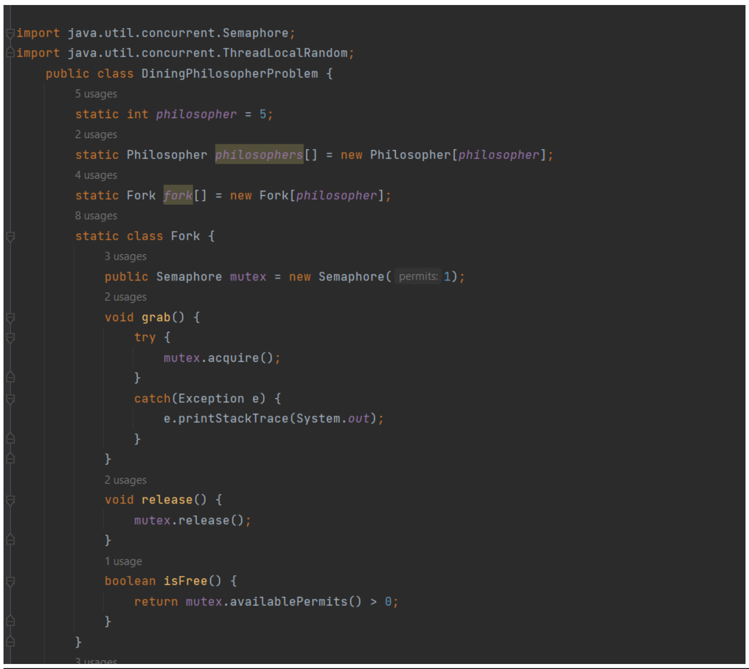
Operating Systems

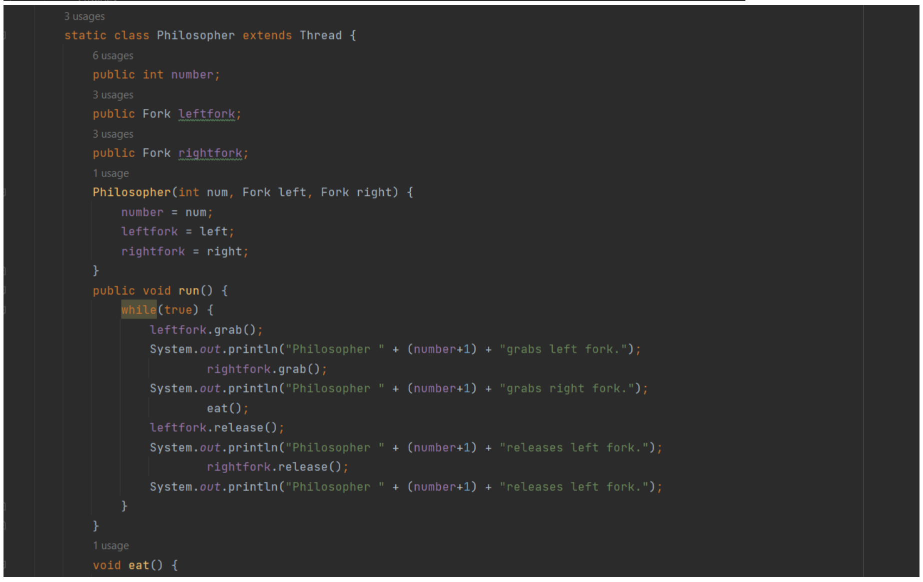
Assignment- 7

1. IPC USING SEMAPHORE – DINING PHILOSOPHERPROBLEM

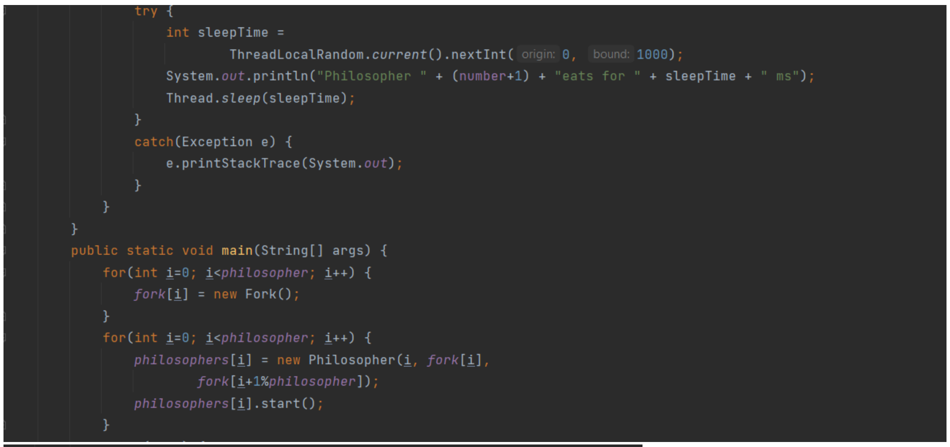
Code:



