Pergunta 1

0,25 em 0,25 pontos

Com relação a Exceções, qual das afirmações abaixo está **incorreta**?

Resposta Selecionada: a. Ao lançarmos uma Exceção com o termo throw a partir de um método, o mesmo gerará uma interrupção de execução de bloco. Desta forma, se não tratarmos a Exceção ao chamarmos aquele método, o sistema volta ao status inicial de reinício do programa.

Respostas:
a. Ao lançarmos uma Exceção com o termo throw a partir de um método, o mesmo gerará uma interrupção de execução de bloco. Desta forma, se não tratarmos a Exceção ao chamarmos aquele método, o sistema volta ao status inicial de reinício do programa.
b. As exceções que não obrigam o tratamento via estrutura try-catch são aquelas que herdam RuntimeException.
c. Em controle de Exceções, quando queremos obrigar o tratamento de Exceções toda vez que chamamos um determinado método, inserimos ao final da declaração daquele método o termo "throws" seguido do nome da Exceção que queremos invocar.
d. É possível tratarmos Exceções específicas em uma estrutura try-catch. Neste caso, se a Exceção que ocorrer não for uma daquelas especificadas nas capturas (parte dos "catch"), a interrupção do programa, e consequentemente do Servidor, ocorrerá mesmo assim.
e. O lançamento de Exceções (com o termo throw) gera uma interrupção na execução de um bloco de comandos.

Comentário da resposta: Resposta: A
Comentário: De acordo com o conceito de exceções, vimos que uma linha de comando com o termo "throw" lança (gera) uma interrupção na execução do método que está sendo executado, retornando (gerando) na JVM um objeto do tipo "Exception" que, se não for devidamente tratado, pode gerar uma interrupção não só no método, mas também desativando todas as outras execuções da JVM, causando o que chamamos de "travamento do servidor" (e não o "retorno a um status inicial"). Vimos que para evitar esse "travamento" precisamos "tratar as exceções" devidamente, de forma que com isso, apesar de não evitarmos a ocorrência da exceção, evitamos a desativação da JVM.

Pergunta 2

0,25 em 0,25 pontos

Imagine que você foi chamado para verificar um sistema construído em Java e suponha que neste sistema exista um conjunto de códigos que pode gerar alguma exceção, mas que não se sabe exatamente qual das exceções ocorrerá. Qual das opções

Resposta Selecionada: c. Envolver aquele conjunto de código em uma estrutura try-catch, capturando uma exceção genérica (como por exemplo a "Exception").

Respostas:
a. Retirar (excluir) o conjunto de código do sistema.
b. Estudar linha por linha e gerar uma estrutura condicional (if-else if) evitando que a exceção ocorra.
c. Envolver aquele conjunto de código em uma estrutura try-catch, capturando uma exceção genérica (como por exemplo a "Exception").
d. Gerar uma estrutura "try-catch" capturando todas as exceções possíveis da biblioteca do java.
e. Esperar que a exceção ocorra para se conseguir visualizar o seu nome e então envolver o conjunto de códigos capturando aquela exceção.

Comentário da resposta: Resposta: C
Comentário: Vimos que a captura de uma exceção (com o termo "catch") deve ser feita de forma correta, pois se capturarmos uma exceção que não aquela que ocorreu, podemos não conseguir evitar o travamento da JVM. Sendo assim, quando temos uma situação em que não sabemos qual exceção ocorrerá, uma solução rápida que evita que ocorra o "travamento" é envolver o grupo de código que pode gerar a exceção com um bloco do tipo "try-catch", capturando no bloco "catch" uma exceção mais genérica (uma das superclasses das exceções) que pode ser a classe "Exception" ou a classe "Throwable", cujo objeto pode se comportar como qualquer uma das exceções (já que mesmo as exceções criadas no sistema do cliente, por padrão herdam a classe Exception).

Pergunta 3

0,25 em 0,25 pontos

Na hierarquia das exceções (e dos erros) da linguagem Java, qual é a exceção (ou o erro) mais genérica, cujo objeto que a representa pode se comportar como qualquer exceção possível do sistema?

Resposta Selecionada: b. Throwable.

Respostas:
a. Exception.
b. Throwable.
c. RuntimeException.
d. SQLExcception.
e. GeneralException.

Comentário da resposta: Resposta: B
Comentário: Vimos que na hierarquia das exceções, a classe mãe de todas as exceções do Java, ou criadas em sistemas feitos na linguagem Java nos clientes, é a classe "Throwable", cujo objeto que a representa pode se comportar (característica de polimorfismo de classes) como qualquer uma de suas classes filhas.

Pergunta 4

0,25 em 0,25 pontos

Ao se lançar uma exceção, qual das opções abaixo mostra a exceção que não obriga o tratamento ao se invocar um método que a lança?

Resposta Selecionada: d. RuntimeException.

