Programação por Contrato
Invariantes

Papéis do programador

- Relativamente a um módulo o programador pode ser
  - Produtor – Se o desenvolveu total ou parcialmente
  - Consumidor – Se o usa de alguma forma

Programador produtor

- Desenvolve o mecanismo do módulo, ou seja, implementa o módulo
- Preocupa-se com:
  - *O que faz* o módulo
  - *Como se usa* o módulo
  - *Como funciona* o módulo

1. Comentários de documentação
2. Contrato (a ver mais à frente)

1. Interface do módulo
2. Exemplos (código de teste)

Implementação do módulo (e.g., corpo de um método).

Programador consumidor

- Usa o módulo
- Preocupa-se com:
  - *O que faz* o módulo
  - *Como se usa* o módulo

1. Comentários de documentação
2. Contrato (a ver mais à frente)

1. Interface do módulo
2. Exemplos (código de teste)

Programação por contrato

- Produtor de um módulo:
  - Fornece manual do consumidor (como se usa)
  - Estabelece *contrato* com seus consumidores (o que faz)

Especificação e verificação do contrato

- Pré e pós-condições especificadas no comentário de documentação das rotinas: etiquetas `@pre` e `@post`
- Pré-condições verificadas usando
  - Métodos não privados: *excepções*
  - Métodos privados: *asserções*
- Pós-condições verificadas usando *asserções*

Vantagens da verificação do contrato

- Erros detectados mais cedo
- Localização mais precisa de erros
- Menos erros no programa final
- Maior confiança na qualidade do código

Manual do consumidor e contrato

```
/**
   Calculates and returns the gcd (greatest common
   divisor) of its two integer arguments ({@code m}
   and {@code n}).
   @param   m the first integer whose gcd is needed
   @param   n second integer whose gcd is needed
   @return  the gcd of {@code m} and {@code n}
   @pre     m ≠ 0 ∨ n ≠ 0
   @post    0 < gcd ∧ m ÷ gcd = 0 ∧ n ÷ gcd = 0 ∧ …
*/
public static int gcd(final int m, final int n) { … }
```

Versão do gcd (mdc) mais genérica que a definida nas primeiras aulas.

Estas etiquetas não são suportadas nativamente pelo javadoc.

Verificação do contrato: pré-condições

```
/** … */
public static int gcd(final int m, final int n) {
    if (m == 0 && n == 0)
        throw new IllegalArgumentException(
            "Illegal arguments: " + m + ", " + n);

    … // implementação    …

    … // verificação da pós-condição

    return …;
}
```

Verificação da pré-condição do contrato.

Verificação do contrato: pós-condições

```
/** … */
public static int gcd(final int m, final int n) {
    … // verificação da pré-condição

    … // implementação

    assert 0 < r : r;
    assert m % r == 0 : "m: " + m + ", r: " + r;
    assert n % r == 0 : “n: " + n + ", r: " + r;

    return r;
}
```

Verificação dos vários termos da pós-condição do contrato. Apenas dentro do que é razoável (não verifica se é o maior dos divisores)!

(Condição) invariante de instância

- Verdadeira se e só se a instância for válida e coerente

- Tem de ser verdadeira:
  - Após construção (através da interface pública)
  - Após qualquer alteração (através da interface pública)

(Condição) invariante de instância

- Deve ser verificada:
  - Após a construção (através da interface pública)
  - Após qualquer alteração (através da interface pública)
- Pode ser verificada:
  - Antes de qualquer acesso (através da interface pública)
- Verificação dá ao *produtor* segurança acerca da:
  - Coerência dos atributos
  - Correcção dos métodos

Invariante de instância: Exemplo

```
public class Lamp {
    private boolean isOn = false;
    private boolean isBroken = false;
    …
    private boolean stateIsValid() {
        return !(isBroken && isOn);
    }
}
```

Por razões técnicas, pode ser necessário tornar público o predicado (método devolvendo `boolean`) que calcula o invariante, mesmo sabendo que ele faz parte da *implementação* de uma classe.

Invariante de instância: Exemplo

```
public class Lamp {
    …
    public boolean turnOn() {
        assert stateIsValid();
        if (!isBroken)
            isOn = true;
        assert stateIsValid();
    }
    …
}
```

Verificar no início dos métodos não privados.

Verificar *sempre* no final dos métodos modificadores não privados.

Invariante de instância: Exemplo

```
public class Contact {
    private String name;
    private String phone;
    …
    private boolean stateIsValid() {
        return name != null
            && phone != null &&
            && 8 <= phone.length
            && …
    }
}
```

Mais informação / Referências

- Y. Daniel Liang, *Introduction to Java Programming*, 7.ª edição, Prentice-Hall, 2010.

Sumário

- Programação por Contrato
- Invariantes