Q1:

A.What is the content of the matrix Need?

	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

B. Is the system in a safe state?

Yes, this is a safe state, the order of the process running is:P0 -> P2 -> P1 -> P3 -> P4
The available recourse matrix changes: (1,5 ,2 ,0) -> After P0 (1,5,3,2) - > P2 (2, 8, 8, 6) -> P1 (3, 8, 8, 6) -> P3 (3, 14, 11, 8) -> P4 (3, 14, 12, 12)
So there were always available resources for at least one process to execute at a time.

Q2:

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
app.use(bodyParser.json());

app.get('', (req, res) => {
const name = req.query.name;
const response = `Welcome ${name}`;
const greeting = `Hello ${name}`;
//res.json({ greeting, response });
if(name) res.send(greeting + ", " + response);
else res.send("Hello World!");
});
```

```
const PORT = 3000;
app.listen(PORT, () => {
console.log(`Server running on http://localhost:${PORT}`);
});
```

Q3:

1. Image processing: When processing lots of pictures, such as cutting, shrinking or adding filters to 1000 images, multi-threading can always save more times because they can handle more pictures at the same time but single-threading application can only do one picture at a time.

2. Game development: When developing a 3D game, multi-threading game can render the light, sounds and other physics effect at same time, but single-threading application will do them one by one. Let’s imagine when a player wants to move, the image starts rendering and background music stops, and this is not acceptable, so multi-threading performs natural and smooth in a 3D game.

Q4:

1. Sequential Tasks: Tasks like basic number calculations or simple data operations may not benefit from multi-threading because the time of threads managing may cost more time than single-threading processing time.

2. Resource-Intensive Tasks: For tasks require lots of resources, such as CPU cores or memory, multi-threading may not provide better performance compared to a single-threaded solution. It can take more time due to context switching, I/O manipulation, and etc. In this case, single-threading application might perform better.

Q5:

Short-term scheduling is responsible for determining which process/thread should be executed on the CPU, medium-term scheduling manages the movement of processes/threads between main memory and secondary storage, while the long-term scheduling controls the admission of processes/threads into the system.

Together, these scheduling levels play a vital role in managing the execution of processes/threads in an operating system and optimizing overall system performance.

Q6:

When using FCFS algorithms:
The order of Process executed is : P1- > P2 -> P3

Process	Arrive Time	Burst Time	Completion Time	Turnaround Time
P1	0.0	8	8	8
P2	0.4	4	12	11.6
P3	1	1	13	12

Average Turnaround time = (8 + 11.6 + 12) / 3 = 10.5333333...

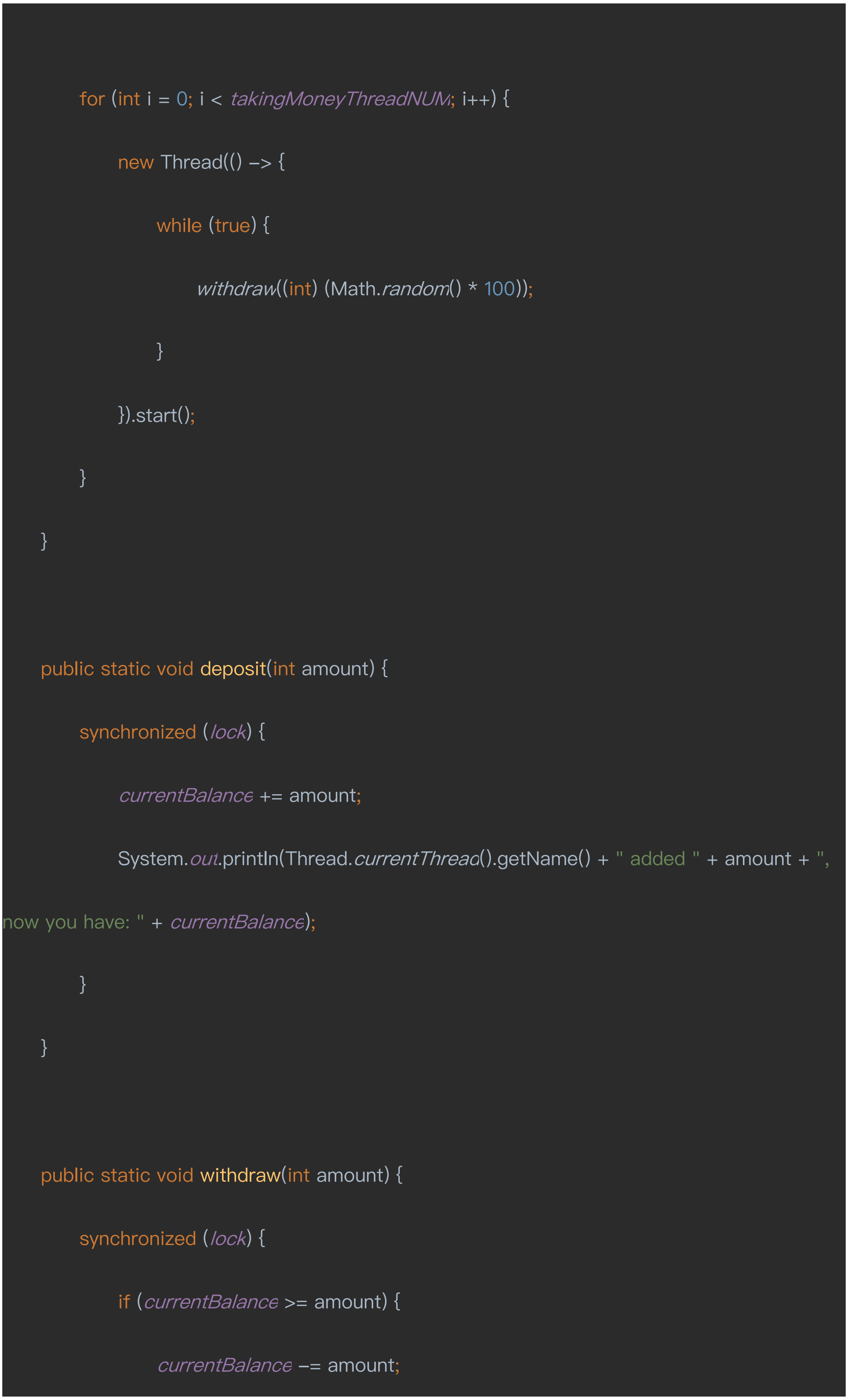
Q7:

```java
public class Q7 {

    static int addingMoneyThreadNUM = 4;
    static int takingMoneyThreadNUM = 5;
    static int currentBalance = 500;
    static Object lock = new Object();

    public static void main(String[] args) {

        for (int i = 0; i < addingMoneyThreadNUM; i++) {
            new Thread(() -> {
                while (true) {
                    deposit((int) (Math.random() * 100));
                }
            }).start();
        }
```



```
        for (int i = 0; i < takingMoneyThreadNUM; i++) {
            new Thread(() -> {
                while (true) {
                    withdraw((int) (Math.random() * 100));
                }
            }).start();
        }
    }

    public static void deposit(int amount) {
        synchronized (lock) {
            currentBalance += amount;
            System.out.println(Thread.currentThread().getName() + " added " + amount + ", now you have: " + currentBalance);
        }
    }

    public static void withdraw(int amount) {
        synchronized (lock) {
            if (currentBalance >= amount) {
                currentBalance -= amount;
                System.out.println(Thread.currentThread().getName() + " took " + amount + ", now you have: " + currentBalance);
            } else {
                System.out.println(Thread.currentThread().getName() + " failed taking " + amount + " from your account");
            }
        }
    }
}
```