Lançar Exceções

Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

Temas

1 Proteger Integridade

2 RuntimeException

3 Lançar uma Exceção

1 | Proteger Integridade

Proteja a integridade de uma classe

Quando criamos uma classe, estamos tentando representar algo que tem um determinado comportamento.

Os valores que ficam armazenados em seus atributos podem ter que respeitar uma faixa de valores, nesse caso, temos que proteger a integridade da classe, vamos ver um exemplo:

Data
- int day - int month - int year
+ Data(int d, int m, int y)

Usamos os nomes dos atributos em inglês para evitar o uso do "ê".

Proteja a integridade de uma classe

Uma data é algo bem conhecido de todos, mas se a classe representa algo não tão comum, quem tem que usar a classe não precisa saber com que faixa de valores trabalhar.

Para efeitos práticos, vamos estabelecer que os dias podem ser entre 1 e 31 independentemente do mês e os meses entre 1 e 12.

Data
- int day - int month - int year
+ Data(int d, int m, int y)

2 | RuntimeException

RuntimeException

```
class Data{
      private int day;
      private int month;
      private int year;

      public Data(int d, int m, int y){
            day=d;
            month=m;
            year=y;
      }
}
```

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

      Data data= new Data(100,-100,1000);
}
```

Ao usar a classe Data, para criar uma nova data, você precisa passar 3 valores inteiros.

Isso está sendo cumprido no exemplo, mas quem usa uma classe não entende necessariamente o comportamento dessa classe.

A classe deve se proteger e evitar obter valores que não estejam na faixa esperada.

RuntimeException

```
class Data{
      private int day;
      private int month;
      private int year;

      public Data(int d, int m, int y){
            if (d<1||d>31||m<1||m>12)
                  throw new RuntimeException("OS valores não são válidos");
            day=d;
            month=m;
            year=y;
      }
}
```

Com throw, lançamos uma exceção em execução; para isso, criamos a nova exceção.

RuntimeException

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

     Data data= new Data(100,-100,1000);
}
```

```
Exception in thread "main" java.lang.RuntimeException: Os valores não são válidos
```

Agora, se tentarmos criar com valores inválidos, será gerada uma exceção.
As exceções do tipo RuntimeException não precisam ser protegidas com blocos try / catch.

3 | Lançar uma Exceção

Lançar uma exceção por código

```
class Data{
      private int day;
      private int month;
      private int year;

      public Data(int d, int m, int y){
            day=d;
            month=m;
            year=y;
      }
}
```

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

      Data data= new Data(100,-100,1000);
}
```

Vamos agora ver outra maneira de proteger esse código. Agora vamos fazer de forma que quem usa o método seja obrigado a **protegê-lo** com try / catch.

Lançar uma Exceção

Da mesma forma que antes, lançamos a exceção com **throw**, mas neste caso é do tipo **Exception**, uma vez que não há exceção definida que diga fora do intervalo esperado. Acrescenta-se que devemos avisar que o método pode gerar uma exceção, acrescentamos, após a assinatura **throws Exception.** É um método que pode ser **throweable.**

```
class Data{
      private int day;
      private int month;
      private int year;

      public Data(int d, int m, int y) throws Exception{
            if (d<1||d>31||m<1||m>12)
                  throw new Exception("Os valores não são válidos");
            day=d;
            month=m;
            year=y;
      }}
```

RuntimeException

Por ser um método throweable, ele nos obriga a protegê-lo com try / catch.

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

     try{

          Data data= new Data(100,-100,1000);
     catch (Exception e) {
          System.err.println("Não são valores válidos para uma data");
     }

}
```


DigitalHouse>
Coding School