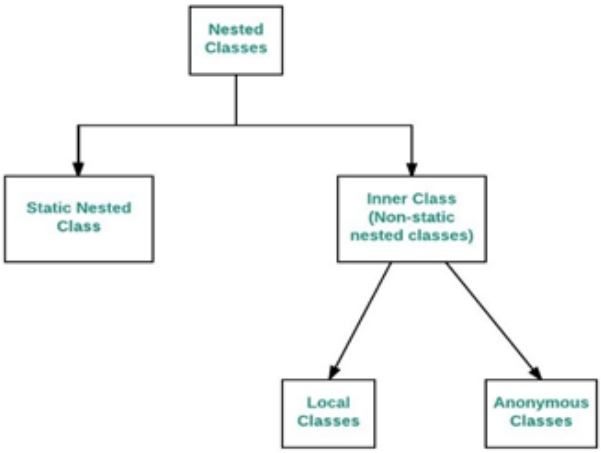
Nested Classes in Java

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

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



- Nested classes are divided into two categories: static and non-static.
- Nested classes that are declared static are called ***static nested classes*.**
- Non-static nested classes are called ***inner classes*.**
- A nested class is a **member** of its enclosing class.
- As a member of the **OuterClass**, a nested class can be declared **private, public, protected, or package level**. (Recall that outer classes can only be declared public or package private.)

```
class OuterClass {
    ...
    static class
StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Why Use Nested Classes?

- **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 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 are accessed using the enclosing class name.
- For example, to create an object for the static nested class, use this syntax:

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

- As like instance methods and variables, an inner class is associated with an instance of its enclosing 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.
- 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
  - Anonymous classes.
- **Local classes** are classes that are defined in a block, which is a group of zero or more statements between balanced braces.
- Typically local classes defined in the body of a method.
- Scope of local classes is up to method, where it is declared.

Anonymous Classes

- Anonymous classes enable us to make our 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.
- In other words local classes are class declaration; anonymous classes are an expression, which means that you define the class in another expression.
- In the below example program, We can use **Anonymous class** instead of creating implementation class for Runnable interface

```
public class Anonymous  {
	public static void main(String[] args) {
		Runnable r = new Runnable() {
			@Override
			public void run() {
				for (int i = 0; i < 5; i++) {
					try {
						Thread.sleep(2000);
						System.out.println("Running r  "+(i+1));
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
				}
			}
		};
		Thread t = new Thread(r);
		t.start();
	}
}
```