inner class. We can access the member of the Inner class in a static method as well as in the non-static method(Instance method).

Outer Class access Inner Class members

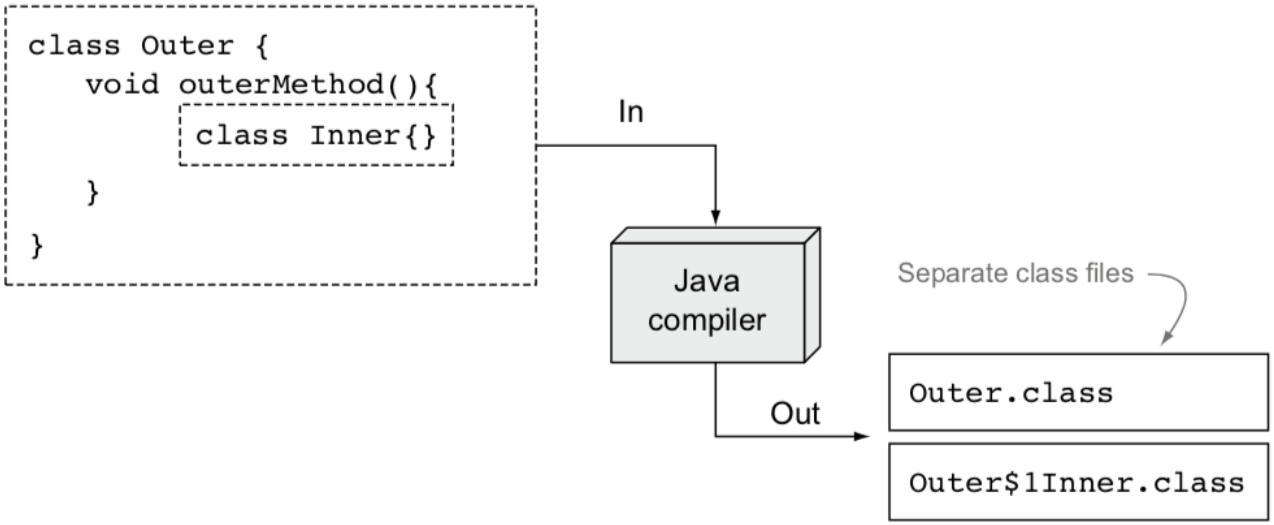
Inner Class access Outer Class instance members

Inner Class access Outer Class static

Demo code: D09 Inner Class.lesson6.innerclass

Internal working by the compiler

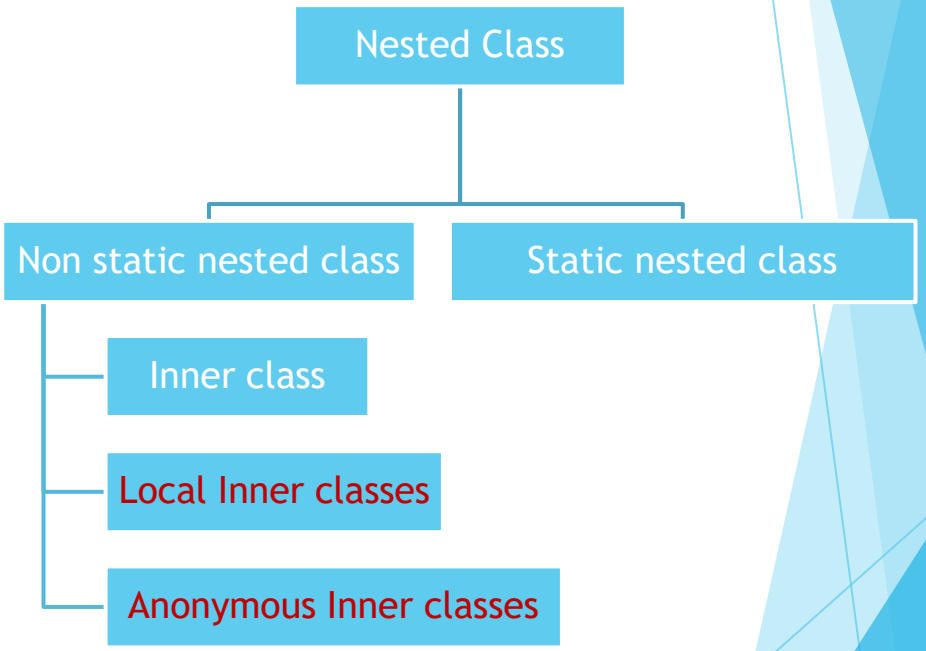
The compiler adds code to nested class definitions and to the enclosing class to support the features of nested classes. When you compile the UserIO class (which has two member inner classes), you will see the inner classes explicitly named in .class files, using a '$' in their names to distinguish them as nested classes.



