

POO - PROGRAMAÇÃO ORIENTADA A OBJETOS

Object a Superclasse de toda Classe JAVA

Rodrigo R Silva

Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense
Campus Bagé

Nesta Aula Veremos...

1 Introdução

2 Object

Introdução

Contextualizando...

- Java define uma classe que forma a base de todos os modelos de classes da linguagem. Desde classes da própria API Java, como classes definidas por programadores.
- Esta classe é denominada **Object** e é uma classe implícita. Portanto, não é necessário estender Object quando definimos uma nova classe.

Object

Object como generalização

- Portanto, toda classe em Java é um subtipo de Object.
- Dessa forma, qualquer objeto de qualquer classe pode ser referenciado como Object.
- Vejamos um exemplo:
`ContaEspecial contaEspecial = new ContaEspecial();`
`Object objetoGenerico = contaEspecial;`

Coerção (Cast) de objetos Java

- Quando um objeto de uma determinada classe é referenciado como Object, devemos realizar uma **ação de coerção** para referenciar o objeto em sua classe original.
- Ex:

```
ContaEspecial especial1 = new ContaEspecial();  
Object obj = especial1;  
...  
ContaEspecial especial2 = (ContaEspecial) obj;
```

Coerção (Cast) em primitivos

- Já em tipos de dados primitivos a coerção realiza uma **conversão de valores**.
- Ex:
`long valorQualquer = 9483762182727L;`
`int outroValor = (int) valorQualquer;`

Passando Object em métodos

- Podemos passar um objeto de uma classe qualquer para um método na sua forma mais genérica. Ou seja, na forma de um Object.

```
public class ClasseExecutavel {  
  
    public static void main(String[] args) {  
  
        ContaEspecial especial = new ContaEspecial();  
        especial.depositar(1500f);  
  
        ContaComum comum = new ContaComum();  
        comum.depositar(1000f);  
  
        sacar(especial, 100.00f);  
        sacar(comum, 200.00f);  
  
        System.out.println("Saldo de conta especial R$ "+especial.getSaldo());  
        System.out.println("Saldo de conta comum R$ "+comum.getSaldo());  
  
    }  
  
    public static void sacar(Object conta, float valor) {  
        ((Conta) conta).sacar(valor);  
    }  
}
```

Passando Object em métodos

- O que acontece se passarmos um tipo qualquer diferente de Conta?

```
public class ClasseExecutavel {  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa();  
        ContaEspecial especial = new ContaEspecial();  
        especial.depositar(1500f);  
  
        ContaComum comum = new ContaComum();  
        comum.depositar(1000f);  
  
        sacar(especial, 100.00f);  
        sacar(pessoa, 200.00f);  
  
        System.out.println("Saldo de conta especial R$ "+especial.getSaldo());  
        System.out.println("Saldo de conta comum R$ "+comum.getSaldo());  
    }  
  
    public static void sacar(Object conta, float valor) {  
        ((Conta) conta).sacar(valor);  
    }  
}
```

instanceof

- Podemos descobrir o tipo de um objeto em tempo de execução.

```
public class ClasseExecutavel {  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa();  
        ContaEspecial especial = new ContaEspecial();  
        especial.depositar(1500f);  
  
        ContaComum comum = new ContaComum();  
        comum.depositar(1000f);  
  
        sacar(especial, 100.00f);  
        sacar(pessoa, 200.00f);  
  
        System.out.println("Saldo de conta especial R$ "+especial.getSaldo());  
        System.out.println("Saldo de conta comum R$ "+comum.getSaldo());  
    }  
  
    public static void sacar(Object conta, float valor) {  
        if(conta instanceof Conta) {  
            ((Conta) conta).sacar(valor);  
            System.out.println(conta instanceof Conta);  
        }  
    }  
}
```

Métodos da classe Object

- A classe object define alguns métodos. Podemos reescrever seus comportamentos em nossas classes.
- Alguns destes métodos são úteis e podemos implementar algumas funcionalidades em nossas classes. São eles:
- `toString();`
- `equals();`
- `hashCode();`

Referenciando toString()

