Programação Orientada a Objetos. SANTOS, Rafael (PLT) e
Guia de Estudo SCJP. SIERRA, Kathy e BATES, Bert

Capítulo 9 – Classes Abstratas e Interfaces

Introdução

- Há situações em que **não** deve haver **instâncias** de determinadas classes
- Há modelos **incompletos** que têm finalidade de servir de **base** para criação de outros modelos
- Às vezes, precisamos definir **o quê** deve ser feito, mas não **como** deve ser feito
- Exemplos:
  - Pessoa e descendentes
  - Veiculo e descendentes

Introdução

- Como visto no capítulo anterior, todos os campos (variáveis) e métodos que **podem** ser acessados a partir de uma variável de referência **dependem do tipo dessa variável de referência e não do objeto referenciado**, o que, às vezes, exige a criação de um método para que as **sobreposições possam ser acionadas**, por meio da referência mais genérica

Método abstrato

- Possui **nome** (identificador), modificadores (***abstract*** obrigatório em classes), **tipo** de retorno, lista de **parâmetros**
- **Sem corpo** (termina com **ponto-e-vírgula**)
- Possibilita a chamada a métodos da subclasse a partir de uma referência à superclasse
- **Deve** estar em uma **classe abstrata** ou em uma **interface**
- Uma classe **não** abstrata deve implementar

Classes abstratas

- **Não** pode ser instanciada
- Possui **construtor**
- Um método pode ser **abstrato** ou **estático**, mas não simultaneamente ambos
  - O método abstrato ainda será implementado
  - O método estático já está pronto para ser usado a partir da própria classe

Classes abstratas

```
public abstract class ClasseAbstrata {

    public abstract int metodoSoma();

    public int metodoSomaConcreto() {
        return 13;
    }

    public static int metodoSomaConcretoEstatico()
  {
        return 15;
    }

}
```

Classes abstratas

- Ver classes:
  - RoboAbstrato (Cap. 9)
  - RoboSimples (Cap. 9)
  - RoboABateria (Cap. 9)
  - DemoRobos (Cap. 9)

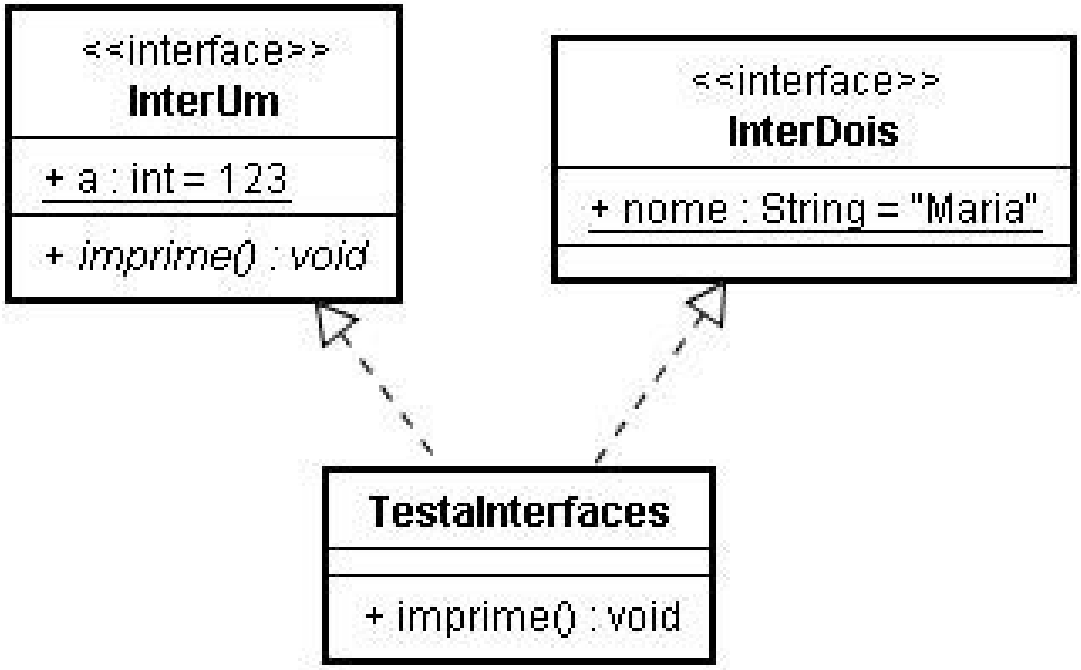
Interfaces

- Pode ser vista uma "**classe abstrata pura**", pois **todos** os métodos são **abstratos**
- **Não** podem ser instanciadas
- Todos os membros são **públicos**
- Todas as variáveis são **estáticas** (de classe) e **finais** (constantes)
- Não são estendidas, mas **implementadas** (palavra-chave **implements**)