UserIO$1.class	1 KB	CLASS File
UserIO$ClearListener.class	2 KB	CLASS File
UserIO$SubmitListener.class	1 KB	CLASS File
UserIO.class	5 KB	CLASS File
UserIO.java	6 KB	JAVA File

Special Inner Class Types

- There are two special kinds of inner classes
  - Local Inner class(method Local Inner class)
  - Anonymous Inner class



Local Inner Classes

- Local classes are classes that are defined in a *block*, which is a group of zero or more statements between balanced braces. You typically find local classes defined in the body of a method.

```
class OuterClass
{
    void methodName()
    {
        // method local inner class
        class InnerClass
        {
            .....
        }
    }
}
```

Local Inner class modifier

- A method local inner class can’t declare with the private, protected, and public access modifiers. It can be only a default access modifier and we already know if we are not defining any access modifier then it is considered as default. A method local inner class is not associated with any object. If we will try to use any modifier with the class name the compiler will show the compilation error. We can only use the abstract keyword or final keyword with it.

Demo code: D09 Inner Class.lesson6.localinnerclass

How to access members in local Inner class

- As we know a local inner class can be defined in the method. Like a member inner class, the method local inner class can access all the variables of the outer class. Even they are private members or not.

Demo code: Day09 Inner Class.lesson6.localinnerclass

Local Inner Classes rules

- A local inner class or method local inner class can't be declared as a member of the outer class. The method local inner class belongs to the block/method in which they are defined.
- The method local inner class can access the fields of the Outer class.
- We can't use any access modifiers with method local inner class. It means the method local inner class can't use private, public, or protected access modifiers with it.
- We can use the abstract keyword and final keyword with method local inner class.
- The method local inner class should be instantiated within a block/method where it is created. You can't instantiate it from outside the block/method.
- A method local inner class in Java can access only the final local variable of the enclosing block. But from JDK 8, it is possible to access the non-final local variable of enclosing block in the local inner class.
- Method local inner class can extend an abstract class or can also implement an interface.

Quiz - What is the Output?

```
public class ShadowTest {
    public int x = 0;
    class FirstLevel {
        public int x = 1;
        void methodInFirstLevel(int x) {
            System.out.println("x = " + x);
            System.out.println("this.x = " + this.x);
            System.out.println("ShadowTest.this.x = " + ShadowTest.this.x);
        }
    }

    public static void main(String[] args) {
        ShadowTest st = new ShadowTest();
        ShadowTest.FirstLevel fl = st.new FirstLevel();
        fl.methodInFirstLevel(23);
    }
}
```

Main Point

- Inner classes – a special kind of nested class -- have access to the private members of their enclosing class. The most commonly used kind of inner class is a *member* inner class. Likewise, when individual awareness is awake to its fully expanded, self-referral state, the memory of its eternal and infinitely dynamic nature becomes lively.

Anonymous Inner Class

- Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time. They are like local classes except that they do not have a name. Use them if you need to use a local class only once.

```
// Demo can be interface, abstract/concrete class
Demo obj = new Demo()
{
  // data members and methods
  public void methodOfDemo()
  {
    ........
  }
};
```

How to create an anonymous class

- There are two ways to create an anonymous class in java.

  Create an anonymous class by use of class
  Create an anonymous class by use of Interface

Create Anonymous Inner Class by use of Abstract Class

```
abstract class Student {
    abstract void record();
}

class AnonymousInnerExample {
    public static void main(String args[]) {
        Student obj = new Student() {
            void record() {
                System.out.println("This method is defined in anonymous class");
            }
        };
        obj.record();
    }
}
```

Create Anonymous Inner Class by use of Concrete Class

```
class Student {
    void record() {
    }
}

class AnonymousInnerExample {
    public static void main(String args[]) {
        Student obj = new Student() {
            void record() {
                System.out.println("This method is defined in concrete class");
            }
        };
    obj.record();
    }
}
```

Create Anonymous Inner Class by use of Interface

```
interface Student {
    void record();
}

class AnonymousInnerExample {
    public static void main(String args[]) {
        Student obj = new Student() {
        public void record() {
            System.out.println("This method is defined in anonymous class");
        }
    };
    obj.record();
    }
}
```

