Class-Level Synchronization (Static Synchronized Methods)

Concept

- When a method is declared **static synchronized**, the **lock is taken on the class object**, not on any object instance.
- This means:
  ✓ Only **one thread** can execute that static synchronized method **for the entire class**,
  ✓ Even if multiple objects of that class are created.

Why?

Because **static methods belong to the class**, so Java uses the **Class-level lock** (also called the *Class monitor lock*).

Program demonstrating class-level synchronization

```
class Printer {

    static synchronized void printNumbers(String threadName) {
        for (int i = 1; i <= 5; i++) {
            System.out.println(threadName + " - " + i);
        }
    }
}

class MyThread extends Thread {
    public void run() {
        Printer.printNumbers(Thread.currentThread().getName());
    }
}

public class Main {
    public static void main(String[] args) {
```

```
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();

        t1.setName("Thread A");
        t2.setName("Thread B");

        t1.start();
        t2.start();
    }
}
```

What Happens?

- Both threads call the **same static synchronized** method.
- Only **one thread at a time** enters `printNumbers()` .
- Example output (order will be consistent per thread):

```
Thread A - 1
Thread A - 2
Thread A - 3
Thread A - 4
Thread A - 5
Thread B - 1
Thread B - 2
Thread B - 3
Thread B - 4
Thread B - 5
```

No intermixing

You will **never** see:

```
Thread A - 1
Thread B - 1
Thread A - 2
```

because class-level lock allows only **one thread** to execute the static synchronized method at a time.