Respostas:
a. Exception.
b. Throwable.
c. SQLExcception.
d. RuntimeException.
e. IOExcception.

Comentário da resposta: Resposta: D
Comentário: Na hierarquia das exceções, existe uma classe, filha da classe "Exception", que apesar de ser uma exceção, é um tipo de exceção que não obriga o tratamento, ou ainda, o envolvimento com o bloco "try-catch", da invocação dos métodos que a lançam: que é a exceção do tipo "RuntimeException" e suas subclasses. Caso um método a lance, quando formos invocar aquele método, enquanto programamos o sistema, a IDE não obrigará o envolvimento com o bloco "try-catch", de forma que caberá ao desenvolvedor conhecer a possibilidade daquele método lançar uma exceção e consequentemente travar a JVM.

Pergunta 5

0,25 em 0,25 pontos

Em um sistema criado na linguagem Java, existe numa determinada classe um método cujo nome é "verificaImposto()" sem parâmetros e que retorna um valor do tipo double. Sabendo que este método é público, a equipe de desenvolvimento (de acordo com as regras de salários da empresa) o construiu de modo que ele lance uma exceção do tipo "Exception" caso se trabalhe com determinados valores de salários. Desta forma, qual das opções abaixo mostra uma declaração daquele método, de forma a obrigar o tratamento com "try-catch" sempre que este método for invocado por métodos de outras classes?

Resposta Selecionada: e. public double verificaImposto () throws Exception {

Respostas:
a. public double verificaImposto (throw Exception) {
b. public double verificaImposto () {
c. public Exception verificaImposto () {
d. public double verificaImposto () throws RuntimeException {
e. public double verificaImposto () throws Exception {

Comentário da resposta: Resposta: E
Comentário: Vimos que o termo "throws", utilizado na declaração de um método de uma classe, quando lançando uma exceção que não seja do tipo "RuntimeException", obriga o envolvimento com "try-catch" (o seu "tratamento") a toda linha de comando que chamar aquele método. Para que um método lance esse tipo de exceção, a sintaxe é:
public tipoRetorno nomeDoMetodo(parametros) throws TipoDaException {...}

Pergunta 6

0,25 em 0,25 pontos

Para se estar sempre de acordo com as regras de negócio de uma empresa, algumas vezes é necessário que se criem novas exceções que definem a ideia de cada uma daquelas regra, e lançá-las sempre que, durante a utilização daquele sistema, aquelas regras forem quebradas. Para se criar uma exceção, basta:

Resposta Selecionada: a. Criar uma Classe que represente a regra a ser seguida e fazer com que ela herde a classe "Exception", gerando nela Métodos Construtores que acionem o método construtor da classe herdada (com a palavra reservada "super").

Respostas:
a. Criar uma Classe que represente a regra a ser seguida e fazer com que ela herde a classe "Exception", gerando nela Métodos Construtores que acionem o método construtor da classe herdada (com a palavra reservada "super").
b. Lançar a exceção "IllegalArgumentException".
c. Criar a classe Exception.
d. Envolver todo o código da classe com o bloco try-catch.
e. Criar uma classe cujo nome termine com o termo "Exception".

Comentário da resposta: Resposta: A
Comentário: Cria-se uma exceção gerando-se uma classe que representa a regra de negócio que deve ser controlada, cujo nome deve conter, por padrão, uma palavra que lembre a regra, terminando com o termo "Exception", deve herdar uma Exceção existente e que, por padrão, contenha basicamente a seguinte sintaxe (geralmente com dois métodos construtores, um com parâmetro do tipo String, e outro sem parâmetro enviando uma mensagem padrão):

```
public class NomeDaRegraException extends Exception {
  public NomeDaRegraException () {
   super("Mensagem padrão!!");
  }
  public NomeDaRegraException (String msg) {
   super(msg);
  }
}
```

Pergunta 7

0,25 em 0,25 pontos

Ao gerarmos uma classe que implementará uma Thread, qual o nome do método que deve existir nesta classe, que será acionado sempre que a Thread for inicializada?

Resposta Selecionada: e. run.

Respostas:
a. start.
b. main.
c. acionarThread.
d. iniciar.
e. run.

Comentário da resposta: Resposta: E
Comentário: Vimos que ao darmos um "start" em uma "Thread", estamos invocando automaticamente o método "run()" criado na classe que representa aquela "thread".

Pergunta 8

0,25 em 0,25 pontos

Qual o nome da interface que deve ser implementada por uma classe para que ela possa ser utilizada através de threads em um sistema, sem que aquela classe herde a classe "Thread"?

Resposta Selecionada: b. Runnable.

Respostas:
a. Throwable.
b. Runnable.
c. Thread.
d. Exception.
e. Runtime.

