Problem Statement 1: Multithreading

1. Write a Java program which accepts multiple employees’ details and perform the following:

a. Create thread class.

b. Execute them using frokjoinpool.

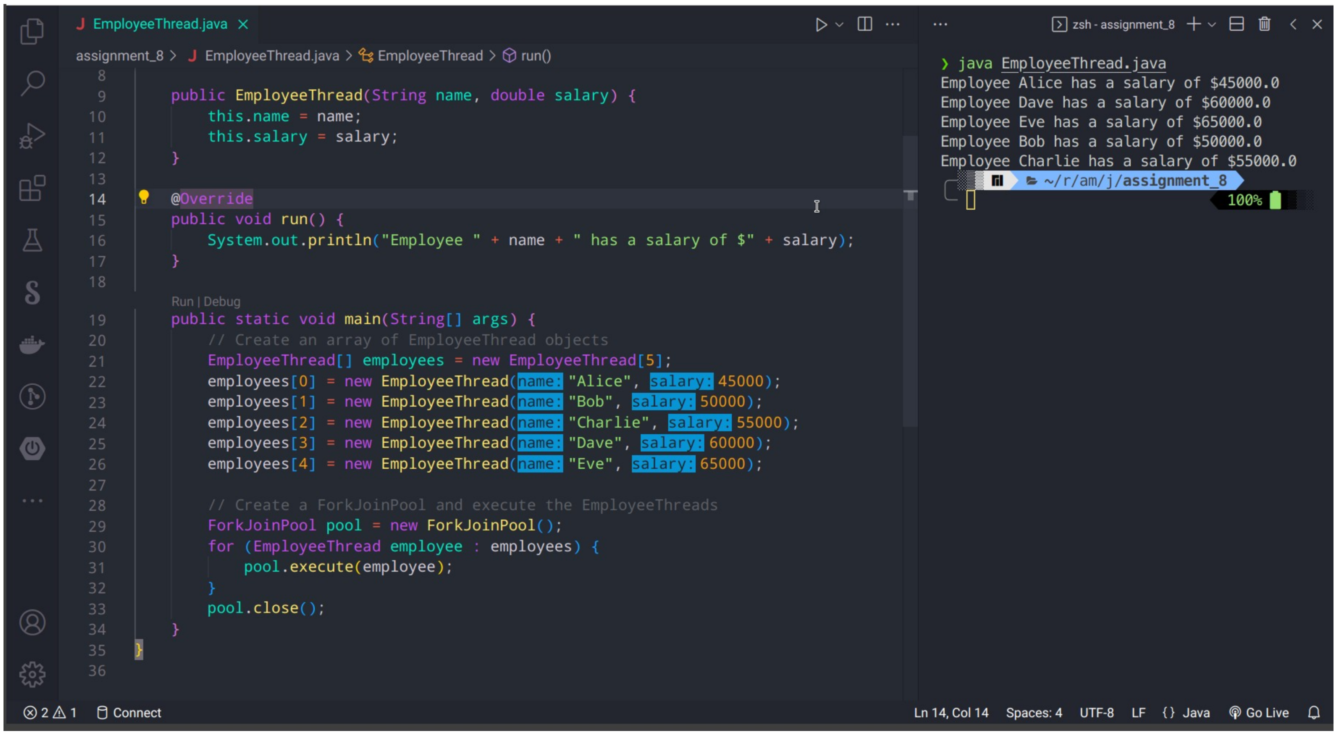
c. Make use of runnable interface in it.

