Acoplamento e Coesão

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3

Modularidade

- Noção de Módulo
  - Unidade identificável em relação a compilação (linguagem de programação)
  - Unidade identificável em nível de design

  - Modularidade é um conceito que tem se mostrado útil em diversos campos que lidam com sistemas complexos. De fato, busca-se modularizar para “pontuar” complexidade

- Idéia central
  - Divisão da complexidade, objetivando desenvolvimento, manutenção e reuso

Modularidade

- Idéias relacionadas
  - Independência entre módulos
  - Interdependência intra-modular

Módulo

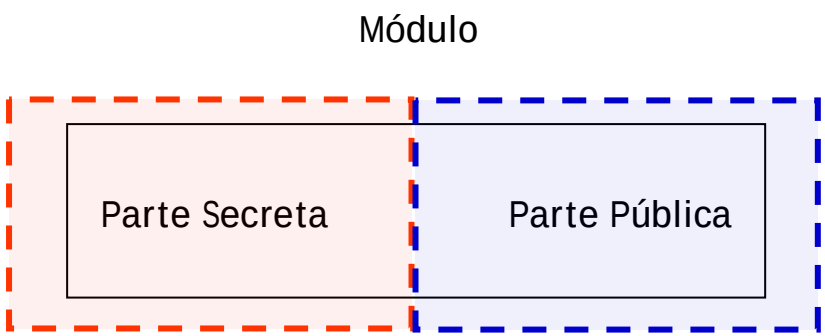
- Definição
  - Um módulo é uma unidade cujos elementos estruturais estão fortemente conectados entre si e (relativamente) fracamente conectados com elementos de outras unidades.
  - Em outras palavras, módulos são unidades estruturalmente independentes em um sistema, mas que funcionam juntos
- Idéia subjacente
  - O sistema deve prover uma arquitetura que permite independência de estrutura e integração de função.

Abstração, Complexidade e Interface

- Idéia Central
  - Um sistema complexo pode ser gerenciado através da divisão em partes menores, seguida pelo exame de cada parte separadamente. Contudo, quando a complexidade de um dos elementos extrapola um determinado limiar, tal complexidade pode ser isolada através da definição de uma abstração separada que possui uma interface simples.
    - A abstração esconde a complexidade do elemento; a interface indica como o elemento interage com o sistema.

Abstração, Complexidade e Interface

- Idéia Central
  - Um subconjunto das propriedades do módulo é escolhido como parte pública, oficial, disponível para os clientes



A parte secreta esconde a complexidade inerente ao módulo

A parte pública de um módulo compreende a Interface do mesmo, a partir da qual outros módulos podem interagir

Modularidade

- Considerações
  - Módulos e conexões entre módulos
- Critérios Estruturais
  - Coesão e Acoplamento

Acoplamento e Coesão

- Exemplo
  - Construir classe Calculadora com operação de somar, que deve retornar o resultado da soma
  - Coesão
    - Código não coeso é: implementação do módulo Calculadora e da operação somar. Além de somar envia e-mail para o usuário com o resultado da operação e mostra mensagem na tela com o resultado.
    - Problema: manda email e mostra mensagem em tela, desnecessário para o requisitado

Acoplamento e Coesão

- Coesão
  - Perguntar sempre: Cada entidade e cada funcionalidade realizam o seu trabalho? Ou fazem mais do que devem?
  - Buscar: código mais coeso possível (mais simples possível)
  - Código sendo coeso, facilita a reutilização

