Final Exam

Q1 : Consider the following snapshot of a system:

	Allocation	**Max**	**Available**
	ABCD	**ABCD**	**ABCD**
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from thread P1 arrives for (0,4,2,0), can the request be

granted immediately?

A1(a):

Need =

0	0	1	2
7	5	1	0
0	9	2	2
0	2	2	0
0	0	42	42

A1(b):

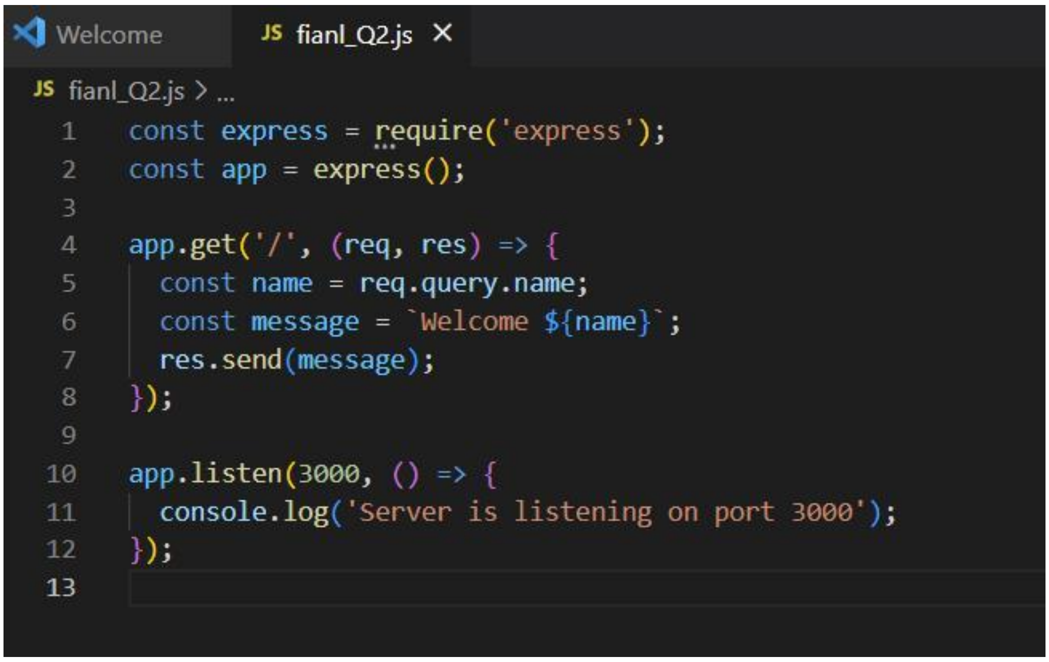
The resulting finished vector is [True, False, True, True, True], so all process can be finished, this system is in safe state

A1(c):

The resulting Finish vector is [True, True, True, True, True], which means that all processes can finish and the system is in a safe state. Therefore, the request from P1 for (0, 4, 2, 0) can be granted without putting the system into an unsafe state.

Q2. Create a nodejs code which implements a get method where you send "Hello {Name}" as the body and the response body is "Welcome {Name}"

A2:



```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  const name = req.query.name;
  const message = `Welcome ${name}`;
  res.send(message);
});

app.listen(3000, () => {
  console.log('Server is listening on port 3000');
});

```

Q3: Provide two programming examples in which multithreading provides better performance than a single-threaded solution.

A3: (1). Data processing: If a processing needs to search or update in a large database, it will waste a lot of time for single thread process. Meanwhile, multithread process can finish it very fast.

(2). Web server: When web server gets many requests at same time, a request have to wait previous request is done, which cause web serve will give response very slowly. And multithread can avoid this problem.

Q4. Provide two examples where multithreading does not provide better performance than single threaded.

A4: (1). Simple code: If a process is simple and workload for this process is small, using single thread is better because multithread will extend total running time and waste resource for extra thread.

(2). Limited resource process: If a process only have limited resource, using single thread is better, cause multithread will cost extra resources.

Q5: Describe the differences among short-term, medium-term, and long- term scheduling.

A5: Short-term scheduling is responsible for selecting which process should be executed next by the CPU. Medium-term scheduling is responsible for deciding which processes should be swapped in and out of memory. Long-term scheduling is responsible for selecting which processes should be admitted into the system.

Q6: What is the average turnaround time for these processes with the FCFS scheduling algorithm?