Ans:

a) Code:

```
package assignment_8;

import java.util.concurrent.ForkJoinPool;

public class EmployeeThread implements Runnable {
    private String name;
    private double salary;

    public EmployeeThread(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    @Override
    public void run() {
        System.out.println("Employee " + name + " has a salary of $" + salary);
    }

    public static void main(String[] args) {
        // Create an array of EmployeeThread objects
        EmployeeThread[] employees = new EmployeeThread[5];
        employees[0] = new EmployeeThread("Alice", 45000);
        employees[1] = new EmployeeThread("Bob", 50000);
        employees[2] = new EmployeeThread("Charlie", 55000);
        employees[3] = new EmployeeThread("Dave", 60000);
        employees[4] = new EmployeeThread("Eve", 65000);

        // Create a ForkJoinPool and execute the EmployeeThreads
        ForkJoinPool pool = new ForkJoinPool();
        for (EmployeeThread employee : employees) {
            pool.execute(employee);
        }
        pool.close();
    }
}
```

Screenshot:



```java
    public EmployeeThread(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    @Override
    public void run() {
        System.out.println("Employee " + name + " has a salary of $" + salary);
    }

    public static void main(String[] args) {
        // Create an array of EmployeeThread objects
        EmployeeThread[] employees = new EmployeeThread[5];
        employees[0] = new EmployeeThread(name: "Alice", salary: 45000);
        employees[1] = new EmployeeThread(name: "Bob", salary: 50000);
        employees[2] = new EmployeeThread(name: "Charlie", salary: 55000);
        employees[3] = new EmployeeThread(name: "Dave", salary: 60000);
        employees[4] = new EmployeeThread(name: "Eve", salary: 65000);

        // Create a ForkJoinPool and execute the EmployeeThreads
        ForkJoinPool pool = new ForkJoinPool();
        for (EmployeeThread employee : employees) {
            pool.execute(employee);
        }
        pool.close();
    }
}
```

```
❯ java EmployeeThread.java
Employee Alice has a salary of $45000.0
Employee Dave has a salary of $60000.0
Employee Eve has a salary of $65000.0
Employee Bob has a salary of $50000.0
Employee Charlie has a salary of $55000.0
 ~/r/am/j/assignment_8
                                         100%
```

2.

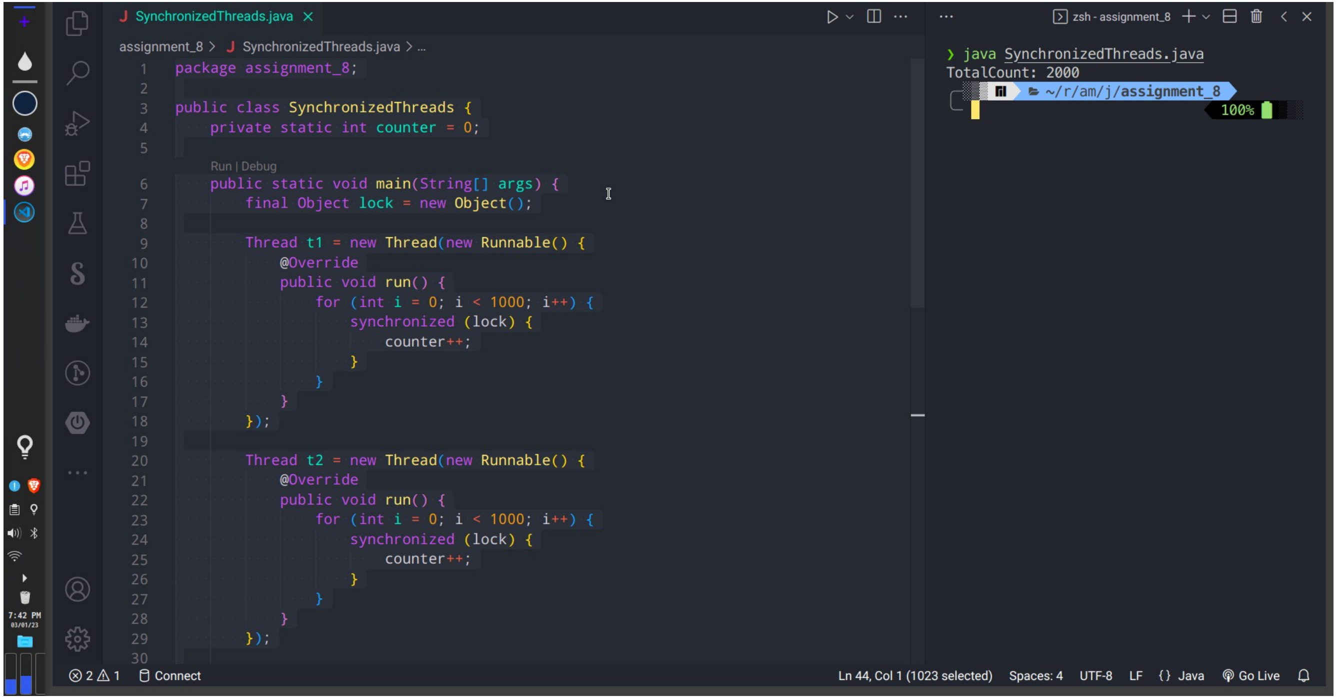
a. Write a program which implements threads with locks by using synchronized keyword.