Acoplamento e Coesão

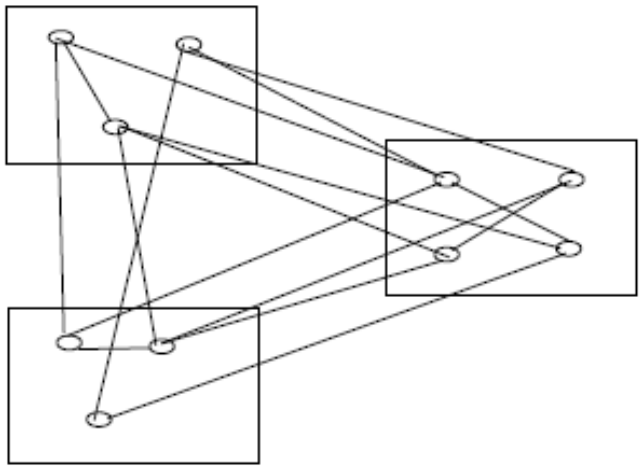
- Acoplamento
  - Código acoplado é: programador implementa o módulo calculadora no mesmo arquivo da tela de entrada de dados
  - Problema: a calculadora pode ser reutilizada em outro projeto, mas acaba levando uma tela de entrada de dados desnecessária.
  - Buscar: código menos acoplado possível
  - Codigo menos acoplado facilita a reutilização

Acoplamento e Coesão

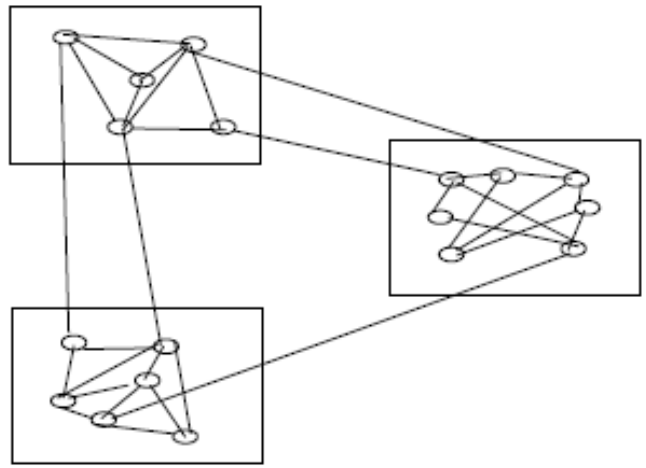
- Coesão
  - Cada módulo deve ser altamente coeso (highly cohesive)
    - Módulo é visto como "unidade“
    - Componentes internos a um módulo estão relacionados

- Acoplamento
  - Módulos devem apresentar baixo acoplamento (low coupling)
    - Módulos possuem poucas interações com outros módulos

