Execício semana 07

- Aluna: Analu Sorbara

Exercício 1:

Cinco lebres disputarão uma corrida. Cada lebre pode dar um salto que varia de 1 a 3 metros de distância.

A distância percorrida é de 20 metros. Na corrida, cada lebre dará um salto de comprimento aleatório (dentro do intervalo permitido) e informará quantos metros ela pulou a cada salto realizado. Em seguida, a lebre para para descansar, ficando parada enquanto as outras semana07.lebres saltam. Escreva um programa, utilizando threads (uma para cada lebre), que informe a lebre vencedora e a colocação de cada uma delas no final da corrida. Informar também quantos pulos foram dados por cada uma.

Solução:

- Lebre:

```
import java.util.Random;

class Lebre extends Thread {
    private final String nome;
    private final Corrida t;

    Lebre(String nome, Corrida t) {
        this.t=t;
        this.nome = nome;
    }

    public void fazUmSalto() {
        t.salta(nome, salto());
    }

    private int salto() {
        int minimum = 1;
        int maximum = 3;

        Random r = new Random();
        int range = maximum - minimum + 1;
        return r.nextInt(range) + minimum;
    }
}
```

- Corrida:

```
import java.util.HashMap;
import java.util.Map;

class Corrida {
    Map<String, Integer> corrida = new HashMap<>();
    String ganhou = null;

    synchronized void salta(String lebre, int salto) {
        Integer distanciaPercorrida = corrida.get(lebre);

        if (distanciaPercorrida == null) {
            distanciaPercorrida = 0;
        }

        if (distanciaPercorrida < 20 && ganhou == null) {
            distanciaPercorrida += salto;
            corrida.put(lebre, distanciaPercorrida);
            System.out.println("A lebre: " + lebre + " Pulou: " + salto + " Total: " + distanciaPercorrida);
        } else if (ganhou == null) {
            ganhou = lebre;
        }
    }

    public String getGanhou() {
        return ganhou;
    }
}
```

- Main:

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Corrida corrida = new Corrida();

        List<Lebre> lebres = new ArrayList<>() {{ // Double-Brace Initialization Java 9
            add(new Lebre("Tan", corrida));
            add(new Lebre("Ian", corrida));
            add(new Lebre("Uan", corrida));
            add(new Lebre("Xan", corrida));
            add(new Lebre("Zan", corrida));
        }};

        while (corrida.getGanhou() == null) {
            for (Lebre l : semana07.lebres) {
                l.fazUmSalto();
            }
        }

        System.out.println("Ganhou: " + corrida.getGanhou());

        try {
            Thread.sleep(400);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Exercício 2:

Campeonato de corrida de motocicletas:

- Programa deve criar 10 threads (sendo 10 o número de competidores numerados de 1 a 10);
- Haverão M corridas (M>10);
- Após cada corrida deve-se esperar o término de todos os competidores;
- A competição se daria pela concorrência entre as threads por acesso a região crítica. Assim que a thread conseguisse acessar a região critica receberia o ponto correspondente a sua posição;
- Exemplo de pontuação:

```
1º = 10,

2º = 9 ,

3º = 8 , ...
```

- Ao término do campeonato haveria 1 competidor campeão;
- Campeão o competidor que juntou o maior numero de pontos.

Obs.: No caso de empates o campeão seria decidido por ordem numérica (crescente).

Solução:

- Motocicleta:

```
class Motocicleta implements Runnable {
    private final int nome;
    private final Corrida corrida;

    Motocicleta(Integer nome, Corrida corrida) {
        this.corrida = corrida;
        this.nome = nome;
    }

    @Override
    public void run() {
        corrida.aguardarNaLargada();

        corrida.linhaDeChegada(nome);
    }
}
```

- Corrida:

```
import java.util.*;
import java.util.concurrent.CountDownLatch;

class Corrida {
    CountDownLatch latch;
    List<Integer> ordemDeChegada = new ArrayList<>();
    HashMap<Integer, Integer> pontos = new HashMap<>();
    int maxCorredores;

    public Corrida(int maxCorredores) {
        this.maxCorredores = maxCorredores;
        latch = new CountDownLatch(1);
    }

    public void aguardarNaLargada() {
        try
        {
            latch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public void iniciarCorrida() {
        latch.countDown();
    }

    public void novaCorrida() {
        ordemDeChegada = new ArrayList<>();
        latch = new CountDownLatch(1);
    }

    public synchronized void linhaDeChegada(Integer nome) {
        Integer pontucao = pontos.get(nome);

        if (pontucao == null) {
            pontucao = 0;
        }
        pontucao += ponto(nome);
        pontos.put(nome, pontucao);
    }

    private synchronized Integer ponto(Integer nome) {
        if (!ordemDeChegada.contains(nome)) {
            ordemDeChegada.add(nome);
        }

        return maxCorredores - ordemDeChegada.indexOf(nome);
    }

    public void printGanhador() {
        Integer ganhador = Collections.max(pontos.entrySet(), Comparator.comparingInt(Map.Entry::getValue)).getKey();
        Integer totalPontos = pontos.get(ganhador);

        System.out.println("Ganhador: " + ganhador.toString() + " Pontos: " + totalPontos);
    }
}
```

- Main:

```
 import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        List<Motocicleta> motocicletas = new ArrayList<>();
        int maxCorredores = 10;
        int numeroCorridas = 11;

        Corrida corrida = new Corrida(maxCorredores);

        for (int i = 1; i <= maxCorredores; i++) {
            motocicletas.add(new Motocicleta(i, corrida));
        }

        for (int i = 0; i < numeroCorridas; i++) {
            List<Thread> threads = new ArrayList<>();
            corrida.novaCorrida();
            for (Motocicleta l : motocicletas) {
                Thread t = new Thread(l);
                t.start();
                threads.add(t);
            }
            Thread.sleep(100);
            corrida.iniciarCorrida();
            for (Thread t : threads) {
                t.join();
            }
            Thread.sleep(100);
        }

        corrida.printGanhador();
    }
}
```