Interfaces

- Uma **classe** pode **implementar** várias **interfaces** (mecanismos simplificado de herança múltipla)
- Uma **interface** pode **estender** várias **interfaces**

Interfaces


<<interface>>
InterUm
+ a : int = 123
+ imprime() : void
<<interface>>
InterDois
+ nome : String = "Maria"
TestaInterfaces
+ imprime() : void

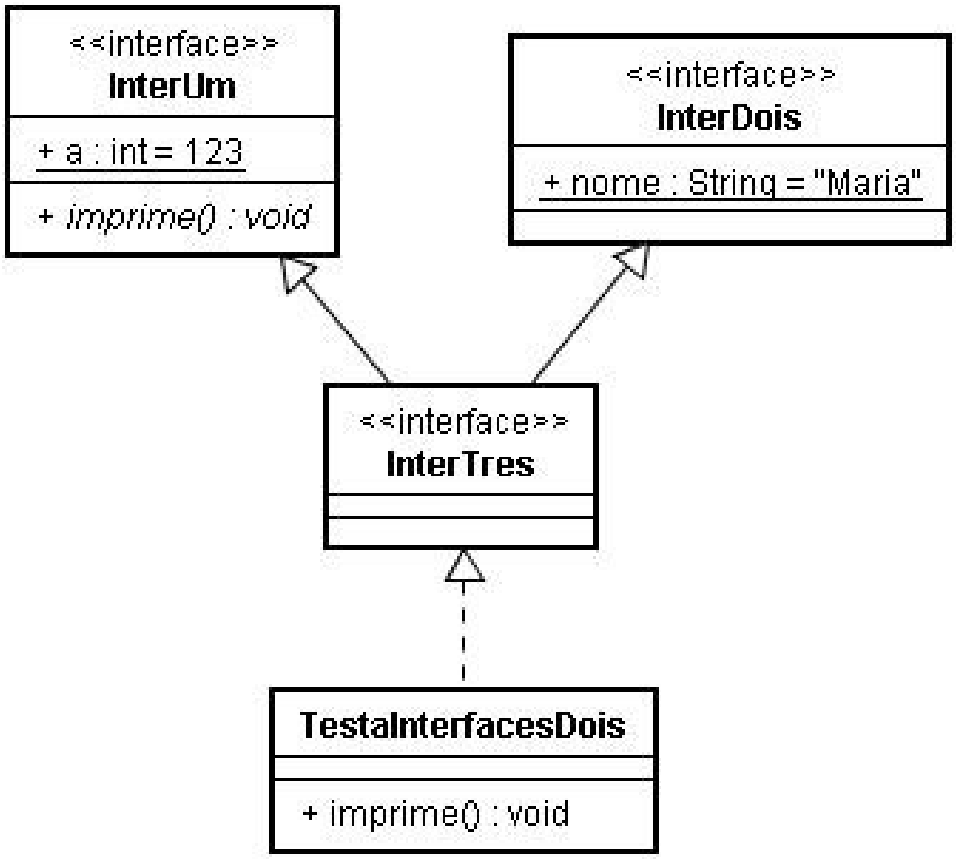
Interfaces

```
interface InterUm {
    int a = 123;
    void imprime();
}

interface InterDois {
    String nome = "Maria";
}

class TestaInterface implements InterUm, InterDois {
    public void imprime() {
        System.out.println(“A : " + a);
        System.out.println(“Nome: " + nome);
    }
}
```

Interfaces


<<interface>>
InterUm
+ a : int = 123
+ imprime() : void
<<interface>>
InterDois
+ nome : String = "Maria"
<<interface>>
InterTres
TestaInterfacesDois
+ imprime() : void

Interfaces

```
interface InterUm {
    int a = 123;
    void imprime();
}

interface InterDois {
    String nome = "Maria";
}

interface InterTres extends InterUm, InterDois {}

class TestaInterface implements InterTres {
    public void imprime() {
        System.out.println("Imprime a: " + a);
        System.out.println("Imprime nome: " + nome);
    }
}
```

Interfaces

- Ver classes:
  - ObjetoGeometrico (Cap. 9)
  - Circulo (Cap. 9)
  - Retangulo (Cap. 9)
  - DemoObjetosGeometricos (Cap. 9)