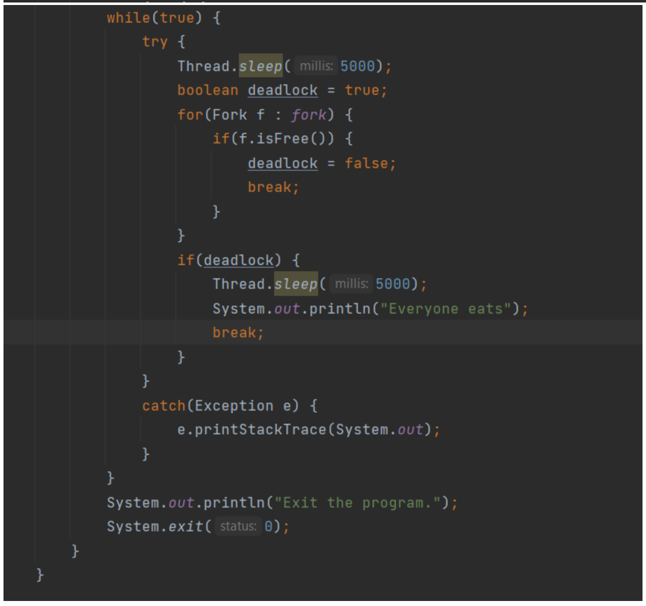
```java
import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadLocalRandom;
    public class DiningPhilosopherProblem {
        static int philosopher = 5;
        static Philosopher philosophers[] = new Philosopher[philosopher];
        static Fork fork[] = new Fork[philosopher];
        static class Fork {
            public Semaphore mutex = new Semaphore( permits: 1);
            void grab() {
                try {
                    mutex.acquire();
                }
                catch(Exception e) {
                    e.printStackTrace(System.out);
                }
            }
            void release() {
                mutex.release();
            }
            boolean isFree() {
                return mutex.availablePermits() > 0;
            }
        }
```



```java
        static class Philosopher extends Thread {
            public int number;
            public Fork leftfork;
            public Fork rightfork;
            Philosopher(int num, Fork left, Fork right) {
                number = num;
                leftfork = left;
                rightfork = right;
            }
            public void run() {
                while(true) {
                    leftfork.grab();
                    System.out.println("Philosopher " + (number+1) + "grabs left fork.");
                        rightfork.grab();
                    System.out.println("Philosopher " + (number+1) + "grabs right fork.");
                        eat();
                    leftfork.release();
                    System.out.println("Philosopher " + (number+1) + "releases left fork.");
                        rightfork.release();
                    System.out.println("Philosopher " + (number+1) + "releases left fork.");
                }
            }
            void eat() {
```

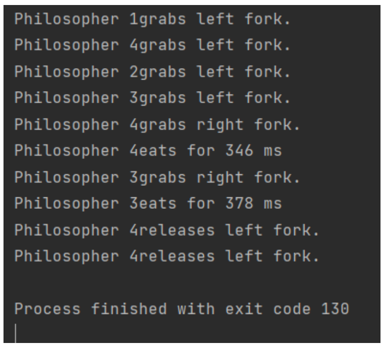


```
            try {
                int sleepTime =
                        ThreadLocalRandom.current().nextInt( origin: 0,  bound: 1000);
                System.out.println("Philosopher " + (number+1) + "eats for " + sleepTime + " ms");
                Thread.sleep(sleepTime);
            }
            catch(Exception e) {
                e.printStackTrace(System.out);
            }
        }
    }
    public static void main(String[] args) {
        for(int i=0; i<philosopher; i++) {
            fork[i] = new Fork();
        }
        for(int i=0; i<philosopher; i++) {
            philosophers[i] = new Philosopher(i, fork[i],
                    fork[i+1%philosopher]);
            philosophers[i].start();
        }
```



```
        while(true) {
            try {
                Thread.sleep( millis: 5000);
                boolean deadlock = true;
                for(Fork f : fork) {
                    if(f.isFree()) {
                        deadlock = false;
                        break;
                    }
                }
                if(deadlock) {
                    Thread.sleep( millis: 5000);
                    System.out.println("Everyone eats");
                    break;
                }
            }
            catch(Exception e) {
                e.printStackTrace(System.out);
            }
        }
        System.out.println("Exit the program.");
        System.exit( status: 0);
    }
}
```

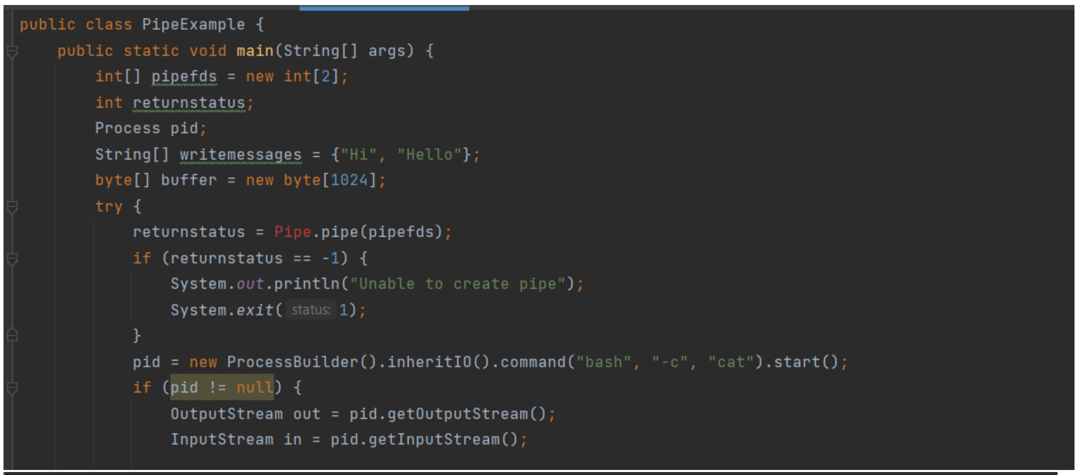
Output:



