Exercícios com exceções – parte 2

Finally

Os blocos try/catch podem conter uma terceira cláusula chamada finally que indica o que deve ser feito após o término do bloco try ou de um catch qualquer.

Geralmente se coloca algo que é imprescindível de ser executado, caso o que você queira fazer tenha dado certo ou não. O mais comum é que o programador use o finally para liberar algum recurso, como um arquivo, uma conexão com o banco de dados, mesmo que algo tenha falhado no decorrer do código.

Veja o código abaixo:

```
try{
        //Bloco do try
} catch(IOException ex){
        //Bloco do catch 1
} catch(SQLException sqlex){
        //Bloco do catch 2
} finally{
        //Bloco do finally
}
```

O bloco finally será executado correndo tudo ok ou dando erro.

Exercícios com Exceções (Projeto fornecido pelo professor)

1) Na classe conta, adicione a classe SaldoInsuficienteException do exercício anterior e faça com que o método saca lance essa exceção.

2) Ainda na classe Conta, modifique o método deposita. Ele deve lançar uma IllegalArgumentException sempre que receber um valor inválido como argumento (um valor negativo por exemplo).

```
public void deposita(double quantidade) {
      if (quantidade < 0) {
            throw new IllegalArgumentException();
```

```
        }else{
                this.saldo += quantidade;
        }
}
```

3) Crie uma classe TestaDeposita com o método main. Crie uma ContaPoupanca e tente depositar valores inválidos:

```
public class TestaDeposita {
        public static void main(String[] args) {
                Conta cp = new ContaPoupanca();
                cp.deposita(-100);
        }
}
```

O que acontece? Uma IllegalArgumentException é lançada já que tentamos depositar um valor inválido. Adicione o try/catch para tratar o erro:

```
        public static void main(String[] args) {
                Conta cp = new ContaPoupanca();
                try {
                        cp.deposita(-100);
                } catch (IllegalArgumentException e) {
                        System.out.println("Você tentou depositar um valor inválido");
                }
        }
```

IMPORTANTE: Se sua classe ContaCorrente está reescrevendo o método saca (SaldoInsuficienteException) ou deposita (IllegalArgumentException) e não utiliza o super.deposita ou o super.saca, ela não lançará a exception no caso do valor inválido. Você pode resolver isso usando o super.deposita ou o super.saca.

4) Ao lançar uma IllegalArgumentException, passe via construtor uma mensagem a ser exibida. Lembre que a String recebida como parâmetro é acessível depois via método getMessage() herdado por todas as Exceptions.

```
        public void deposita(double quantidade) {
```

```
            if (quantidade < 0) {
                  throw new IllegalArgumentException("Você tentou
depositar um valor negativo.");
            }else{
                  this.saldo += quantidade;
            }
      }
```

5) Agora altere sua classe TestaDeposita para exibir a mensagem da exceção através da chamada do método getMessage():

```
      public static void main(String[] args) {
            Conta cp = new ContaPoupanca();
            try {
                  cp.deposita(-100);
            } catch (IllegalArgumentException e) {
                  System.out.println(e.getMessage());
            }
      }
```

6) Crie sua própria Exception, ValorInvalidoException. Para isso você precisa de uma classe com esse nome que estenda RunTimeException. O Eclipse vai sgerir que você serialize esta rotina. Faça isso! Mais adiante discutiremos melhor isso.

```
public class ValorInvalidoException extends RuntimeException {
      /**
       *
       */
      private static final long serialVersionUID = 1L;

      public ValorInvalidoException(String message) {
            super(message);
      }
}
```

IMPORTANTE: Nem sempre é interessante criarmos um novo tipo de exceção. Depende do caso. Nesse caso, por exemplo, bastava usar uma IllegalArgumentException. Na dúvida prefira as já existentes.

7) Mude o construtor de ValorInvalidoException para que ele receba como argumento o valor inválido que ele tentou passar. Quando estendemos uma classe, não herdamos seu construtor, mas podemos acessá-los através da palavra chave super de dentro de um de seus construtores. As exceções em Java possuem uma série de construtores úteis para poder injetar uma mensagem de erro. No exemplo acima, delegamos para o construtor da classe mãe. Vamos continuar fazendo isso neste exercício.

```
public class ValorInvalidoException extends RuntimeException {
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public ValorInvalidoException(double valor) {
		super("Valor inválido: "+valor);
	}
}
```

8) Declare a classe ValorInvalidoException como filha direta de Exception em vez de RuntimeException. Ela agora passa a ser checked. No que isso resulta?

Você vai precisar avisar que o seu método deposita() throws ValorInvalidoException, pois ela é uma checked Exception. Além disso, quem chama esse método vai precisar tomar uma decisão entre try/catch ou throws. Utilize-se do QuickFix do Eclipse novamente!

Depois retorne a exception para unchecked (filha de RuntimeException), pois iremos utilizá-la assim mais adiante.

9) **E o método transferePara?** Como esse método utiliza os métodos saca e deposita, é provável que vc tenha que adicionar um throws em sua assinatura em algum momento. Faça isso!