Acoplamento e Coesão



Acoplamento Alto

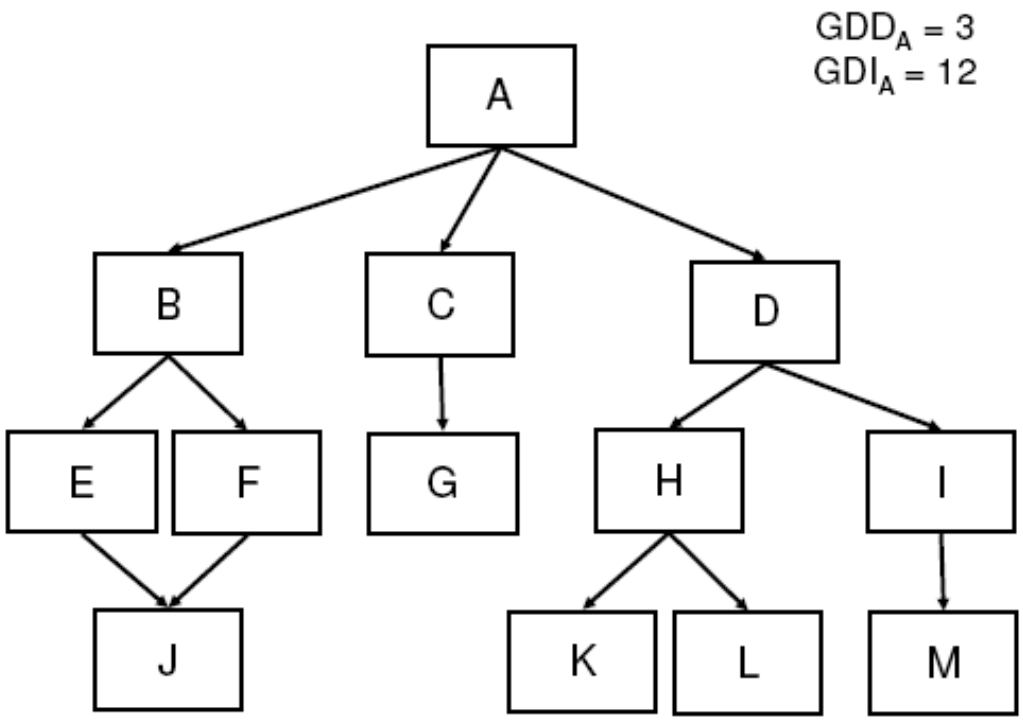


Coesão Alta
Acoplamento Baixo

Grau de Dependência

- Grau de dependência
  - Medida de ligação entre classes
  - Grau de dependência direto: Indica Quantas classes são referenciadas diretamente por uma determinada classe. Grau de dependência indireto: indica quantas classes são referenciadas diretamente ou indiretamente (recursivamente) por uma determinada classe

Grau de Dependência



Grau de Dependência

- Uma classe A referencia diretamente uma classe B se:
  - A é subclasse direta de B
  - Atem atributo do tipo B
  - A tem parâmetro de método do tipo B
  - A tem variáveis em métodos do tipo B
  - A chama métodos que retornam valores do tipo B

Coesão

Coesão

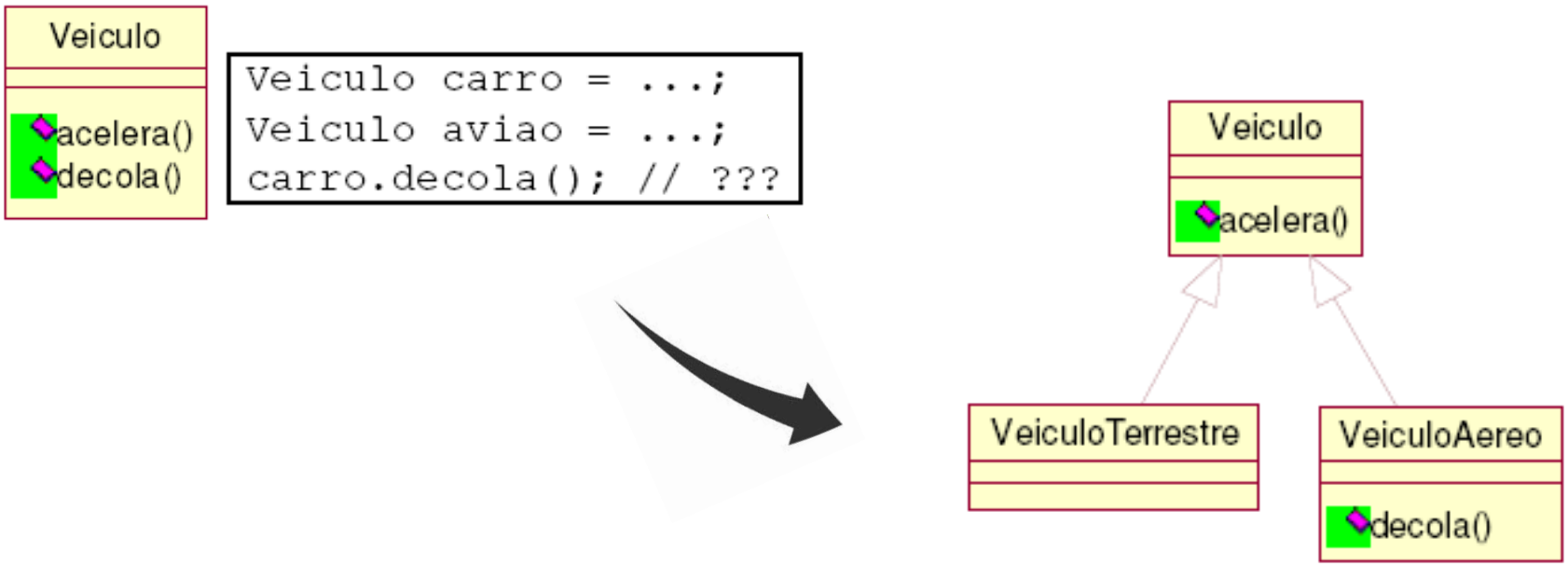
- Definição
  - Coesão é a medida da afinidade mútua entre os componentes de um módulo. Coesão pode ser vista como a “cola” que mantém componentes de um módulo “juntos”.
  - Em geral, buscamos maximizar a coesão de um módulo
    - Módulos fortemente coesos são desejáveis

