Lesson-8- Class Notes

```java
@FunctionalInterface
public interface TriFunction<T,U,V,R> {
  R apply(T t, U u, V v);
  public abstract String toString();
  int hashCode();

}
```

The inclusion of Object methods in functional interfaces in Java doesn't count toward the interface's abstract method count because these methods are inherently part of every Java object through the Object class.

They are included to allow their explicit overriding in implementing classes, potentially to ensure their behavior aligns more closely with the interface.

However, these inclusions do not alter the nature of the interface as functional if there is precisely one abstract method defined inside the interface.

Final or effectively final

```java
package lesson8.lecture.inclassdemo;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

public class EffectivelyFinal {

	private int x = 10; // Instance Variable

	public void test(int a, List<String> list) {
		int z = 50; // Local Variable
		class local {
			private int y = 20; // Instance field
			void localM() {
				System.out.println(x);
				System.out.println(y);
				System.out.println(x + y);
				// a = a + y; // a is final or effectively final
				list.add("Hello");
				// list should be final or effectively final
			// list = new LinkedList<>(); // After initialization you cannot reassign
```

```
                    }

                }
                new local().localM();
        }

        public static void main(String args[]) {
                EffectivelyFinal ob = new EffectivelyFinal();
                ob.test(20, new ArrayList<String>(Arrays.asList("Welcome")));
        }

}
```

Generate Random Number using Lambda and Functional Interface

```
Unsupplied sr1 = () -> new Random().nextInt(); // RandomGenerator.getDefault().nextInt()
System.out.println(r1.getAsInt());
```