Ans

a)Code

```java
public class SynchronizedThreads {
    private static int counter = 0;

    public static void main(String[] args) {
        final Object lock = new Object();

        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 1000; i++) {
                    synchronized (lock) {
                        counter++;
                    }
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 1000; i++) {
                    synchronized (lock) {
                        counter++;
                    }
                }
            }
        });

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("TotalCount: " + counter);
    }
}
```

b) Screenhot



SynchronizedThreads.java
assignment_8 > SynchronizedThreads.java > ...
zsh - assignment_8
Run | Debug
Ln 44, Col 1 (1023 selected)   Spaces: 4   UTF-8   LF   {} Java   Go Live
Connect

```
package assignment_8;

public class SynchronizedThreads {
    private static int counter = 0;

    public static void main(String[] args) {
        final Object lock = new Object();

        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 1000; i++) {
                    synchronized (lock) {
                        counter++;
                    }
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 1000; i++) {
                    synchronized (lock) {
                        counter++;
                    }
                }
            }
        });

```

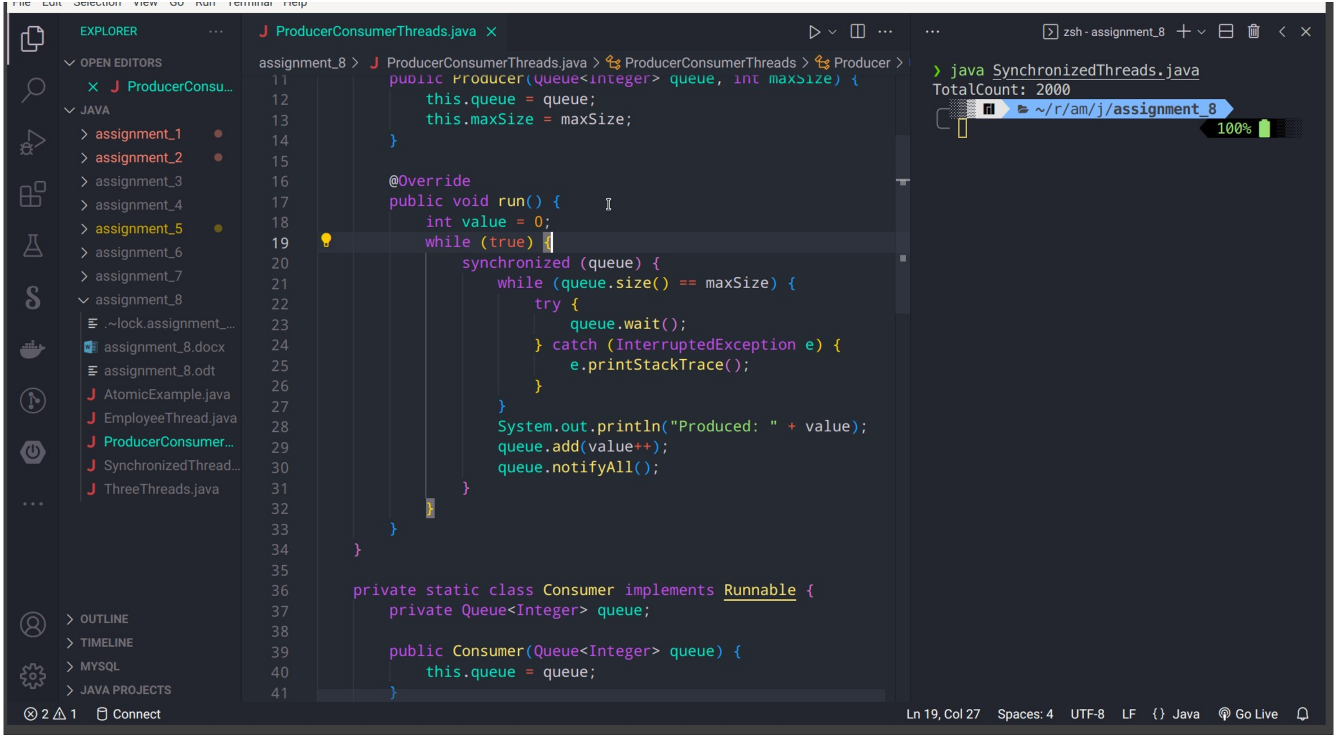
```
❯ java SynchronizedThreads.java
TotalCount: 2000
~/r/am/j/assignment_8
```