Coesão - Tipos

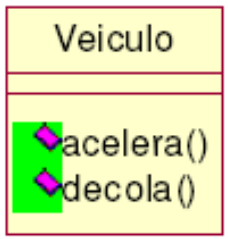
- Tipos de Coesão
  - Coesão de instância mista
  - Coesão de domínio misto
  - Coesão de papel misto
  - Coesão alternada
  - Coesão múltipla
  - Coesão funcional

Coesão de Instância Mista

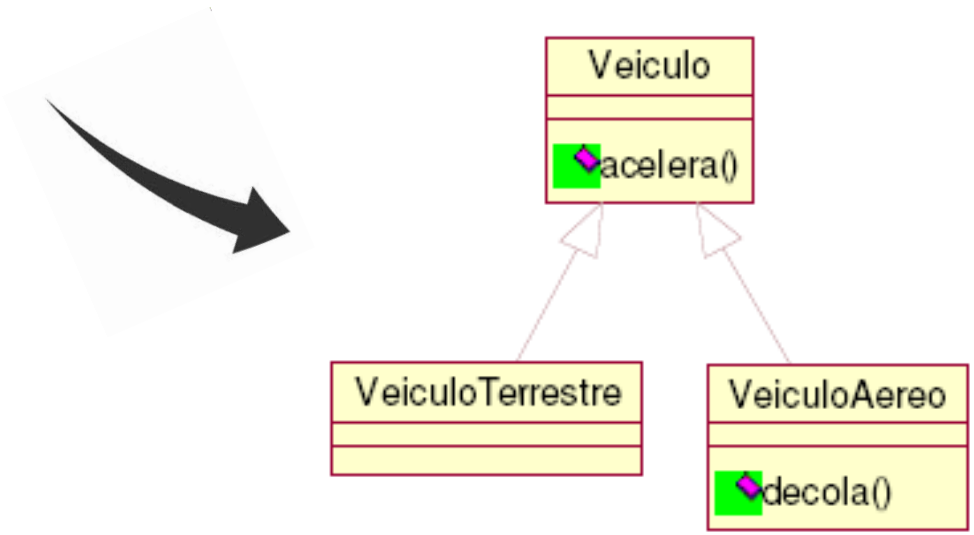
- Coesão de instância mista
  - Ocorre quando algumas características ou comportamentos não são válidos para todos os objetos da classe.Normalmente, problemas de coesão de instância mista podem ser corrigidos através da criação de subclasses utilizando herança



Coesão de Instância Mista



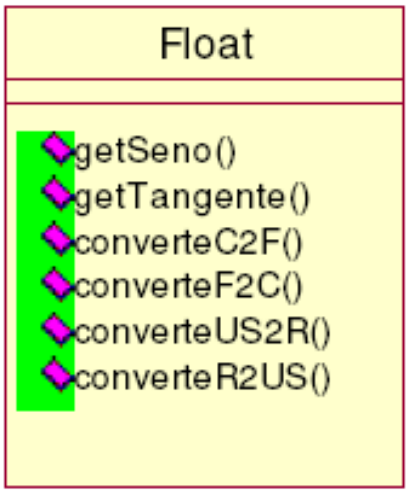

```
Veiculo carro = ...;
Veiculo aviao = ...;
carro.decola(); // ???
```

