02

Use super e this para acessar objetos e construtores

Um construtor pode ser sobrecarregado assim como os métodos, e pode ter qualquer modificador de visibilidade.

O ponto mais importante sobre os construtores é que, para construir um objeto de uma classe filha, obrigatoriamente, precisamos chamar um construtor da classe mãe antes. Sempre, em todos os casos. Para chamar o construtor da mãe, usamos a chamada ao `super` (passando ou não argumentos):

```
class Mae {
    public Mae(String msg) {
        System.out.println(msg);
    }
}

class Filha extends Mae {
    public Filha(String nome) {
        super("construindo parte mae");
        System.out.println("construindo parte filha");
    }
}
```

COPIAR CÓDIGO

Mas, na maioria dos casos, não chamamos o construtor da mãe explicitamente. Se nenhum construtor da mãe foi escolhido através da palavra `super(...)`, o compilador coloca **automaticamente** `super();` no começo do nosso construtor, sem nem olhar para a classe mãe.

```
class Mae {
    public Mae() {
        System.out.println("construindo parte mae");
    }
}

class Filha extends Mae {
    public Filha(String nome) {
        // super() esta implícito!!!
        System.out.println("construindo parte filha");
    }
}
```

COPIAR CÓDIGO

Considerando agora o código:

```
public class X{
    public static void main(String [] args){
        Filha filha = new Filha("Teste");
    }
}
```

COPIAR CÓDIGO

Vai primeiro imprimir "Construindo parte mae" e só depois "Construindo parte filha".

Uma outra possibilidade, no caso de termos mais de um construtor, é chamarmos outro construtor da própria classe através do `this()` :

```
class Mae {
    public Mae() {
        System.out.println("construindo parte mae");
    }
}

class Filha extends Mae {
    public Filha() {
        // super() implicito!
        System.out.println("construindo filha parte 1");
    }

    public Filha(String nome) {
        this();
        System.out.println("construindo filha parte 2");
    }
}

public class X{
    public static void main(String [] args){
        Filha filha = new Filha("Teste");
    }
}
```

COPIAR CÓDIGO

Agora vai produzir "Construindo parte mae", "Construindo parte filha parte 1" e "Construindo parte filha parte 2".

Atenção, a chamada do construtor com `super` ou `this` só pode aparecer como primeira instrução do construtor. Portanto, só podemos fazer uma chamada desses tipos.

```
class Filha extends Object {
    public Filha() {
        // super() implicito!
    }

    public Filha(String nome) {
        this();
    }

    public Filha(int idade) {
        super();
        this(); // não compila, ou um ou outro!
    }
    public Filha(long valor) {
        this();
        this(); // não compila, só uma vez!
    }

    public Filha(char caracter) {
        super();
        super(); // não compila, só uma vez!
    }
}
```

COPIAR CÓDIGO

this e variáveis membro

Por vezes, temos variáveis membro com o mesmo nome de variáveis locais. O acesso sempre será a variável local, exceto quando colocamos o `this`, que indica que a variável membro será acessada. O código a seguir imprimirá 3 e depois 5:

```
class Teste {
    int i = 5;
    void roda(int i) {
        System.out.println(i);
        System.out.println(this.i);
    }
    public static void main() {
        new Teste().roda(3);
    }
}
```

COPIAR CÓDIGO

No acesso a variáveis membro com o `this` podem parecer que serão acessados somente valores da classe atual, mas buscam também nas classes da qual ela herda:

```
class A{
    int i = 5;
}
class Teste extends A{
    void roda(int i) {
        System.out.println(this.i); // imprime 5
    }
    public static void main(String [] args) {
        new Teste().roda(3);
    }
}
```

COPIAR CÓDIGO

Tentar acessar uma variável local com `this` não compila:

```
class Teste {
    void roda(int i) {
        System.out.println(this.i); // não há variável membro
i
    }
    public static void main(String [] args) {
        new Teste().roda(3);
    }
}
```

COPIAR CÓDIGO

Como mostramos, caso a variável seja escondida por uma variável com mesmo nome em uma classe filha, podemos diferenciar o acesso à variável membro da classe filha ou da pai, explicitando `this` ou `super` :

```
class A{
    int i = 5;
}
class Teste extends A{
    int i = 10;
    void roda(int i) {
        System.out.println(i); // imprime 3
        System.out.println(this.i); // imprime 10
        System.out.println(super.i); // imprime 5
    }
    public static void main(String [] args) {
        new Teste().roda(3);
    }
}
```

COPIAR CÓDIGO

O `this` é em geral opcional para acessar um método do nosso objeto atual (se ele não foi redefinido, da classe mãe):

```
class A{
    int i() { return 5; }
}
class Teste extends A{
    void roda() {
        System.out.println(this.i()); // imprime 5
    }
    public static void main(String [] args) {
        new Teste().roda();
    }
}
class Teste2 {
    int i() { return 5; }
    void roda() {
        System.out.println(this.i()); // imprime 5
    }
    public static void main(String [] args) {
        new Teste().roda();
    }
}
```

COPIAR CÓDIGO

this e super em variável membro

E o que acontece quando uma variável membro tem o mesmo nome que a definida na classe que herdamos? Se não definirmos o acesso através de `this` nem `super`, o acesso é à variável da classe filha. Se usarmos `this` é à classe filha novamente e se usarmos `super` é à classe pai:

```
class Veiculo {
    double velocidade = 30;
}
class Carro extends Veiculo {
    double velocidade = 50;
    void imprime() {
        System.out.println(velocidade); // 50
        System.out.println(this.velocidade); // 50
        System.out.println(super.velocidade); // 30
    }
}
class Teste {
    public static void main(String[] args) {
        Carro c = new Carro();
        c.imprime();
    }
}
```

COPIAR CÓDIGO

Lembre-se que o binding de uma variável ao tipo é feito em compilação, portanto se tentarmos acessar a variável velocidade fora do `Carro` através de uma referência a `Carro` , o valor alterado é o da variável `Carro.velocidade` :

```
class Veiculo {
    double velocidade = 30;
}
class Carro extends Veiculo {
    double velocidade = 50;
    void imprime() {
        System.out.println(velocidade); // 1000
        System.out.println(this.velocidade); // 1000
```

```
        System.out.println(super.velocidade); // 30
    }
}
class Teste {
    public static void main(String[] args) {
        Carro c = new Carro();
        c.velocidade = 1000;
        c.imprime();
    }
}
```

COPIAR CÓDIGO

E se fizermos o mesmo através de uma referência a `Veiculo`, alteramos a velocidade do `Veiculo`:

```
class Veiculo {
    double velocidade = 30;
}
class Carro extends Veiculo {
    double velocidade = 50;
    void imprime() {
        System.out.println(velocidade); // 50
        System.out.println(this.velocidade); // 50
        System.out.println(super.velocidade); // 1000
    }
}
class Teste {
    public static void main(String[] args) {
        Carro c = new Carro();
        ((Veiculo) c).velocidade = 1000;
        c.imprime();
    }
}
```

COPIAR CÓDIGO

Estático não tem this nem super

Contextos estáticos não possuem nem `this` nem `super`, uma vez que o código não é executado dentro de um objeto:

```
class A{
    int i = 5;
}
class Teste extends A{
    int i = 10;
    public static void main(String [] args) {
        this.i = 5; // this? não compila. código estático
        super.i = 10; // super? não compila. código estático
    }
}
```

COPIAR CÓDIGO

Por fim, uma última restrição: interfaces não podem ter métodos estáticos, não compila (métodos `default` não são cobrados nesta prova).