- Quando nossas aplicações referenciam a instância de uma classe, será chamado o método toString().
- Isso pode ser utilizado em instruções que mostram um objeto na tela.
- Ex: vamos mostrar na tela o objeto do tipo Pessoa.

```
Pessoa correntista = new Pessoa("Ciclano",33,999887766,"ciclano@gmail.com");  
System.out.println(correntista);
```

Saída padrão de toString()

- Se uma classe não reescrever o comportamento do método toString(), será implementado o comportamento padrão definido na classe Object.
- Esse comportamento mostra o nome da classe, o símbolo @ e um número hexadecimal.
- Ex:
`classes.java.Pessoa@7e774085`

O método toString()

- O método toString() retorna uma representação do objeto na forma textual.
- Geralmente, o valor de um ou mais atributos definidos na classe.
- Ex: vamos reescrever o comportamento de toString() na classe Pessoa.

```
@Override  
public String toString() {  
    return "Correntista: nome = " + nome + ", celular = " + celular;  
}
```

O método equals()

- Este método deve ser utilizado para implementarmos em nossas classes um comportamento de comparação.
- Tal comportamento visa verificar se dois objetos da mesma classe são iguais.
- Isso é muito útil nas nossas aplicações, pois podemos controlar se o usuário está tentando criar dois objetos iguais.
- O critério de igualdade é definido pelo programador. Geralmente, pela combinação dos valores de atributos.

Exemplo de equals()

- Vejamos como implementar o comportamento do método equals() na classe Pessoa.
- Vamos comparar dois objetos do tipo Pessoa através do atributo número do CPF. Visto que duas pessoa não podem possuir um mesmo número.

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Pessoa other = (Pessoa) obj;
    return celular == other.celular && cpf == other.cpf && Objects.equals(email, other.email)
        && idade == other.idade && Objects.equals(nome, other.nome);
}
```

Comparando dois objetos

- Veja a comparação de dois objetos.

```
Pessoa correntista = new Pessoa("Ciclano",22233344455l,33,999887766,"ciclano@gmail.com");
Pessoa correntista1 = new Pessoa("Beltrano",22233344455l,22,999112233,"beltrano@gmail.com");

if(correntista.equals(correntista1)) {
    System.out.println("Pessoas iguais =(");
}else {
    System.out.println("Pessoas diferentes");
}

System.out.println("-----");
System.out.println(correntista);
System.out.println("-----");
System.out.println(correntista1);
```

O método hashCode()

- O **hashCode** é uma ferramenta da JVM usada para montar a tabela de hash de modo correto.
- Tabela Hash [também conhecida como Tabela de Dispersão ou Tabela de Espalhamento] é uma tabela onde as informações são armazenadas conforme um **numero hash** calculado com base nas propriedades da informação. Isso permite que seja muito rápido recuperar uma informação na tabela.

Tempo de busca

- Imagine uma tabela com as informações de todos os pacientes de um hospital. Se fosse buscar um paciente em especial iria demorar um tempo ($O(n)$ numa busca linear ou $O(\log N)$ para buscas binárias) o que pode ser extremamente ruim em uma situação real onde existe um volume de dados gigantescos. Usando uma tabela hash a busca reduz seu tempo de busca ($O(1)$) para qualquer situação, bastando apenas o cálculo do valor hash na entrada e na saída da informação.

Exemplo de implementação do hascode

```
package classe.executavel;  
  
import classes.java.Telefone;  
  
public class ClasseExecutavelTelefone {  
  
    public static void main(String[] args) {  
  
        Telefone tel1 = new Telefone();  
        tel1.setMarca("Motorola");  
        tel1.setModelo("Razor");  
        tel1.setImei(12345);  
  
        Telefone tel2 = new Telefone();  
        tel2.setMarca("Motorola");  
        tel2.setModelo("Razor");  
        tel2.setImei(123456);  
  
        System.out.println(tel1);  
        System.out.println(tel1.hashCode());  
        System.out.println(tel2);  
        System.out.println(tel2.hashCode());  
  
        System.out.println(tel1.equals(tel2));  
    }  
}
```

```
<terminated> ClasseExecutavelTe  
classes.java.Telefone@3058  
12376  
classes.java.Telefone@1e25f  
123487  
false
```

**hash gerado conforme
sobrescrita do
método.**

OBRIGADO!