1) Fazer uma aplicação que rode 5 Threads que cada uma delas imprima no console o seu número (TID).
Dica: O número que deve ser impresso é a saída da operação int id = getId() (Para JDK até 18) ou int id = threadId() (Para JDK a partir da 19) da Thread.

2) Fazer uma aplicação que insira números aleatórios em uma matriz 3 x 5 e tenha 3 chamadas de Threads, onde cada uma calcula a soma dos valores de cada linha, imprimindo a identificação da linha e o resultado da soma.
Dica: A main deve gerar uma matriz com números aleatórios, mas a Thread recebe um vetor (uma linha da matriz)

3) Criar em java um projeto com uma Thread chamada ThreadVetor, que receba um valor numérico e vetor como parâmetros. Caso o valor numérico seja par, a thread deve percorrer o vetor utilizando uma estrutura for (int i = 0 ; i < vet.length; i++) e contar o tempo para percorrer o vetor. Caso o valor numérico seja ímpar, a thread deve percorrer o vetor utilizando uma estrutura foreach e contar o tempo para percorrer o vetor. No final, ambas as possibilidades devem apresentar o tempo em segundos. A operação main deve gerar um vetor de 1000 posições com valores aleatórios de 1 a 100. Deve iniciar 2 ThreadVetor e para uma passar o número 1 e o vetor e para a outra, passar o número 2 e o mesmo vetor.

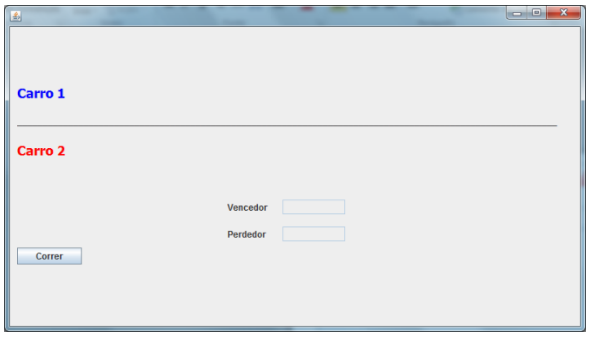
4) Fazer uma aplicação de uma corrida de sapos, com 5 Threads, cada Thread controlando 1 sapo. Deve haver um tamanho máximo para cada pulo do sapo (em metros) e a distância máxima para que os sapos percorram. A cada salto, um sapo pode dar um salto de 0 até o tamanho máximo do salto (valor aleatório). Após dar um salto, a Thread, para cada sapo, deve mostrar no console, qual foi o tamanho do salto e quanto o sapo percorreu. Assim que o sapo percorrer a distância máxima, a Thread deve apresentar que o sapo chegou e qual sua colocação.
Dica: O exercício deve ser resolvido todo em console, ou seja, como se estivesse sendo narrado. Atenção para a forma de definir a ordem de chegada.

5) No Sistema Operacional Linux, o comando para realizar uma operação de ping com 10 iterações é ping -4 -c 10 <servidor>. Fazer uma aplicação Java que rode 3 Thread em S.O. Linux fazendo operação de ping para os servidores UOL (www.uol.com.br), Terra (www.terra.com.br) e Google (www.google.com.br). Cada thread deve ler a saída do ping imprimindo, a cada iteração, o nome do servidor (usar fixo: UOL, Terra ou Google) e o tempo daquela iteração. Ao fim, deve-se exibir o nome do servidor (usar fixo: UOL, Terra ou Google) e o tempo médio obtido pela operação. Outros Sistemas Operacionais devem ser descartados.

Para as questões 6 e 7 (Não obrigatórias), devem ser utilizados como materiais extras:

- Para aprender o básico, que deverá ser aprimorado depois, sobre o WindowBuilder, assistir aos vídeos Instalando Window Builder no Eclipse (http://www.youtube.com/watch?v=veJi6Ndy8E4)
- Introdução ao Window Builder (http://www.youtube.com/watch?v=r2WQfEswJlI)
- Códigos Exemplos com JFrame e Threads:
  - https://github.com/lecolevati/ThreadBolinha
  - https://github.com/lecolevati/ExemploThreadsAninhadas

6) (Desafio) Utilizando o Java SWING, criar uma tela, semelhante à tela abaixo, para criar uma corrida de carros, tipo *drag race*. A aplicação deve ter a distância que os carros devem correr e a velocidade máxima dos carros. Os carros (Jlabel) devem, a cada 100 mS, dar uma arrancada de velocidade que pode estar entre 0 e a velocidade máxima (definida aleatoriamente). Assim que o primeiro carro chegar, o JTextField Vencedor deve receber o nome deste e o JTextField Perdedor receberá o nome do outro carro. Assim que a corrida se inicia, o botão Correr deve sumir.



7) (Desafio) Fazer, com o Java SWING, uma aplicação de caça-níquel, conforme figura abaixo. O caça níquel tem 3 JTextFields, independentes, que giram, aleatoriamente, de 1 a 150 vezes e apresentará um número de 1 a 7. Quando iniciado, o botão Jogar deve desaparecer.