Trabajo Práctico
Programación Concurrente
Thread Safety

Ejercicio 1

Decir V o F a la siguiente afirmación y justificar la respuesta:

Declarar el siguiente método en la clase A:

```
public class A {
    static int method() {
        synchronized (A.class) {
            // "acá va el código que corresponda, con el return correspondiente"
        }
    }
}
```

es equivalente a haberlo codificado de la siguiente forma:

```
public class A {
    static synchronized int method() {
        // "acá va el código que corresponda, con el return correspondiente"
    }
}
```

Ejercicio 2

El siguiente código implementa un Stack. Escribir un código java que utilice este stack y pruebe que se pueden generar inconsistencias del stack debido al mal manejo thread-safety.

```
public class Stack {
    private static final int MAX_SIZE = 10;
    private int top = 0;
    private final int[] values = new int[MAX_SIZE];

    public void push(final int n) {
        if (top == MAX_SIZE) {
            throw new IllegalStateException("stack full");
        }
        values[top++] = n;
    }

    public int pop() {
        if (top == 0) {
            throw new IllegalStateException("stack empty");
        }
        return values[--top];
    }
}
```

Ejercicio 3

Modificar la implementación de Stack del Ejercicio 2 para que sea **Thread Safe.**

Ejercicio 4

Evalúe indicando si son correctos o no para funcionar en un ambiente concurrente, explicando la manera por la cual se puede provocar un error, excepción, inconsistencia o mal uso además de proveer una solución si no lo fueran.

a)

```
public class CountingFactorizer implements Servlet {
    private Long count = 0L;

    public void service(ServletRequest req, ServletResponse resp) {
        synchronized (count) {
            BigInteger i = extractFromRequest(req);
            BigInteger[] factors = factor(i);
            ++count;
            encodeIntoResponse(resp, factors);
        }
    }

    public Long getCount() {
        synchronized (count) {
            return count;
        }
    }
}
```

b)

```
public class ExpensiveObjectFactory {
    private ExpensiveObject instance = null;

    //es un singleton a todos les retorno la misma instancia.
    public ExpensiveObject getInstance() {
        if (instance == null) {
            instance = new ExpensiveObject(); //tarda mucho en construir.
       }
        return instance;
    }
}
```

c)

```
public class Account {
    private double balance;
    private int id;

    public void withdraw(double amount) {
        balance -= amount;
```

```
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public static void transfer(Account from, Account to, double amount) {
        synchronized (from) {
            synchronized (to) {
                from.withdraw(amount);
                to.deposit(amount);
            }
        }
    }

}
```

d)

```
public class MovieTicketsAverager {
    public int tope;
    public List<Movie> movies;

    /**  */
    public MovieTicketsAverager(int tope, List<Movie> movies) {
        super();
        this.tope = tope;
        this.movies = new ArrayList<>(movies);
    }

    public double average() {
        return movies.stream().filter(movie -> movie.getYear() > tope)
                .collect(Collectors.averagingInt(movie -> movie.getTicketsSold()));
    }
}

class Movie {
    private int year;
    private String title;
    private int ticketsSold;

    public Movie(int year, String title, int ticketsSold) {
        super();
        this.year = year;
        this.title = title;
        this.ticketsSold = ticketsSold;
    }

    public int getYear() {
        return year;
    }

    public String getTitle() {
        return title;
    }
```

```
    public int getTicketsSold() {
        return ticketsSold;
    }
}
```

Ejercicio 5

Escribir un programa que reciba como parámetro un directorio y devuelva la cantidad de líneas totales que contienen los archivos de dicho directorio.
Para hacer esta tarea de manera paralela se pide que dicho programa genere un thread por cada archivo y que la tarea a ejecutar por dicho thread sea contar las lineas de dicho archivo o devolver 0 si el archivo era en realidad un directorio.
Una vez que los threads han terminado sumarizar y imprimir en pantalla la cantidad total.