b. Write a program which elaborates the concept of producer consumer problem using wait(), notify() & all required functionalities in it.

Ans:
a) Code

```
package assignment_8;

import java.util.LinkedList;
import java.util.Queue;

public class ProducerConsumerThreads {
    private static class Producer implements Runnable {
        private Queue<Integer> queue;
        private int maxSize;

        public Producer(Queue<Integer> queue, int maxSize) {
            this.queue = queue;
            this.maxSize = maxSize;
        }

        @Override
        public void run() {
            int value = 0;
            while (true) {
                synchronized (queue) {
                    while (queue.size() == maxSize) {
                        try {
                            queue.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    System.out.println("Produced: " + value);
                    queue.add(value++);
                    queue.notifyAll();
                }
            }
        }
    }

    private static class Consumer implements Runnable {
        private Queue<Integer> queue;

        public Consumer(Queue<Integer> queue) {
            this.queue = queue;
        }

        @Override
        public void run() {
            while (true) {
                synchronized (queue) {
                    while (queue.isEmpty()) {
                        try {
                            queue.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    int value = queue.poll();
                    System.out.println("Consumed " + value);
                    queue.notifyAll();
                }
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        int maxSize = 5;

        Thread producer = new Thread(new Producer(queue, maxSize));
        Thread consumer = new Thread(new Consumer(queue));

        producer.start();
        consumer.start();
    }
}
```

b) Screenshot



```
        public Producer(Queue<Integer> queue, int maxSize) {
            this.queue = queue;
            this.maxSize = maxSize;
        }

        @Override
        public void run() {
            int value = 0;
            while (true) {
                synchronized (queue) {
                    while (queue.size() == maxSize) {
                        try {
                            queue.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    System.out.println("Produced: " + value);
                    queue.add(value++);
                    queue.notifyAll();
                }
            }
        }
    }

    private static class Consumer implements Runnable {
        private Queue<Integer> queue;

        public Consumer(Queue<Integer> queue) {
            this.queue = queue;
        }
```