Anonymous Inner Class rules

- Anonymous class has no name
- It can be instantiated only once
- It is usually declared inside a method or a code block, curly braces ending with a semicolon.
- It is accessible only at the point where it is defined.
- It does not have a constructor simply because it does not have a name
- It cannot be static

Main point

- Classes are the fundamental notion in Java – programs are built from classes. With nested classes, Java makes it possible for this fundamental construct to play the roles of instance variable (member inner classes), static variable (static nested classes), and local variable (local inner classes). Likewise, in the unfoldment of creation, pure intelligence assumes the role of creative intelligence – in all of creation we find pure intelligence.

this and constructors

- Like ordinary classes, when a member inner class is instantiated, it has an implicit parameter this.
- The “this” of the enclosing class is accessible from within itself (as ever), and also from within the inner class.
- The “this” of the inner class is accessible from within itself, but *not* from the enclosing class.

this and constructors

```
class OuterClass6 {

    InnerClass inner;
    private String param;

    OuterClass6(String param) {
        inner = new InnerClass("innerStr");
        this.param = param;
    }

    void outerMethod() {
        System.out.println("OuterClass6.outerMethod OuterClass this " + this.param);
        System.out.println("OuterClass6.outerMethod InnerClass param " + inner.param);
        inner.innerMethod();
    }

    private class InnerClass {
        private String param;

        InnerClass(String param) {
            this.param = param;
        }

        void innerMethod() {
            System.out.println("InnerClass.innerMethod OuterClass this " + OuterClass6.this.param);
            System.out.println("InnerClass.innerMethod InnerClass this " + this.param);
        }
    }

    public static void main(String[] args) {
        (new OuterClass6("outerStr")).outerMethod();
    }
}
```

Demo: Day09-Inner Classes/src/lesson6/innerclass/OuterClass6.java

Implementing an ActionListener with an Anonymous Inner Class

```
public class AnonymousInnerClass extends JFrame {

    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

    public AnonymousInnerClass() {

        JButton button = new JButton("Print");
        JPanel panel = new JPanel();
        panel.add(button);
        add(panel);

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Process Print");
            }
        });
    }

    public static void main(String[] args) {
        AnonymousInnerClass frame = new AnonymousInnerClass();

        frame.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        frame.setLocationByPlatform(true);
        frame.setTitle("AnonymousListenerDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

    }
}
```

Lambda Expressions

1. An interface that contains just one (abstract) method is called a *functional interface*.
2. An implementation of a functional interface is called a *functor*
3. A *closure* is a functor that remembers the state of its enclosing environment

Example: ActionListener

- ActionListener is an interface with just one method – actionPerformed, so it is a ***functional interface***
- An anonymous inner class **implementation of ActionListener** is a **functor**
- An anonymous inner class **implementation of actionPerformed** is a **closure**
- A lambda expression can be used in place of an anonymous inner class

Example

```
// Without Lambda, but Anonymous Inner Class
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Process Print”);
    }
});


// Using Lambda
button.addActionListener(
evt -> {
        System.out.println("Process Print”);
});
```

Application: Implementing Singleton Using Nested Class

There are several ways to implement the singleton pattern:

- Use lazy initialization to instantiate a private static instance variable. (Not thread safe.) (Concept we learned in Lesson-3) – (SingleThreadedSingleton.java)
- Store instance as a public static constant, constructed when class is loaded. (SingletonAsPublicStatic.java)
- Implement as an Enum. Also constructed when enum is loaded. (SingletonAsEnum.java)
- Use Spring's Singleton Holder pattern, whereby the singleton is stored as a static variable of a static nested class. Permits lazy initialization and is thread safe. (SingletonAsInnerClass.java)

UNITY CHART

CONNECTING THE PARTS OF KNOWLEDGE

WITH THE WHOLENESS OF KNOWLEDGE

Inner class is an integral part of the unlimited potential

A *nested class* is a class that is defined inside another class.

An *inner class* has full access to its context, its enclosing class.

Transcendental Consciousness: TC is the unbounded context for individual awareness.

***Impulses within the Transcendental field**: The impulses within the transcendental field are the simplest (and most powerful) ones to achieve the desired result.*

***Wholeness moving within Itself**: When individual awareness is permanently and fully established in its transcendental "context" – pure consciousness – every impulse of creation is seen to be an impulse of one's own awareness.*

PRACTICE