Exercício para Fixação

1. Crie uma interface pública de nome Imprimivel, em um pacote a sua escolha. Defina duas constantes de classe: IMPRIME_NOME_DA_EMPRESA com valor true e NAO_IMPRIME_NOME_DA_EMPRESA com valor false. Defina também um método público e abstrato que não retorne valor chamado imprime e que receba um parâmetro do tipo boolean.
2. Crie uma classe abstrata de nome Modelo, no mesmo pacote. Essa classe deve definir uma variável de instância nomeDaEmpresa, privada com métodos get e set públicos, um método toString abstrato e um construtor que receba como parâmetro o nome da empresa e inicialize a variável de instância equivalente.
3. Crie uma classe LinguagemDeProgramacao que estenda Modelo e implemente Imprimivel. Essa classe deve definir uma variável de instância nomeDaLinguagem privada e seus métodos get e set públicos. Deve definir também um construtor que acione o construtor da superclasse e inicialize a variável de instância criada na classe. Por último, deve ser implementado o método imprime que condicionará a impressão do nome da empresa ao valor do argumento recebido com parâmetro.
4. Crie uma classe TestaAPI que instancia objetos da classe LinguagemDeProgramacao, usando diferentes tipos de referência, atribui valores a suas variáveis de instância e imprime os objetos, usando o método imprime.

Polimorfismo e interfaces

- Como qualquer superclasse, uma interface pode ser um **supertipo** de (i.e., ter sido implementada por) várias classes, permitindo, assim, o tratamento **polimórfico** a partir de uma **referência do tipo da interface**
- Ver classes:
  - DemoObjetosGeometricosEPolimorfismo (Cap. 9)

Herança múltipla usando interfaces

- Ver classes:
  - Escalavel (Cap. 9)
  - CirculoEscalavel (Cap. 9)
  - DemoCirculoEscalavel (Cap. 9)
  - ItemBiblioteca (Cap. 9)
  - Livro (Cap. 9)
  - LivroDeBiblioteca (Cap. 9)
  - DemoLivroDeBiblioteca (Cap. 9)

Conflitos de herança múltipla

- Ver:
  - ItemRaroDeBiblioteca (Cap. 9)
  - Mapa (Cap.9)
  - MapaDeBiblioteca (Cap. 9)

Exercícios sugeridos para sala

- Analise detalhadamente os exemplos do capítulo, introduzindo modificações para que sejam completamente entendidos
- Todos os exercícios do capítulo devem ser resolvidos e eventuais dúvidas trazidas para debate
- Quando necessário, resolver exercícios de capítulos anteriores que são solicitados como base para exercícios do capítulo corrente

Exercício Extra – 1ª Parte

- Construa um conjunto de classes onde a primeira seja uma classe abstrata. As demais devem ser subclasses da primeira. Todas as classes devem definir variáveis de instância, seus métodos get e set, um método toString e um construtor sobrecarregado para inicializar todas variáveis definidas na classes e na superclasse. Sempre que necessário, chame o construtor da superclasse.