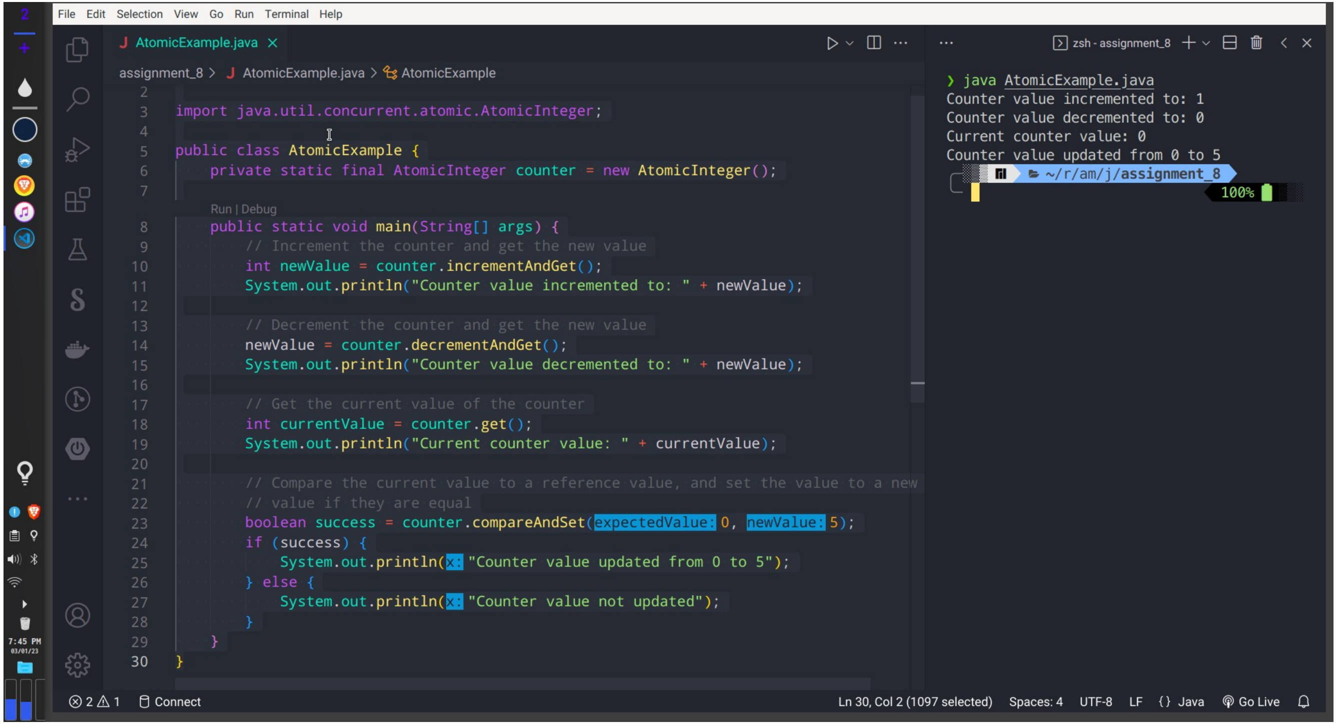
```
❯ java SynchronizedThreads.java
TotalCount: 2000
~/r/am/j/assignment_8
                                        100%
```

c. Write a program with data structure using atomic methods like get(), incrementAndGet(), decrementAndGet(), compareAndSet(). Also use all other functionalities to make the program more responsive.

a) Code:

```
package assignment_8;

import java.util.concurrent.atomic.AtomicInteger;

public class AtomicExample {
    private static final AtomicInteger counter = new AtomicInteger();

    public static void main(String[] args) {
        // Increment the counter and get the new value
        int newValue = counter.incrementAndGet();
        System.out.println("Counter value incremented to: " + newValue);

        // Decrement the counter and get the new value
        newValue = counter.decrementAndGet();
        System.out.println("Counter value decremented to: " + newValue);

        // Get the current value of the counter
        int currentValue = counter.get();
        System.out.println("Current counter value: " + currentValue);

        // Compare the current value to a reference value, and set the value to a
new
        // value if they are equal
        boolean success = counter.compareAndSet(0, 5);
        if (success) {
            System.out.println("Counter value updated from 0 to 5");
        } else {
            System.out.println("Counter value not updated");
        }
    }
}
```

