**RICARDO LEONARDO PANSONATO 29SCJ**

1. Crie um sistema que inicie duas threads. Uma ficará enviando mensagens pré-configuradas de uma lista definida a cada 20 segundos. Quando todas as mensagens forem exibidas, a thread deverá se encerrar. A outra thread mostrará a cada 10 a data e hora atual no formato "dia/mês/ano hora:minuto:segundo" (10/09/2012 14:00:33). Esta última thread funcionará em loop eterno. Após o início de ambas as threads, crie um loop eterno que apresente o estado de cada uma das threads a cada 5 segundos.

**import** java.time.LocalDateTime;

**import** java.time.format.DateTimeFormatter;

**import** java.util.Arrays;

**public** **class** Exercicio01 {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Runnable firstTaskWithoutScheduler = () -> {

**for** (String message : Arrays.*asList*("message 1", "message 2", "message 3")) {

**try** {

System.***out***.println("Thread[1]: " + message);

Thread.*sleep*(20000);

} **catch** (InterruptedException e) {

}

}

};

Thread firstThreadWithoutScheduler = **new** Thread(firstTaskWithoutScheduler);

Runnable secondTaskWithoutScheduler = () -> {

**while** (**true**) {

**try** {

String value = DateTimeFormatter

.*ofPattern*("dd/MM/yyyy hh:mm:ss")

.format(LocalDateTime.*now*());

System.***out***.println("Thread[2]: " + value);

Thread.*sleep*(10000);

} **catch** (InterruptedException e) {

}

}

};

Thread secondThreadWithoutScheduler = **new** Thread(secondTaskWithoutScheduler);

**while** (**true**) {

System.***out***.println("Thread[1] - State: " + firstThreadWithoutScheduler.getState());

System.***out***.println("Thread[2] - State: " + secondThreadWithoutScheduler.getState());

**if** (firstThreadWithoutScheduler.getState().equals(Thread.State.***NEW***)) {

firstThreadWithoutScheduler.start();

}

**if** (secondThreadWithoutScheduler.getState().equals(Thread.State.***NEW***)) {

secondThreadWithoutScheduler.start();

}

Thread.*sleep*(5000);

}

}

}

1. Implemente com o uso adequado de threads o seguinte problema abaixo.

Simulador de Aeroporto

Crie a classe Aviao para a simulação de voos em um aeroporto com as seguintes características

* Deverá receber no construtor o nome do voo e o objeto da classe Aeroporto (será explicado adiante).
* Deverá possuir os métodos decolar, voar e aterrissar.
* No método voar, coloque uma espera de 2 segundos.
* No método decolar e aterrissar deverá verificar se na classe Aeroporto possui pista disponível por meio do método aguardarPistaDisponivel.

Crie a classe Aeroporto para simulação de um aeroporto com as seguintes características.

* Deverá receber no construtor o nome do aeroporto.
* Deverá ter um método chamado aguardarPistaDisponivel onde verificará a disponibilidade da pista. Caso a pista esteja disponível, deverá aguardar até ficar disponível.
* Deverá ter um método chamado alterarEstadoPista que será responsável por alterar o estado da pista (verdadeiro/falso). A cada atualização do estado da pista deverá notificar as threads desta condição para continuidade das operações (aterrissagem e decolagem).
* O estado da pisa deverá ser alterado a cada 3 segundos.

As classes Aeroporto e Aviao deverão herdar a classe Thread.

Crie uma classe Main para simular o comportamento das threads. Para isso, crie 3 objetos da classe Aviao para simular os voos: “TAN 123”, “OLÉ 111” e “LINHA 222” e analise o comportamento deles no aeroporto de acordo com o estado da pista.