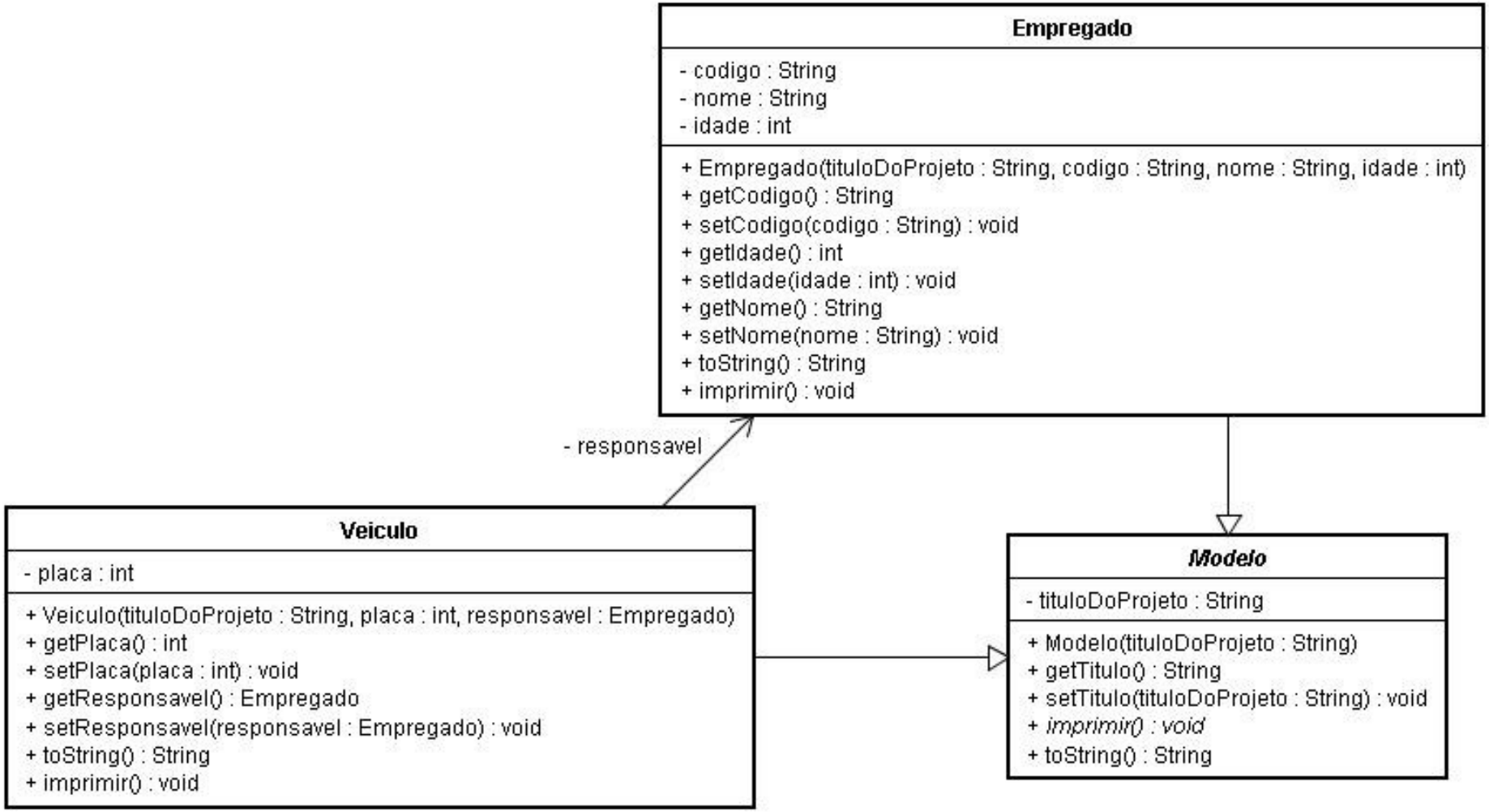
Exercício Extra – 2ª Parte

- Crie uma aplicação que use o conjunto de classes construído na primeira parte do exercício. Pelo menos dois objetos de cada classe não-abstrata devem ser instanciados, inicializados e depois impressos. Acrescente um método abstrato na classe abstrata e o implemente em todas as subclasses. Na aplicação, faça a chamada a este último método, demonstrando o polimorfismo.

