ÁREA DO ALUNO

Fundamentos de Java 15

Avaliação

1) Marque as alternativas corretas que representam diferenças entre checked exceptions e unchecked exceptions.
*** Marque todas as alternativas que respondem o enunciado da questão**

- [x] Unchecked exceptions normalmente são utilizadas em situações onde não se espera que elas sejam tratadas. **(alternativa correta)**
- [x] Checked exceptions herdam de `Exception` direta ou indiretamente, mas nunca de `RuntimeException`. **(alternativa correta)**
- [ ] Checked exceptions não precisam ser declaradas na assinatura dos métodos que as lançam através do `throws`. **(alternativa incorreta)**
- [x] Checked exceptions precisam ser declaradas na assinatura dos métodos que as lançam através do `throws`. **(alternativa correta)**
- [ ] Unchecked exceptions herdam de `Exception` direta ou indiretamente, mas nunca de `RuntimeException`. **(alternativa incorreta)**

Correto

Resposta correta!
O tipo da exceção depende da sua hierarquia. Se ela herdar de `RuntimeException` direta ou indiretamente, será uma unchecked exception. Já se ela herdar de `Exception`, direta ou indiretamente, e não possuir `RuntimeException` na sua hierarquia, será uma checked exception. Normalmente as checked exceptions são utilizadas quando você espera que o código que fez a chamada trate a exceção de alguma forma. Já as unchecked exceptions são utilizadas quando o tratamento da exceção não é esperado.

---

2) Assumindo que todos os pacotes relativos às exceções estão importados, quais alternativas representam o que acontecerá se o código abaixo for compilado e executado?
*** Marque todas as alternativas que respondem o enunciado da questão**

```
01 public class Exemplo {
02   public static void read() throws Exception {
03     try {
04       throw new IOException();
05     } catch (FileNotFoundException e) {
06       System.out.println("catch 1");
```

```
07        return;
08      }
09    }
10
11    public static void main(String[] args) throws Exception {
12      try {
13        read();
14      } catch (FileNotFoundException f) {
15        System.out.println("catch 2");
16      } finally {
17        System.out.println("finally");
18      }
19    }
20  }
```

- [x] O código compila e executa, mas termina com uma exceção sendo lançada para a JVM. **(alternativa correta)**
- [ ] As palavras "finally" e "catch 2" serão impressas. **(alternativa incorreta)**
- [x] Dentre chamadas a `System.out.println()`, apenas a palavra `finally` será impressa. **(alternativa correta)**
- [ ] As palavras "finally", "catch 1" e "catch 2" serão impressas. **(alternativa incorreta)**
- [x] O código não compila. **(alternativa incorreta)**

Incorreto

Resposta incorreta!
Nenhum dos blocos `catch` do código será executado, uma vez que a exceção lançada é a `IOException`, que não é do tipo `FileNotFoundException` (apenas o inverso é verdadeiro). Logo, apenas a palavra "finally" que aparece na linha 17 será impressa.
Como o método `read()` lança uma exceção que não é tratada, esta exceção é propagada para a JVM (através da cláusula `throws` na linha 11), fazendo com que a JVM aborte a execução da aplicação.

---

3) Qual afirmativa é verdadeira com relação ao bloco `finally`?
*** Marque todas as alternativas que respondem o enunciado da questão**

- [x] Se um bloco `catch` existir junto com um bloco `finally`, o bloco `finally` deve ser sempre declarado depois dele. **(alternativa correta)**
- [ ] Blocos `finally` só são executados quando ocorre uma exceção dentro do bloco `try`. **(alternativa incorreta)**
- [x] O bloco `finally` correspondente ao bloco `try`, quando existir, sempre é executado, independente de uma exceção ter sido lançada por uma chamada a um método dentro do bloco `try` ou não. **(alternativa correta)**
- [ ] Blocos `finally` nunca são executados quando ocorre uma exceção dentro do bloco `try`. **(alternativa incorreta)**
- [x] Quando um bloco `finally` não estiver associado a um bloco `try`, ele será executado apenas quando exceções não forem lançadas dentro do método. **(alternativa incorreta)**

Incorreto

Resposta incorreta!

O bloco `finally` não é obrigatório, mas sempre que existir deve estar associado a um bloco `try`. É possível a existência do bloco `finally` sem blocos `catch`. O bloco `finally` sempre será executado quando uma exceção for lançada de dentro do bloco `try` ou até se o bloco `try` terminar normalmente. Se o bloco `catch` estiver presente, o bloco `finally` deve ser sempre declarado depois dele.

4) O que será impresso quando o código abaixo for executado?

```
01 public class Exemplo {
02   public static void main(String[] args) {
03     try {
04       throw new NullPointerException();
05     } catch (NullPointerException ne) {
06       System.out.print("1 ");
07     } catch (RuntimeException re) {
08       System.out.print("2 ");
09     } finally {
10       System.out.print("3 ");
11     }
12   }
13 }
```

- 3 1 **(alternativa incorreta)**
- 2 3 **(alternativa incorreta)**
- 3 **(alternativa incorreta)**
- 1 **(alternativa incorreta)**
- 1 2 3 **(alternativa incorreta)**
- 1 3 **(alternativa correta)**

Incorreto

Resposta incorreta!

O código da linha 4 desvia a execução para a linha 6, imprimindo o "1". Após o término do bloco `catch`, o bloco `finally` é executado, fazendo com que o "3" também seja impresso. No máximo um bloco `catch` do mesmo nível pode ser executado de cada vez. `NullPointerException` é uma exceção do tipo `RuntimeException`, mas a linha 6 é executada ao invés da linha 8 porque a ordem dos blocos `catch` influencia o fluxo da execução.

5) O que acontecerá se o código abaixo for compilado e executado?

*** Marque todas as alternativas que respondem o enunciado da questão**

```
01 public class Exemplo {
02   private static int processar(int i) {
03     try {
04       if (i < 5) {
05         return 5 / i;
06       } else {
07          return i * 5;
08       }
09     } catch (ArithmeticException e) {
10       System.out.println("ArithmeticException");
11     } finally {
12       System.out.println("Finally");
13     }
14     return 0;
15   }
16
```

```
17    public static void main(String[] args) {
18      System.out.println(processar(4));
19      System.out.println(processar(10));
20    }
21  }
```

- [ ] Quando a linha 18 executar, o programa imprime "Finally" e 104. **(alternativa incorreta)**
- [ ] A linha 14 não será executada. **(alternativa correta)**
- [ ] Quando a linha 19 executar, o programa imprime "Finally" e 110. **(alternativa incorreta)**
- [x] Quando a linha 18 executar, o programa imprime "Finally" e 1. **(alternativa correta)**
- [x] Quando a linha 19 executar, o programa imprime "Finally" e 50. **(alternativa correta)**

Incorreto

Resposta incorreta!
O bloco `finally` é sempre executado após a finalização do bloco `try` ou do bloco `catch`. Logo, a expressão "Finally" é sempre mostrada. A linha 14 não será executada neste código, pois ambas as chamadas ao método `processar()` atingem os comandos `return` das linhas 5 e 7.

Retornar para a página do curso (/course/home/id/1)