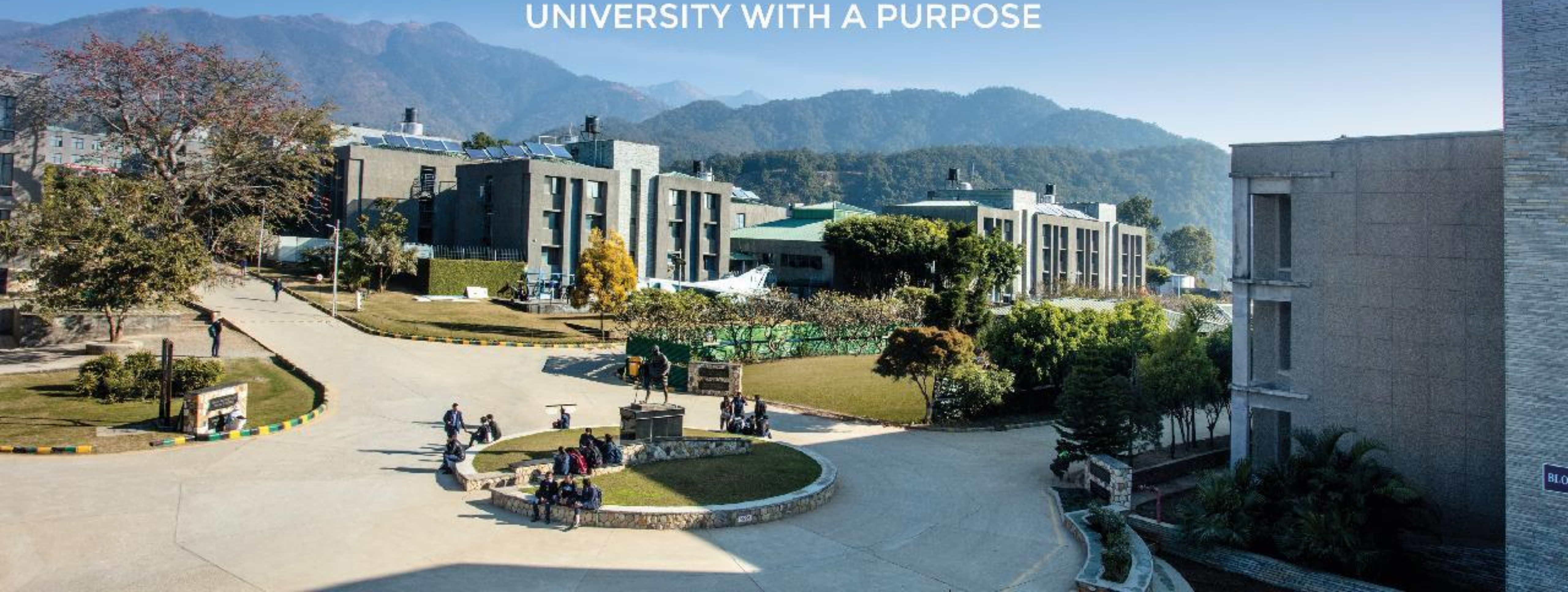
UPES

UNIVERSITY WITH A PURPOSE



UPES
UNIVERSITY WITH A PURPOSE

Nested Classes in Java

Nested Class in Java

- The Java programming language allows you to define a class within another class. Such a class is called a nested class and is illustrated here:

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

- They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and creates more readable and maintainable code.
- The scope of a nested class is bounded by the scope of its enclosing class. Thus, class *`NestedClass`* does not exist independently of class *`OuterClass`*.
- A nested class has access to the members, including private members, of the class in which it is nested. However, the reverse is not true i.e., the enclosing class does not have access to the members of the nested class.
- A nested class is also a member of its enclosing class.
- As a member of its enclosing class, a nested class can be declared *private*, *public*, *protected*, or *package private*(default).
- Nested classes are divided into two categories:
    - **static nested class :** Nested classes that are declared *static* are called static nested classes.
    - **inner class :** An inner class is a non-static nested class.

Why Use Nested Classes

Compelling reasons for using nested classes include the following:

- **It is a way of logically grouping classes that are only used in one place:** If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.

- **It increases encapsulation:** Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

- **It can lead to more readable and maintainable code:** Nesting small classes within top-level classes places the code closer to where it is used.