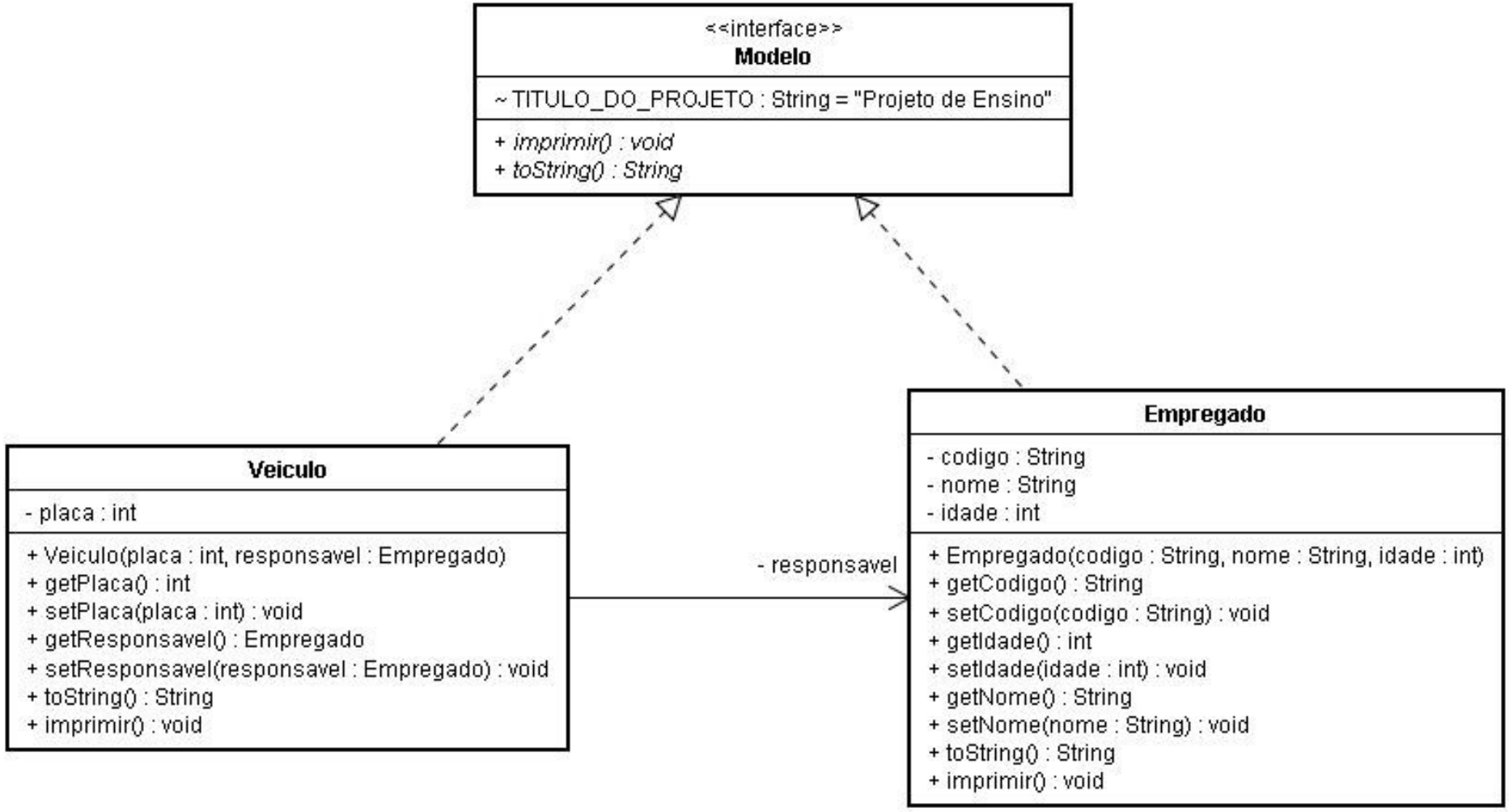
Exercício Extra – 3ª Parte

- Gere uma nova versão das classes criadas nas partes 1 e 2, trocando a classe abstrata por uma interface. Faça todos os ajustes necessários para que sua aplicação funcione. Depois, analise as diferenças entre a classe abstrata e a interface, indicando quando devemos optar por usar cada uma.

Solução com classe abstrata



Solução com interface



Classes Anônimas

- Classes abstratas podem ser estendidas por classes anônimas
- Interfaces podem ser implementadas por classes anônimas
- Ver exemplos

Classe Anônima que estende classe abstrata

```
public abstract class ClasseAbstrata {
    int a;
    public ClasseAbstrata(int a) {
        this.a = a;
    }
    abstract void imprime();
}

public class TestaClasseAnonimaExtendendoAbstrata {
    public static void main(String[] args) {
        ClasseAbstrata ca = new ClasseAbstrata(123) {
            void imprime() {
                System.out.println("a = " + a);
            }
        };
        ca.imprime();
    }
}
```

Classe Anônima que implementa interface

```
public interface Interface {
    int a = 123;
    void imprime();
}


public class TestaClasseAnonimaImplementandoInterface {

    public static void main(String[] args) {
        Interface i = new Interface() {
            public void imprime() {
                System.out.println("a = " + a);
            }
        };
        i.imprime();
    }
}
```

Acesso a partir de classe anônima

- A classe anônima tem um **relacionamento especial** com a classe onde foi criada podendo **acessar** os seus membros **privados**
- **Não** é possível acessar **variáveis locais**, mesmo que estas tenham sido declaradas no mesmo método onde a classe foi declarada. A **exceção** é para **constantes** (variáveis marcadas como *final*). Essa regra se aplica a qualquer **classe interna**.

Classes internas

- As **classes anônimas** são casos especiais de **classes internas** e seu uso deve ser feito de forma controlada, pois **aumenta a complexidade** do código e, conseqüentemente, gera efeitos colaterais para o processo de **manutenção**
- Apenas classes podem ser **internas aos métodos**, mas não **interface** ou **enum**