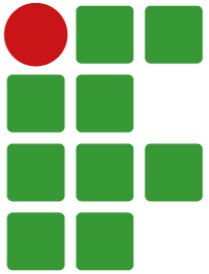

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Linguagem de Programação 2

Prof. Dr. Domingos Bernardo Gomes Santos

Ementa

- Aprofundando conhecimentos sobre o método ***equals*** para comparação entre objetos;
- Verificando a implementação e funcionamento do método ***hashcode;***
- Como implementar o método ***clone*** para clonagem de Objetos;
- Diferenças entre primitivas e wrappers;
- Classes imutáveis;
- Exercícios.

Comparando Objetos

- O método ***equals*** é utilizado para comparar objetos;
- Observar os detalhes da implementação do método ***equals*** na classe Carro;
- Observar o funcionamento do método ***equals*** na classe TesteEquals;
- Assinatura do método ***equals*** na classe **Object**.

`boolean`

`equals(Object obj)`
Indicates whether some other object is "equal to" this one.

Comparando Objetos

- Simetria: para duas instâncias, ***a*** e ***b***, ***a.equals(b)*** se e somente se ***b.equals(a)***;
- Reflexividade: para todas referências não nulas, ***a.equals(a)***;
- Transitividade: se ***a.equals(b)*** e ***b.equals(c)***, então ***a.equals(c);***
- Consistência com ***hashcode()***: dois objetos iguais precisam ter o mesmo ***hashcode()***;

Implementação do Método hashcode

- Observar a implementação do método ***hashcode*** nas classe **Usuario** e **Aviao**;
- Observar a funcionalidade do ***hashcode*** nas classe **TesteHashcode** e **TesteHashcode2**;
- Assinatura do método ***hashcode*** na classe **Object**.

```
int                                    hashCode()
                                       Returns a hash code value for the object.
```

Clonagem de Objetos

- O método ***clone*** é utilizado para clonagem de objetos;
- Observar a implementação do método clone nas classes **Carro** e **Funcionário**;
- Observar a funcionalidade do método ***clone*** na classe **TesteClone**.

Clonagem de Objetos

- Implementar a interface ***Cloneable***;
- Incluir na sobre escrita do método clone() suporte a ***exceptions***.

```
public class Carro implements Cloneable {
```

```
public Carro clone() throws CloneNotSupportedException {
```

Primitivas

		Valores possíveis				
Tipos	***Primitivo***	***Menor***	***Maior***	***Valor Padrão***	***Tamanho***	***Exemplo***
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;

	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d;
						ou
						double ex6 = 10.20;

Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;

Wrappers

- Uma classe empacotadora, em inglês ***wrapper class,*** na linguagem de programação Java são oito classes fornecidas no pacote **java.lang** para fornecer métodos de objeto para oito tipos primitivos:
  - Boolean;
  - Byte;
  - Character;
  - Double;
  - Float;
  - Integer;
  - Long;
  - Short.

Objetos Imutáveis

- Objetos **imutáveis** são objetos que uma vez instanciados não podem ter seus estados internos modificados;
- A Java API apresenta diversas classes que são **imutáveis**, como a classe **String** e as classes ***Wrappers*** (**Integer**, **Double**, etc);
- A utilização deste recurso simplifica a codificação e a execução dos aplicativos Java.

Objetos Imutáveis

- São de simples construção, teste e utilização;
- São automaticamente "**thread-safe**" e não possuem problemas de sincronização;
- Possibilitam seu uso em caches de objetos por utilizar **hashcode** (não alterar o valor do **hashcode**);
- Não necessitam serem copiados de forma defensiva quando utilizados como atributos em outros objetos (**String**, **Integer**, etc);
- São perfeitos para utilizar como índices em **Maps** e elementos de um **Set** por não mudar de estado enquanto estão atribuídos a uma coleção;

Exercícios