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** Aeroporto **extends** Thread {

**private** List<Aviao> avioes = **new** ArrayList<>();

**private** **boolean** aguardarPistaDisponivel = **false**;

**private** String nomeAeroporto;

**public** Aeroporto(String nomeAeroporto) {

**this**.nomeAeroporto = nomeAeroporto;

}

@Override

**public** **void** run() {

**while**(**true**) {

**try** {

Thread.*sleep*(3000);

alterarEstadoPista();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**public** **void** alterarEstadoPista() {

**this**.aguardarPistaDisponivel = !aguardarPistaDisponivel;

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println(!aguardarPistaDisponivel ? nomeAeroporto + " permite decolar/aterrisar" : nomeAeroporto + " não permite decolar/aterrisar");

**if** (!aguardarPistaDisponivel) {

avioes.forEach(a -> {

**synchronized** (a) {

a.notify();

}

});

}

avioes.forEach(a -> { System.***out***.println(a.getVoo() + " - " + a.getState()); });

}

**public** **boolean** aguardarPistaDisponivel() {

**return** aguardarPistaDisponivel;

}

**public** String getNomeAeroporto() {

**return** nomeAeroporto;

}

**public** **void** add(Aviao aviao) {

**this**.avioes.add(aviao);

}

}

**public** **class** Aviao **extends** Thread {

**private** String voo;

**private** Aeroporto aeroporto;

**public** Aviao(String voo, Aeroporto aeroporto) {

**super**();

**this**.voo = voo;

**this**.aeroporto = aeroporto;

aeroporto.add(**this**);

}

@Override

**public** **void** run() {

**while**(**true**) {

**try** {

decolar();

voar();

aterrisar();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**public** String getVoo() {

**return** voo;

}

**public** Aeroporto getAeroporto() {

**return** aeroporto;

}

**public** **void** voar() **throws** InterruptedException {

System.***out***.println(voo + " - Voando... ");

Thread.*sleep*(2000);

}

**public** **void** decolar() **throws** InterruptedException {

**if**(aeroporto.aguardarPistaDisponivel()) {

**synchronized** (**this**) {

**this**.wait();

}

}

System.***out***.println(voo + " - Decolando... ");

}

**public** **void** aterrisar() **throws** InterruptedException {

**if**(aeroporto.aguardarPistaDisponivel()) {

**synchronized** (**this**) {

**this**.wait();

}

}

System.***out***.println(voo + " - Aterrisando... ");

}

}

**public** **class** Exercicio02 {

**public** **static** **void** main(String[] args) {

Aeroporto aeroporto = **new** Aeroporto("GRU");

Aviao aviao1 = **new** Aviao("TAN 123", aeroporto);

Aviao aviao2 = **new** Aviao("OLÉ 111", aeroporto);

Aviao aviao3 = **new** Aviao("LINHA 222", aeroporto);

aeroporto.start();

aviao1.start();

aviao2.start();

aviao3.start();

}

}

1. Crie uma classe enum chamada Cores que informe ao sistema o código de determinadas cores que serão mostrados a seguir. Esta classe deve possuir:

**Atributos**

* 1. Nome da Cor
  2. Código da Cor

**Métodos**

* 1. Recuperar Nome da Cor
  2. Recuperar Código da Cor
  3. Construtor deve receber como parâmetro o código e o nome da Cor

Depois de criada a classe enum Cores, crie uma classe Main para mostrar em tela todas as cores, informando seu nome e seu código. Por fim, mostre em tela especificamente o código da cor Branca.

|  |  |
| --- | --- |
| Cor | Código |
| Branco | 21 |
| Preto | 22 |
| Vermelho | 23 |
| Amarelo | 24 |
| Azul | 25 |

**public** **enum** Cores {

***BRANCO***(21, "Branco"),

***PRETO***(22, "Preto"),

***VERMELHO***(23, "Vermelho"),

***AMARELO***(24, "Amarelo"),

***AZUL***(25, "Azul");

**private** Integer codigo;

**private** String cor;

**private** Cores(Integer codigo, String cor) {

**this**.codigo = codigo;

**this**.cor = cor;

}

**public** Integer getCodigo() {

**return** codigo;

}

**public** String getCor() {

**return** cor;

}

@Override

**public** String toString() {

**return** **this**.codigo + " - " + **this**.cor;

}

}

**import** java.util.Arrays;

**public** **class** Exercicio03 {

**public** **static** **void** main(String[] args) {

Arrays.*asList*(Cores.**class**.getEnumConstants()).forEach(c -> {

System.***out***.println(c.toString());

});

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println(Cores.***BRANCO***);

}

}