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.
- 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.
- Static nested classes are accessed using the enclosing class name:

```
OuterClass.StaticNestedClass
```

- For example, to create an object for the static nested class, use this syntax:

```
OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();
```

Static Nested Class Example

```
// outer class
class OuterClass {
          // static member
          static int outer_x = 10;
          // instance(non-static) member
          int outer_y = 20;
          // private member
          private static int outer_private = 30;
          // static nested class
          static class StaticNestedClass {
                    void display()      {
                              // can access static member of outer class
                              System.out.println("outer_x = " + outer_x);
                              // can access display private static member of outer class
                              System.out.println("outer_private = " + outer_private);
                              // The following statement will give compilation error
                              // as static nested class cannot directly access non-static membera
                              // System.out.println("outer_y = " + outer_y);    }     }    }

// Driver class
public class StaticNestedClassDemo {
          public static void main(String[] args)  {
                    // accessing a static nested class
                    OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();
                    nestedObject.display();

          }
}
```

Inner Classes

- 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.

- Objects that are instances of an inner class exist within an instance of the outer class. Consider the following classes:

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

- 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.

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

- There are two special kinds of inner classes: **local classes and anonymous classes.**

Local Inner Class in Java

- Local Inner Classes are the inner classes that are defined inside a block. Generally, this block is a method body. Sometimes this block can be a for loop, or an if clause.
- Local Inner classes are not a member of any enclosing classes. They belong to the block they are defined within, due of which local inner classes cannot have any access modifiers associated with them. However, they can be marked as final or abstract.
- These class have access to the fields of the class enclosing it. Local inner class must be instantiated in the block they are defined in.

Rules of Local Inner Class:

1. The scope of local inner class is restricted to the block they are defined in.
2. Local inner class cannot be instantiated from outside the block where it is created in.
3. Till JDK 7,Local inner class can access only final local variable of the enclosing block. However From JDK 8, it is possible to access the non-final local variable of enclosing block in local inner class.
4. A local class has access to the members of its enclosing class.
5. Local inner classes can extend an abstract class or can also implement an interface.

Local Inner Class in Java

Declaring a Local Inner class: A local inner class can be declared within a block. This block can be either a method body, for loop or even an if statement.

Accessing Members: A local inner class has access to fields of the class enclosing it as well as the fields of the block that it is defined within. These classes, however, can access the variables or parameters of the block that encloses it only if they are declared as final or are effectively final. A variable whose value is not changed once initialized is called as effectively final variable. A local inner class defined inside a method body, has access to it's parameters.

What happens at compile time?

When a program containing a local inner class is compiled, the compiler generates two .class files, one for the outer class and the other for the inner class that has the reference to the outer class. The two files are named by compiler as:

- Outer.class
- Outer$1Inner.class

Declaration within a method body

```
public class Outer {
          private void getValue() {
                    int sum = 20;
                    // Local inner Class inside method
                    class Inner {
                              public int divisor;
                              public int remainder;

                              public Inner() {
                                        divisor = 4;
                                        remainder = sum%divisor; }
                              private int getDivisor()     {
                                        return divisor;     }
                              private int getRemainder()    {
                                        return sum%divisor; }
                              private int getQuotient()     {
                                        System.out.println("Inside inner class");
                                        return sum / divisor;} }

                    Inner inner = new Inner();
                    System.out.println("Divisor = "+ inner.getDivisor());
                    System.out.println("Remainder = " + inner.getRemainder());
                    System.out.println("Quotient = " + inner.getQuotient());    }

          public static void main(String[] args)  {
                    Outer outer = new Outer();
                    outer.getValue();   }  }
```

Output:

```
Divisor = 4
Remainder = 0
Inside inner class Quotient = 5
```

Declaration within a method body

Note :- A local class can access local variables and parameters of the enclosing block that are *effectively final*. For example, if you add the following assignment statement in the *Inner* class constructor or in any method of *Inner* class in above example :

```
public Inner()
{
       sum = 50;
       divisor = 4;
       remainder = sum%divisor;
}
```

Because of this assignment statement, the variable sum is not effectively final anymore. As a result, the Java compiler generates an error message similar to “local variables referenced from an inner class must be final or effectively final”.

Declaration inside an if statement

```
public class Outer {
        public int data = 10;
        public int getData() {
                return data; }
        public static void main(String[] args)     {
                Outer outer = new Outer();

                if(outer.getData() < 20)  {
                        // Local inner class inside if clause
                        class Inner  {
                                public int getValue()
                                {
                                        System.out.println("Inside Inner class");
                                        return outer.data;
                                }
                        }

                        Inner inner = new Inner();
                        System.out.println(inner.getValue());
                }
                else
                {
                        System.out.println("Inside Outer class");
                }
        }
}
```

References

1. Accessed from https://www.geeksforgeeks.org/local-inner-class-java/
2. Accessed from https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

THANK YOU

UPES

UNIVERSITY WITH A PURPOSE