A6: Turnaround time for P1 = 8.0 - 0.0 = 8.0

Turnaround time for P2 = 12.4 - 0.4 = 12.0

Turnaround time for P3 = 13.4 - 1.0 = 12.4

Average turnaround time = (8.0 + 12.0 + 12.4) / 3 = 10.47 units of time

Q7: Write code for following system:

Person has a bank account

There are 4 threads adding money in the bank account

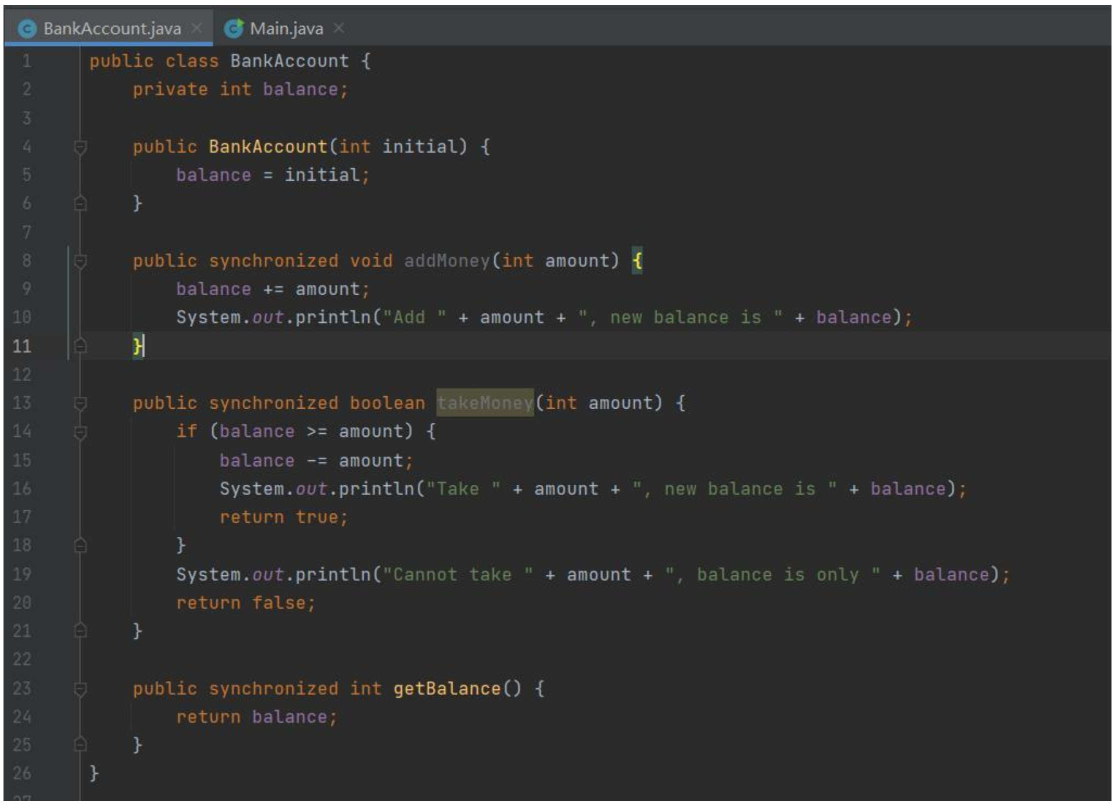
There are 5 threads taking money out of the bank account

Make sure that threads cannot take out money which is not there i.e. synchronisation of the bank account