Comentário da resposta: Resposta: B
Comentário: A forma mais usual dentre as possíveis formas de se gerar classes que serão geradas rodando em "paralelo" na JVM (caracterizando um multiprocessamento) é aquela em que criamos uma classe em que implementamos a interface "Runnable", obrigando com isso a criação do método "run()", de forma que para que tenhamos processamento em paralelo, geramos uma Thread em cujo método construtor enviamos a classe que implementa "Runnable" e acionamos o método "start" daquela "Thread".

Pergunta 9

0,25 em 0,25 pontos

Com relação a Threads, analise as afirmações abaixo:

I - A linguagem Java não permite o multiprocessamento de Threads.
II - Utilizamos Threads para executarmos mais de um método em paralelo (ao mesmo tempo).
III - Para que uma classe possa ser utilizada com Threads, ela deve herdar a classe Thread ou implementar a interface Runnable, além de possuir o método run() que é acionado ao darmos o "start" no Thread.
IV - Para acionarmos uma Thread, basta que invoquemos diretamente o método "run()" da classe que a implementa.

De acordo com as afirmações acima, assinale a alternativa correta:

Resposta Selecionada: c. Somente II e III estão corretas.

Respostas:
a. Somente a II está correta.
b. Somente I, IV e V estão corretas.
c. Somente II e III estão corretas.
d. Somente II e V estão corretas.
e. Somente III e IV estão corretas.

Comentário da resposta: Resposta: C
Comentário: Vimos que para que tenhamos um multiprocessamento em paralelo em programas gerados na linguagem Java, nos utilizamos da classe "Thread" que permite esse recurso, muito utilizado em jogos e outros tipos de sistemas. Para isso, devemos em qualquer um dos dois modos de criação de Thread (implementando-se a interface "Runnable", ou herdando-se a classe "Thread"), existirá um objeto do tipo "Thread", que para que seja acionado o múltiplo processamento, devemos acionar o método "start" daquele objeto do tipo "Thread", que, por sua vez, acionará automaticamente o método "run()" da classe criada.

Pergunta 10

0,25 em 0,25 pontos

Levando-se em consideração a criação, lançamento e captura de exceções, que resultado será apresentado na Tela da Console quando este programa for executado (método "main()" da Classe Exemplo)?

```
-----------------------------------
public class ErroEspecificoException extends Exception {
 public ErroEspecificoException() {
  super("");
 }
}
-----------------------------------
public class RegraDeNegocio {
 public int status;
 public void verificaSituacao() throws Exception {
  System.out.print("H");
  if (status == 0) {
   throw new ErroEspecificoException();
  }
 }
}
-----------------------------------
public class Exemplo {
 public static void main(String[] args){
  System.out.print("A");
  RegraDeNegocio rn = new RegraDeNegocio();
  rn.status = 0;
  try {
   System.out.print("B");
   rn.verificaSituacao();
   System.out.print("C");
  } catch (ErroEspecificoException ex) {
   System.out.print("D");
  } catch (Exception ex) {
   System.out.print("E");
  } finally {
   System.out.print("F");
  }
  System.out.print("G");
 }
}
-----------------------------------
```

A saída desse programa, ao acionarmos o método "main" da classe "Exemplo", será:

Resposta Selecionada: a. ABHDFG.

Respostas:
a. ABHDFG.
b. ABCDEFGH.
c. ABHG.
d. ABFG.
e. ABDHEG.

Comentário da resposta: Resposta: A
Comentário: Percebe-se nas classes do programa do enunciado, que toda linha de comando de impressão é do tipo "print" (e não "println" que pula linha ao imprimir um texto na console), de forma que toda impressão estará logo ao lado da impressão anterior. Assim, ao acionarmos o método "main" da classe "Exemplo" (simplesmente "rodando a classe Exemplo") percebemos que ele já imprime inicialmente a letra "A". Em seguida é instanciado um objeto da classe "RegraDeNegocio" (objeto "rn"), do qual colocamos o valor 0(zero) para seu atributo "status". Assim, continuando o método "main", roda-se o bloco "try-catch" deste método, em que de início é impressa a letra "B" (em seguida da letra "A"). Roda-se com isso o método "verificaSituacao()" do objeto "rn" que já imprime a letra "H" (em seguida da letra "B"). Seguindo a execução do método, como o status é igual a 0(zero), então a exceção "ErroEspecificoException" será lançada, que de acordo com a estrutura "try-catch" do método "main", o bloco "catch" que trata essa exceção imprime a letra "D" (em seguida da letra "H"). Como o bloco "finally" sempre é rodado, independentemente de ter ocorrido ou não a exceção, a letra "F" será impressa logo em seguida da letra "D", e como não houve "travamento" da execução da JVM (devido ao tratamento da exceção), o sistema continua a executar os códigos existentes após a estrutura "try-catch", imprimindo no caso, por fim, a letra "G", terminando a execução do programa, resultando no anagrama "ABHDFG" .