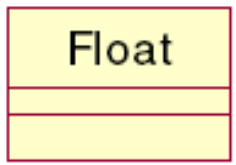


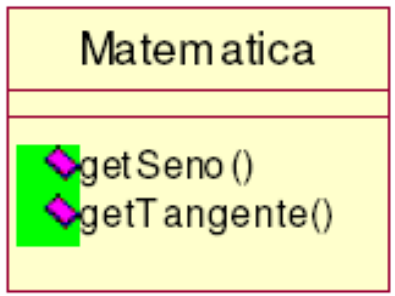
Coesão de Domínio Misto

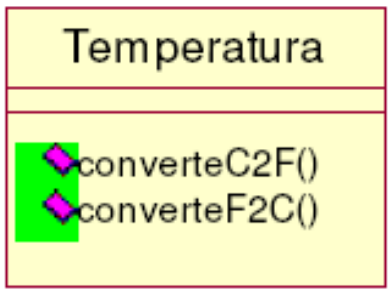
- Coesão de domínio misto
  - Ocorre quando algumas características ou comportamentos não fazem parte do domínio em questão. Quando a coesão de domínio misto ocorre, a classe tende a perder o seu foco com o passar do tempo

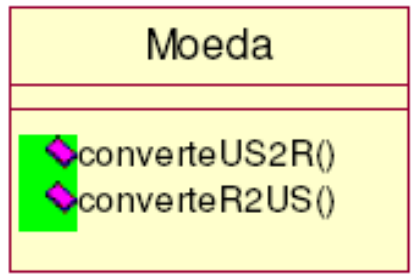
Coesão de Domínio Misto











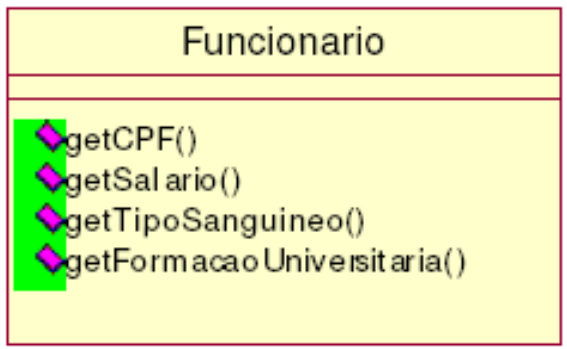
A solução para esse problema é a separação das responsabilidade em classes de diferentes domínios, tirando a sobrecarga da classe Float

Coesão de Papel Misto

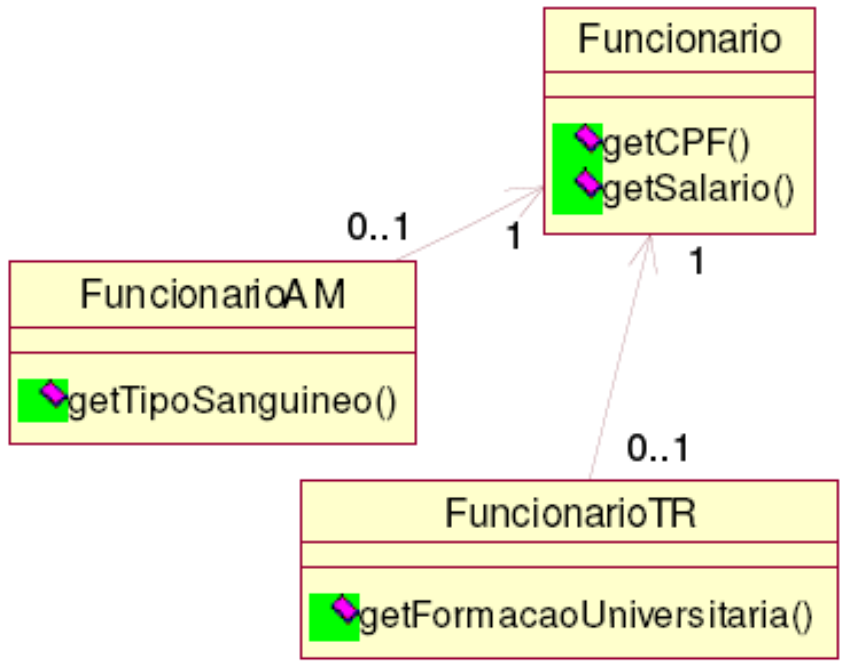
- Coesão de papel misto
  - Ocorre quando algumas características ou comportamentos criam dependência entre classes de contextos distintos em um mesmo domínio. Problemas de coesão de papel misto são os menos importantes dos problemas relacionados à coesão.
  - O maior impacto desse problema está na dificuldade de aplicar reutilização devido a bagagem extra da classe.