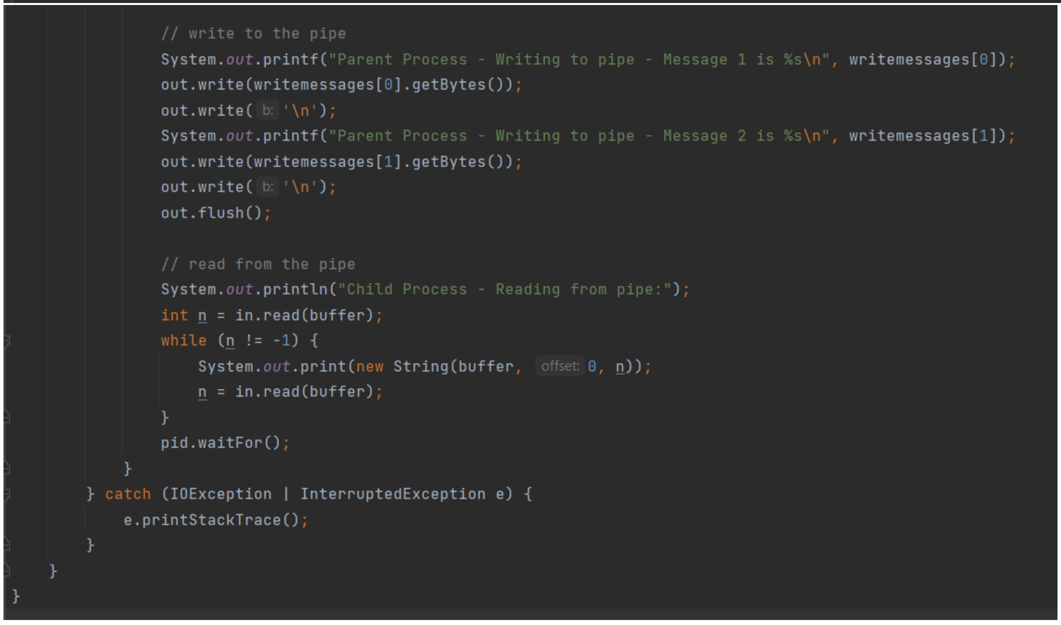
```
Philosopher 1grabs left fork.
Philosopher 4grabs left fork.
Philosopher 2grabs left fork.
Philosopher 3grabs left fork.
Philosopher 4grabs right fork.
Philosopher 4eats for 346 ms
Philosopher 3grabs right fork.
Philosopher 3eats for 378 ms
Philosopher 4releases left fork.
Philosopher 4releases left fork.

Process finished with exit code 130
```

2. IPC USING PIPES
   To Achieve two-way communication using pipes.

Code:

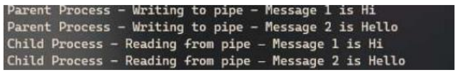


```java
public class PipeExample {
    public static void main(String[] args) {
        int[] pipefds = new int[2];
        int returnstatus;
        Process pid;
        String[] writemessages = {"Hi", "Hello"};
        byte[] buffer = new byte[1024];
        try {
            returnstatus = Pipe.pipe(pipefds);
            if (returnstatus == -1) {
                System.out.println("Unable to create pipe");
                System.exit( status: 1);
            }
            pid = new ProcessBuilder().inheritIO().command("bash", "-c", "cat").start();
            if (pid != null) {
                OutputStream out = pid.getOutputStream();
                InputStream in = pid.getInputStream();
```



```java
                // write to the pipe
                System.out.printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);
                out.write(writemessages[0].getBytes());
                out.write( b: '\n');
                System.out.printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);
                out.write(writemessages[1].getBytes());
                out.write( b: '\n');
                out.flush();

                // read from the pipe
                System.out.println("Child Process - Reading from pipe:");
                int n = in.read(buffer);
                while (n != -1) {
                    System.out.print(new String(buffer,  offset: 0, n));
                    n = in.read(buffer);
                }
                pid.waitFor();
            }
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Output:



```
Parent Process - Writing to pipe - Message 1 is Hi
Parent Process - Writing to pipe - Message 2 is Hello
Child Process - Reading from pipe - Message 1 is Hi
Child Process - Reading from pipe - Message 2 is Hello
```

Submitted by:

Harshita Pasupuleti

21BCE8421