b) Screenshot



File Edit Selection View Go Run Terminal Help
AtomicExample.java
assignment_8 > AtomicExample.java > AtomicExample
zsh - assignment_8
Run | Debug
Ln 30, Col 2 (1097 selected) Spaces: 4 UTF-8 LF {} Java Go Live
⊗ 2 ⚠ 1 Connect

```

import java.util.concurrent.atomic.AtomicInteger;

public class AtomicExample {
    private static final AtomicInteger counter = new AtomicInteger();

    public static void main(String[] args) {
        // Increment the counter and get the new value
        int newValue = counter.incrementAndGet();
        System.out.println("Counter value incremented to: " + newValue);

        // Decrement the counter and get the new value
        newValue = counter.decrementAndGet();
        System.out.println("Counter value decremented to: " + newValue);

        // Get the current value of the counter
        int currentValue = counter.get();
        System.out.println("Current counter value: " + currentValue);

        // Compare the current value to a reference value, and set the value to a new
        // value if they are equal
        boolean success = counter.compareAndSet(expectedValue: 0, newValue: 5);
        if (success) {
            System.out.println(x: "Counter value updated from 0 to 5");
        } else {
            System.out.println(x: "Counter value not updated");
        }
    }
}
```

```
❯ java AtomicExample.java
Counter value incremented to: 1
Counter value decremented to: 0
Current counter value: 0
Counter value updated from 0 to 5
 ~/r/am/j/assignment_8
                                    100%
```

Problem Statement 2: Multithreading

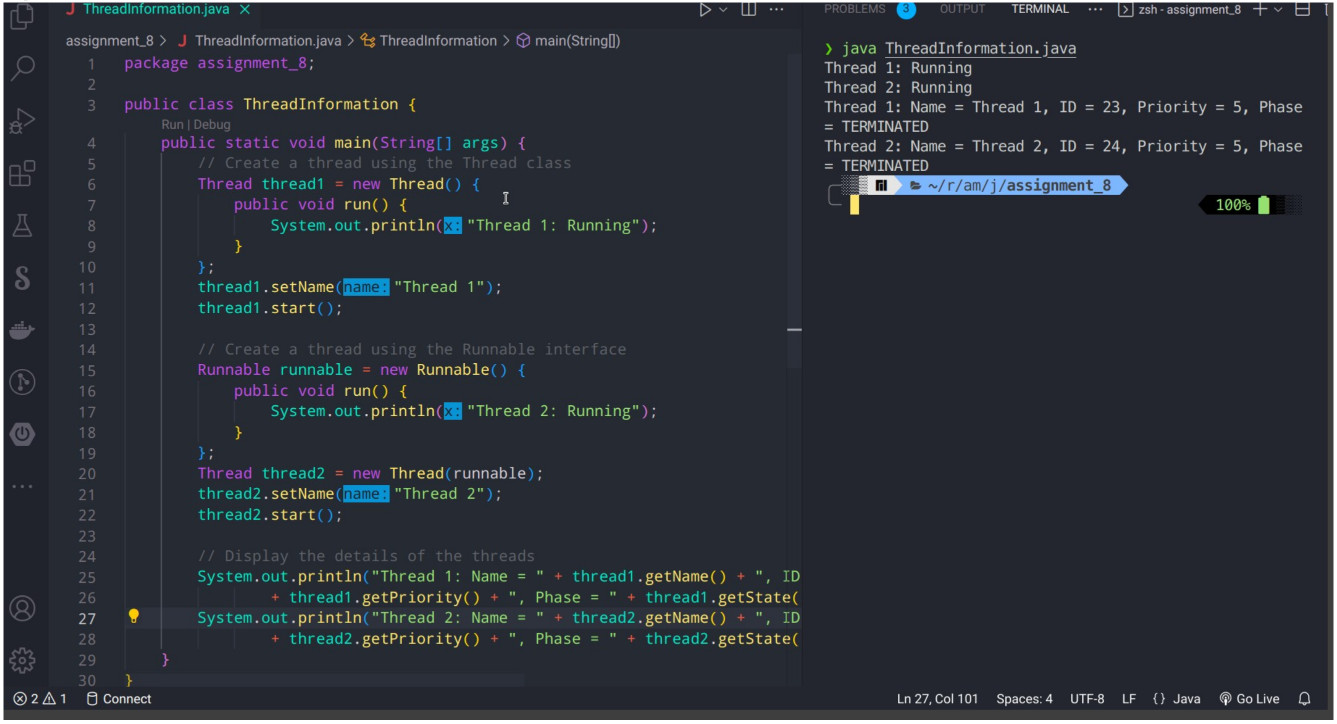
1. Write a Java program to create a thread by using Thread class and also by using

