Synchronization is to control the access of multiple threads to any shared resource
Synchronization is used when we want to allow only one thread to access the shared resource
It follows sequential execution of the threads

Synchronized Method
More execution time
Performance is low

Synchronized Block
Less execution time
Performance is high

```
class A {
public synchronized void m1() {

for (int i = 0; i < 5; i++) {
System.out.println(i + " : " + Thread.currentThread().getName());

try {
Thread.sleep(2000);
} catch (InterruptedException e) {
e.printStackTrace();
}

}
}
}
```

```
class B extends Thread {

A a;

public B(A a) {
this.a = a;
}

public void run() {
a.m1();
}
}
```

```
class C extends Thread {

A a;

public C(A a) {
this.a = a;
}

public void run() {
a.m1();
}
}
```

```
class D extends Thread {

A a;

public D(A a) {
this.a = a;
}

public void run() {
a.m1();
}
}
```

```
public class Client extends Thread {

A a;

public Client(A a) {
this.a = a;
}

public void run() {
a.m1();
}

public static void main(String[] args) {

A a = new A();
B b = new B(a);
C c = new C(a);
D d = new D(a);

b.start();
c.start();
d.start();

}
}
```

```
0 : Thread-0
1 : Thread-0
2 : Thread-0
3 : Thread-0
4 : Thread-0
0 : Thread-2
1 : Thread-2
2 : Thread-2
3 : Thread-2
4 : Thread-2
0 : Thread-1
1 : Thread-1
2 : Thread-1
3 : Thread-1
4 : Thread-1
```

```
class A {
public void m1() {

for (int i = 0; i < 5; i++) {
System.out.println(i + " : " + Thread.currentThread().getName());

try {
Thread.sleep(2000);
} catch (InterruptedException e) {
e.printStackTrace();
}

}
}
}
```

```
class B extends Thread {

A a;

public B(A a) {
this.a = a;
}

public void run() {
a.m1();
}
}
```

```
class C extends Thread {

A a;

public C(A a) {
this.a = a;
}

public void run() {
a.m1();
}
}
```

```
class D extends Thread {

A a;

public D(A a) {
this.a = a;
}

public void run() {
a.m1();
}
}
```

```
public class Client extends Thread {

A a;

public Client(A a) {
this.a = a;
}

public void run() {
a.m1();
}

public static void main(String[] args) {

A a = new A();
B b = new B(a);
C c = new C(a);
D d = new D(a);

b.start();
c.start();
d.start();

}
}
```

```
0 : Thread-1
0 : Thread-2
0 : Thread-0
1 : Thread-2
1 : Thread-1
1 : Thread-0
2 : Thread-0
2 : Thread-1
2 : Thread-2
3 : Thread-0
3 : Thread-1
3 : Thread-2
4 : Thread-1
4 : Thread-2
4 : Thread-0
```

```
class ThreadOne extends Thread {

public void m1() {

for (int i = 1; i <=5; i++) {
System.out.println(i + " Non Synchronized Area");
}

synchronized (this) {
for (int i = 6; i <= 10; i++) {
System.out.println(i + ": Synchronized Only some part of code");
}
}

System.out.println("Non Synchronized Area");
}
}
```

```
1 Non Synchronized Area
2 Non Synchronized Area
3 Non Synchronized Area
4 Non Synchronized Area
5 Non Synchronized Area
6: Synchronized Only some part of code
7: Synchronized Only some part of code
8: Synchronized Only some part of code
9: Synchronized Only some part of code
10: Synchronized Only some part of code
Non Synchronized Area
```

```
public class Test extends Thread {

ThreadOne one;

public Test(ThreadOne one) {
this.one = one;
}

@Override
public void run() {
one.m1();
}

public static void main(String[] args) {

ThreadOne t1 = new ThreadOne();
Test test1 = new Test(t1);
test1.start();

}
}
```