IMPORTANTE: Existe uma péssima prática de programação que é a de escrever o catch e o throws com Exception. Isso é muito genérico. Procure classificar as Exceptions para que quem receba saiba qual é o tipo de exceção com o qual está lidando.

Outro exercício com finally

1) Para simular um recurso como uma conexão, um arquivo ou qualquer outra coisa que tenhamos que utilizar e depois liberar da memória independentemente de se deparar

com uma exceção ou não, crie o projeto projeto-finally e a classe Recurso abaixo. Essa classe simula a obtenção, utilização e liberação de um recurso qualquer.

```
2  public class Recurso{
3      public Recurso() {
4          System.out.println("Obtendo um determinado recurso...");
5      }
6
7      public void utilizar() {
8          System.out.println("Utilizando um determinado recurso...");
9      }
10
11     public void liberar() {
12         System.out.println("Liberando/fechando um determinado recurso...");
13     }
14 }
```

2) Crie uma classe de testes para simular a utilização do recurso que acabamos de criar. Em sequência iremos obter um recurso, utiliza-lo e liberá-lo sem grandes problemas.

```
public class TestaRecurso {
    public static void main(String[] args) {
        Recurso recurso = new Recurso();
        recurso.utilizar();
        recurso.liberar();
    }
}
```

3) Para ter um problema real com o qual precisaremos lidar, crie a exceção RecursoComProblemasException que será uma checked exception (tratamento obrigatório).

```
2  public class RecursoComProblemasException extends Exception {
3      public RecursoComProblemasException() {
4          super();
5      }
6
7      public RecursoComProblemasException(String msg) {
8          super(msg);
9      }
10 }
```

4) Agora faremos com que o método utiliza lance a exceção de acordo com algum teste fictício. Como estamos lançando uma checked exception, o eclipse vai te obrigar a deixar o método mais legível, avisando na assinatura que ele está sujeito a lançar (throws) um RecursoComProblemasException.

```
2  public class Recurso{
3      public Recurso() {
4          System.out.println("Obtendo um determinado recurso...");
5      }
6
7      public void utilizar() throws RecursoComProblemasException {
8          System.out.println("Utilizando um determinado recurso...");
9          if(2+2==4)
10             throw new RecursoComProblemasException();
11     }
12
13     public void liberar() {
14         System.out.println("Liberando/fechando um determinado recurso...");
15     }
16 }
```

5) Agora que utilizar está sujeito a lançar uma exceção de tratamento obrigatório, na chamada do método o eclipse pede para o programador decidir se trata ou se deixa a exceção passar.

```
2  public class TestaRecurso {
3      public static void main(String[] args) {
4          Recurso recurso = new Recurso();
5          recurso.utilizar();
6          recurso.liberar();
7      }
8  }
```

6) Faça com que TestaRecurso trate a exceção apenas imprimindo o rastro da pilha. Lembrando que imprimir o rastro da pilha é muito útil enquanto estamos desenvolvendo um sistema. Quando um sistema está pronto, geralmente utilizamos mensagens que sejam de fácil compreensão para o usuário final. Perceba ainda que dando tudo certo (try) ou encontrando algum erro (catch), temos que liberar o recurso utilizado.

```
2  public class TestaRecurso {
3      public static void main(String[] args) {
4          Recurso recurso = new Recurso();
5          try {
6              recurso.utilizar();
7              recurso.liberar();
8          } catch (RecursoComProblemasException e) {
9              e.printStackTrace();
10             recurso.liberar();
11         }
12     }
13 }
```

7) Faça com que o método utilizar de Recurso lance a exceção já com uma mensagem de erro.

```
9          if(2+2==4)
10             throw new RecursoComProblemasException("Não foi possível utilizar o recurso.");
```

8) Faça com que essa mensagem de erro seja exibida no tratamento da exceção. Perceba que não é muito produtivo ter que liberar o recurso em dois lugares.

```
2  public class TestaRecurso {
3      public static void main(String[] args) {
4          Recurso recurso = new Recurso();
5          try {
6              recurso.utilizar();
7              recurso.liberar();
8          } catch (RecursoComProblemasException e) {
9              System.out.println(e.getMessage());
10             recurso.liberar();
11         }
12     }
13 }
```

9) Por fim, acrescente o bloco finally e chame o método liberar apenas nesse bloco. Afinal, o finally SEMPRE é executado! Perceba que o código ficou muito mais enxuto e legível. Execute TestaRecurso com a exceção sendo lançada (if(2+2==4)) e depois fazendo com que ela não seja lançada (if(2+2==5)). De um jeito ou de outro o finally será executado e o recurso será liberado!

```
2  public class TestaRecurso {
3      public static void main(String[] args) {
4          Recurso recurso = new Recurso();
5          try {
6              recurso.utilizar();
7          } catch (RecursoComProblemasException e) {
8              System.out.println(e.getMessage());
9          }finally {
10             recurso.liberar();
11         }
12     }
13 }
```