the Runnable interface and display the details of thread like, thread name, id, priority, its phase and other details.

Ans
a) Code

```
package assignment_8;

public class ThreadInformation {
    public static void main(String[] args) {
        // Create a thread using the Thread class
        Thread thread1 = new Thread() {
            public void run() {
                System.out.println("Thread 1: Running");
            }
        };
        thread1.setName("Thread 1");
        thread1.start();

        // Create a thread using the Runnable interface
        Runnable runnable = new Runnable() {
            public void run() {
                System.out.println("Thread 2: Running");
            }
        };
        Thread thread2 = new Thread(runnable);
        thread2.setName("Thread 2");
        thread2.start();

        // Display the details of the threads
        System.out.println("Thread 1: Name = " + thread1.getName() + ", ID = " +
thread1.threadId() + ", Priority = "
                + thread1.getPriority() + ", Phase = " + thread1.getState());
        System.out.println("Thread 2: Name = " + thread2.getName() + ", ID = " +
thread2.threadId() + ", Priority = "
                + thread2.getPriority() + ", Phase = " + thread2.getState());
    }
}
```

b) Screenshot



```
package assignment_8;

public class ThreadInformation {
    public static void main(String[] args) {
        // Create a thread using the Thread class
        Thread thread1 = new Thread() {
            public void run() {
                System.out.println(x: "Thread 1: Running");
            }
        };
        thread1.setName(name: "Thread 1");
        thread1.start();

        // Create a thread using the Runnable interface
        Runnable runnable = new Runnable() {
            public void run() {
                System.out.println(x: "Thread 2: Running");
            }
        };
        Thread thread2 = new Thread(runnable);
        thread2.setName(name: "Thread 2");
        thread2.start();

        // Display the details of the threads
        System.out.println("Thread 1: Name = " + thread1.getName() + ", ID
                + thread1.getPriority() + ", Phase = " + thread1.getState(
        System.out.println("Thread 2: Name = " + thread2.getName() + ", ID
                + thread2.getPriority() + ", Phase = " + thread2.getState(
    }
}
```

```
❯ java ThreadInformation.java
Thread 1: Running
Thread 2: Running
Thread 1: Name = Thread 1, ID = 23, Priority = 5, Phase
= TERMINATED
Thread 2: Name = Thread 2, ID = 24, Priority = 5, Phase
= TERMINATED
 ~/r/am/j/assignment_8
                                          100%
```