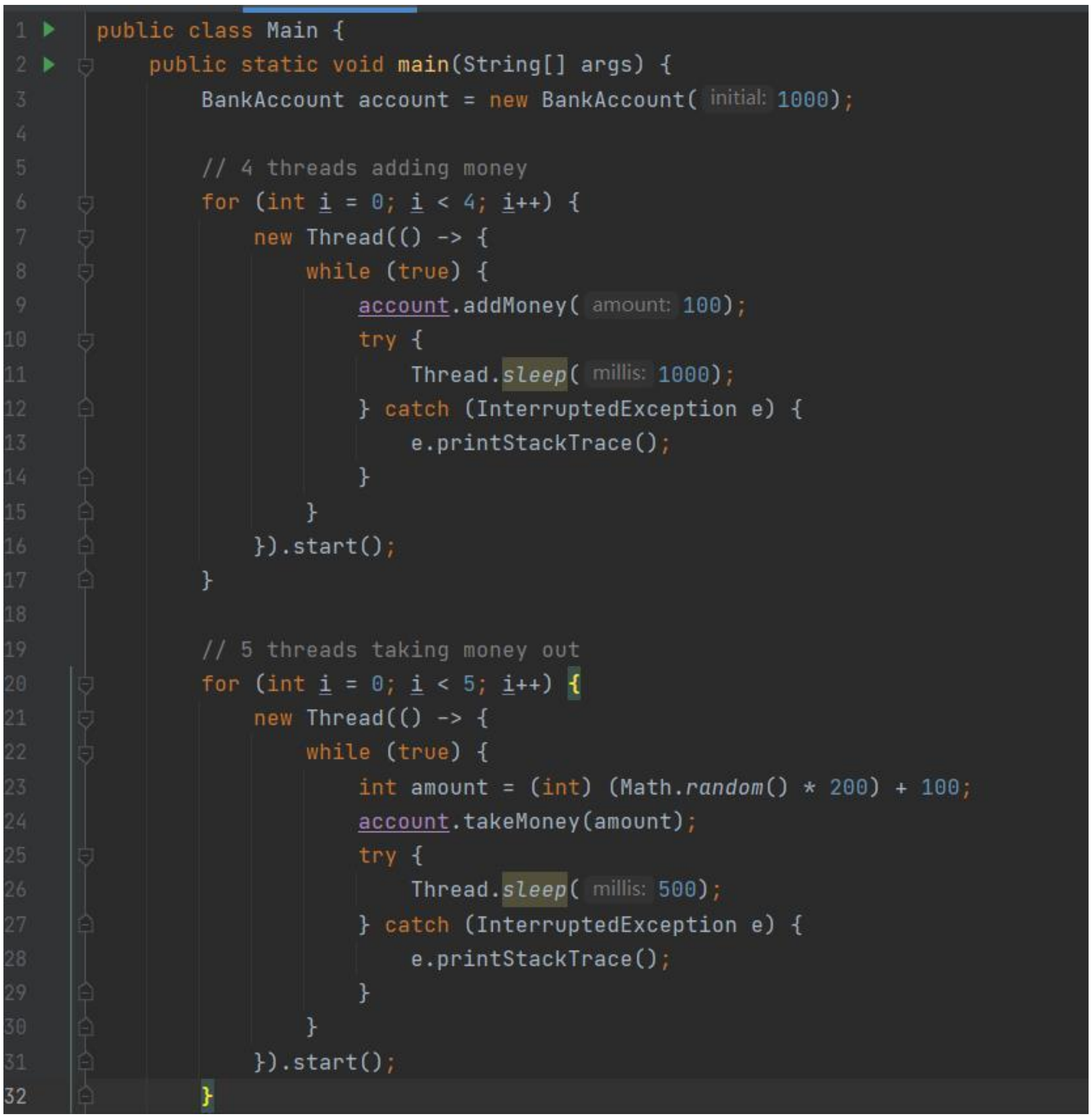
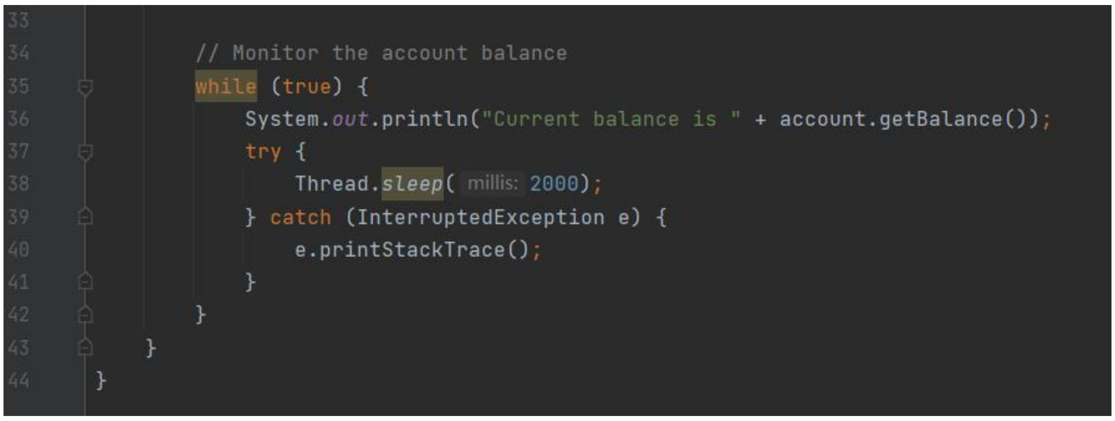
Make sure that the account tally is always correct.

This system will run forever.

A7:



```java
public class BankAccount {
    private int balance;

    public BankAccount(int initial) {
        balance = initial;
    }

    public synchronized void addMoney(int amount) {
        balance += amount;
        System.out.println("Add " + amount + ", new balance is " + balance);
    }

    public synchronized boolean takeMoney(int amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Take " + amount + ", new balance is " + balance);
            return true;
        }
        System.out.println("Cannot take " + amount + ", balance is only " + balance);
        return false;
    }

    public synchronized int getBalance() {
        return balance;
    }
}
```


```java
public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount( initial: 1000);

        // 4 threads adding money
        for (int i = 0; i < 4; i++) {
            new Thread(() -> {
                while (true) {
                    account.addMoney( amount: 100);
                    try {
                        Thread.sleep( millis: 1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }).start();
        }

        // 5 threads taking money out
        for (int i = 0; i < 5; i++) {
            new Thread(() -> {
                while (true) {
                    int amount = (int) (Math.random() * 200) + 100;
                    account.takeMoney(amount);
                    try {
                        Thread.sleep( millis: 500);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }).start();
        }
```

```java

        // Monitor the account balance
        while (true) {
            System.out.println("Current balance is " + account.getBalance());
            try {
                Thread.sleep( millis: 2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```