Sobrecarga e `toString()`

POO

Prof. Marcio Delamaro

Sobrecarga

- Assim como acontece com os construtores, é possível que uma classe tenha vários métodos com o mesmo nome
- A assinatura deve ser diferente
- Na verdade, os parâmetros precisam ser diferentes
- O tipo pode ser igual ou diferente

Exemplo de sobrecarga

- Vamos supor que queremos, na nossa classe de números aleatórios, três métodos que retornem um número aleatório inteiro:
- um que retorna o valor entre 0 e n
- um que retorna o valor entre n_1 e n_2
- um que retorne um valor inteiro qualquer, ou seja até o limite máximo.

getIntRand

```
int getIntRand(int max)

int getIntRand(int min, int max)

int getIntRand()
```

Sobrecarga 1

```
public int getIntRand(int max)
{
    double d = getRand() * max;
    return (int) d;
}
```

Sobrecarga 2

```
public int getIntRand(int min, int max)
{
   return min + getIntRand(max - min);
}
```

Sobrecarga 3

```
public int getIntRand()
{
   return getIntRand(Integer.MAX_VALUE);
}
```

Aproveitando....

- Na versão 2, existe algum caso que não dá pra computar o número?

Aproveitando....

- Na versão 2, existe algum caso que não dá pra computar o número?
- Valor de `max` deve ser maior que valor de `min`
- O que fazer se não for?

Aproveitando....

- Na versão 2, existe algum caso que não dá pra computar o número?
- Valor de `max` deve ser maior que valor de `min`
- O que fazer se não for?
- É caso que podemos sinalizar com uma exceção.

Aproveitando... exceção

```
public int getIntRand(int min, int max)  {
   if ( max <= min )
       throw new
        IllegalArgumentException("Parâmetros
              inválidos");
   return min + getIntRand(max - min);
}
```

Aproveitando... exceção

```
public int getIntRand(int min, int max)throws
IllegalArgumentException
   {
      if ( max <= min )
          throw new
           IllegalArgumentException("Parâmetros
                 inválidos");
      return min + getIntRand(max - min);
   }
```

Sobrecarga

- É bastante comum o uso de sobrecarga
- Dá mais flexibilidade ao código
- Permite que uma funcionalidade seja invocada de diversas maneiras diferentes

método `println`

- `println(“Isso é uma mensagem qualquer”);`

método `println`

- `println("Isso é uma mensagem qualquer");`
- `println("O valor é: " + x);`
  - ????

método `println`

- `println("Isso é uma mensagem qualquer");`
- `println("O valor é: " + x);`
  - ????
- `println("O valor de X é: " + x +`
  `" e o valor de Y é: " + y);`

método `println`

- `println("Isso é uma mensagem qualquer");`
- `println("O valor é: " + x);`
  - Concatenando String com um número
- `String s = "O valor é" + x;`
  `println(s);`

Método `println`

- `println("Isso é uma mensagem qualquer");`
- `println("O valor é: " + x);`
  - Concatenando String com um número
- ```
  String s = "O valor é" + x;
  println(s);
  ```
- ```
  Random rd = new Random();
  String s = "Instanciei um objeto: " + rd;
  println(s);
  ```

Método `println`

- `println("Isso é uma mensagem qualquer");`
- `println("O valor é: " + x);`
  - Concatenando String com um número
- ```
  String s = "O valor é" + x;
  println(s);
  ```
- ```
  Random rd = new Random();
  String s = "Instanciei um objeto: " + rd;
  println(s);
  ```

Instanciei um objeto: Random@10bedb4

Método `toString()`

- Todo objeto instanciado tem esse método
- O comportamento padrão, varia de acordo com o objeto instanciado
- Se você definir uma classe, você pode alterar esse comportamento

Alterando método `toString()`

```
@Override
    public String toString()
    {
       return "Esse é meu objeto Random";
    }
```

Alterando método `toString()`

```
@Override
    public String toString()
    {
       return xi + "";
    }
```

Alterando método `toString()`

```
@Override
    public String toString()
    {
       return xi + "";
    }
```

Instanciei um objeto: 894756301