Parte II

1.) Se creo una clase que extiende a Thread, luego obtuviera la fachada y recorriera su segmento de listas que se represento de start a end, por cada lista revisa si la ip esta reportada y si lo está, entonces la guarda el número de esa lista.

- Evidencia Código:

```java
@Getter
@RequiredArgsConstructor
public class BlackListSearchThread  extends Thread{
    private final int start;
    private final int end;
    private final String ip;
    private final List<Integer> occurrences = new LinkedList<>();
    private int count = 0;

    @Override
    public void run() {
        HostBlacklistsDataSourceFacade skds =
                HostBlacklistsDataSourceFacade.getInstance();

        IntStream.range(start, end).forEach(k -> {
            count++;
            if (skds.isInBlackListServer(k, ip)) {
                occurrences.add(k);
            }
        });
    }

}
```

2.) El punto 2 nos pedía paralelizar la búsqueda de una IP en miles de listas negras usando N hilos entre otras condiciones.

Así que se complementó el método de checkHost, se integró N que es el número de hilos y el método decide como dividir el trabajo según ese valor asignándole a cada hilo un segmento de listas de IP usando BlackListSearchThread.

Este método espera a que todos terminen usando join, consolida los resultados y finalmente decide si el host es confiable o no, esto lo determina si aparece en 5 o más listas negras de lo contrario se considera fiable.

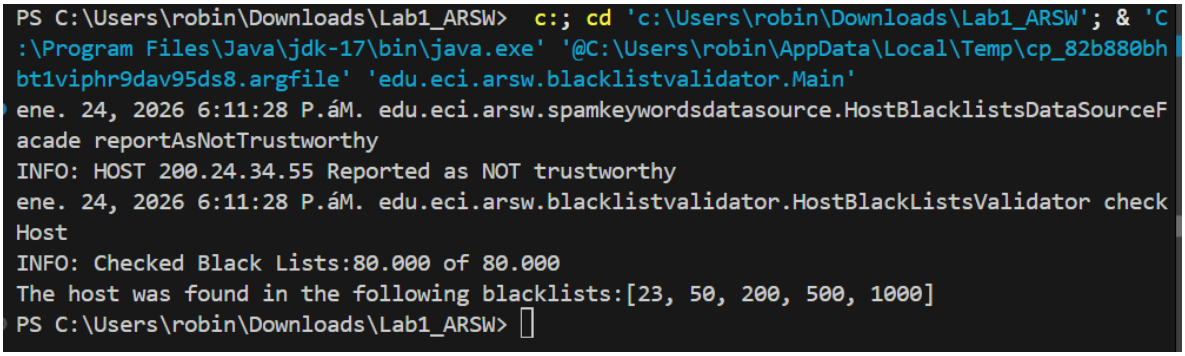
- Código implementado:

```java
public List<Integer> checkHost(String ipaddress, int N){

    LinkedList<Integer> blackListOcurrences=new LinkedList<>();

    int ocurrencesCount=0;

    HostBlacklistsDataSourceFacade skds=HostBlacklistsDataSourceFacade.getInstance();

    int checkedListsCount=0;

    int totalServers = skds.getRegisteredServersCount();
    int segmentSize = totalServers / N;

    List<BlackListSearchThread> threads = new LinkedList<>();

    for (int i = 0; i < N; i++) {
        int start = i * segmentSize;
        int end = (i == N - 1)
                ? totalServers
                : start + segmentSize;

        threads.add(new BlackListSearchThread(start, end, ipaddress));
    }
```

```java
    for (BlackListSearchThread thread : threads) {
        thread.start();
    }

    for (BlackListSearchThread t : threads) {
        try {
            t.join();
        } catch (InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
    }

    for (BlackListSearchThread t : threads) {
        blackListOcurrences.addAll(t.getOccurrences());
        ocurrencesCount += t.getOccurrences().size();
        checkedListsCount += t.getCount();
    }

    if (ocurrencesCount >= BLACK_LIST_ALARM_COUNT) {
        skds.reportAsNotTrustworthy(ipaddress);
    } else {
        skds.reportAsTrustworthy(ipaddress);
    }

    LOG.log(Level.INFO,
            msg: "Checked Black Lists:{0} of {1}",
            new Object[]{checkedListsCount, totalServers});

    return blackListOcurrences;
}
```

Para comprobar la solución se hizo así:

Se usaron 8 hilos en el main, cada hilo revisa un segmento distinto de las listas negras, permitiendo que la búsqueda se ejecute en paralelo y reduciendo el tiempo de ejecución total.



En este caso se probó la IP 200.24.34.55 y se pudo determinar que es peligrosa o no confiable, ya que aparece en al menos 5 listas negras.