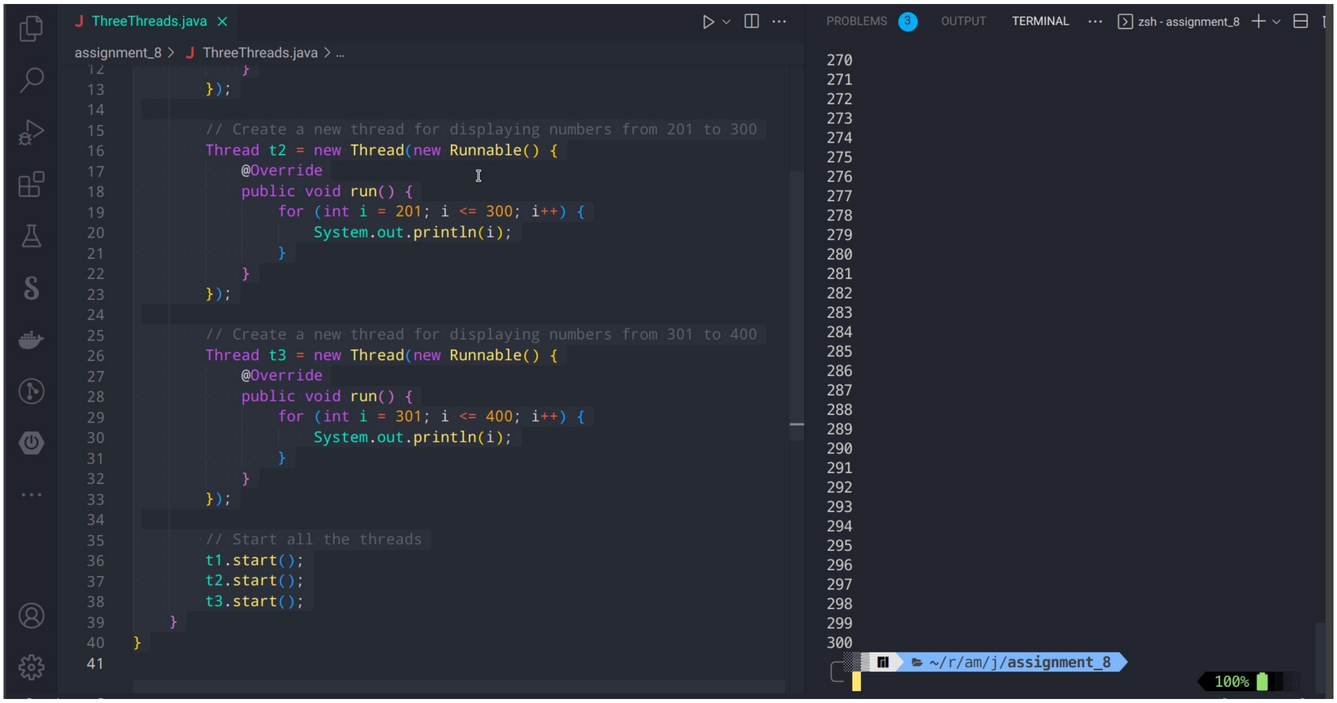
2. Write a Java program to create three different threads, with first thread displaying numbers from 101 to 200, second from 201 to 300 and third from 301 to 400. Verify that all the threads are running simultaneously or not.

Ans

a) Code

```
package assignment_8;

public class ThreeThreads {
    public static void main(String[] args) {
        // Create a new thread for displaying numbers from 101 to 200
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 101; i <= 200; i++) {
                    System.out.println(i);
                }
            }
        });

        // Create a new thread for displaying numbers from 201 to 300
        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 201; i <= 300; i++) {
                    System.out.println(i);
                }
            }
        });

        // Create a new thread for displaying numbers from 301 to 400
        Thread t3 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 301; i <= 400; i++) {
                    System.out.println(i);
                }
            }
        });

        // Start all the threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Screenshot:



```
            }
        });

        // Create a new thread for displaying numbers from 201 to 300
        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 201; i <= 300; i++) {
                    System.out.println(i);
                }
            }
        });

        // Create a new thread for displaying numbers from 301 to 400
        Thread t3 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 301; i <= 400; i++) {
                    System.out.println(i);
                }
            }
        });

        // Start all the threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

```
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
 ~/r/am/j/assignment_8
                                                  100%
```