Samir Monteiro Ibrahim

Software Concorrente e Distribuído

1.

```
public class exerc1{
    public static void main(String[] args) {
        ThreadSimples t01 = new ThreadSimples("Thread 01");
        t01.run();
    }
}
class ThreadSimples extends Thread{
    String nome;

    public ThreadSimples (String nome){
        this.nome = nome;
    }
    @Override
    public void run(){
        for(int i = 1; i <= 100; i++){
            System.out.println(i + " - " + nome);
        }
        System.out.println("Processo finalizado");
    }
}
```

2.

```
public class exerc1{
    public static void main(String[] args) {
        ThreadSimples t01 = new ThreadSimples("Melao");
        ThreadSimples t02 = new ThreadSimples("Banana");
        ThreadSimples t03 = new ThreadSimples("Mamao");
        ThreadSimples t04 = new ThreadSimples("Laranja");
        ThreadSimples t05 = new ThreadSimples("Uva");
        t01.run();
        t02.run();
        t03.run();
        t04.run();
```

```
        t05.run();
    }
}
class ThreadSimples extends Thread{
    String nome;

    public ThreadSimples (String nome){
        this.nome = nome;
    }
    @Override
    public void run(){
        System.out.println(nome);
    }
}
```

Não consegui entender o método sleep/interrupt. Tive a ideia de pausar a thread após ter dado o print e após 3 segundo limpar o que tava no console e chamar outra thread até acabar, mas não consegui.

3. Isso é possível de acontecer, com um programa que foi mal escrito e não leva em consideração processos concorrentes, trazendo erros que podem causar danos críticos para o comportamento do mesmo. Para contornar esse problema, usaria técnicas de programação para tratar com sistemas que têm funcionalidades concorrentes, visando encontrar a região crítica daquela propriedade e não deixando a execução de dois processos que disputam recursos que podem causar um erro.