Coesão de Papel Misto

Exemplo: algumas das características e comportamentos da classe Funcionario não são necessárias em todos contextos



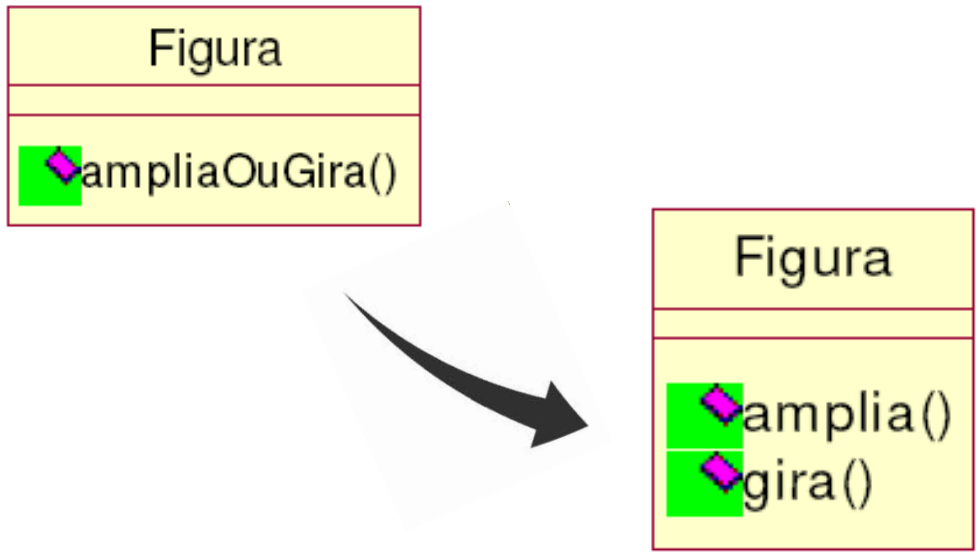
A classe Funcionário pode ser reutilizada sob o ponto de vista dos sistemas de assistência médica (AM) ou de treinamento (TR)



Coesão Alternada

- Coesão alternada
  - Ocorre quando existe seleção de comportamento dentro do método. Usualmente o nome do método contém OU, de algum modo, é informada a chave para o método poder identificar o comportamento desejado
  - Internamente ao método é utilizado switch-case ou if aninhado. Para corrigir o problema, o método deve ser dividido em vários métodos, um para cada comportamento

Coesão Alternada



Coesão Múltipla

- Coesão múltipla
  - Ocorre quando mais de um comportamento é executado sempre em um método. Usualmente o nome do método contém E não é possível executar um comportamento sem que o outro seja executado, a não ser através de improviso (ex.: parâmetro null)
  - Para corrigir o problema, o método deve ser dividido em vários métodos, um para cada comportamento

Coesão Funcional

- Coesão funcional
  - Ocorre quando é encontrado o nível ideal de coesão para uma classe
  - Também é conhecida como coesão ideal

Acoplamento

Acoplamento

- Definição
  - Acoplamento é a medida da força das conexões entre módulos
  - É uma medidade de quão fortemente um elemento
    - está conectado a, tem conhecimento de, ou depende de outros elementos
  - Acoplamento forte indica forte dependência entre módulos
  - Em geral, buscamos minimizar o acoplamento entre módulos
    - Módulos fracamente acoplados são desejáveis

Acoplamento

- Consequências - Acoplamento forte
  - Dificulta compreensão
  - Dificulta manutenção
    - A manutenção é comprometida, pois uma mudança em um módulo pode resultar na necessidade de mudar outros módulos, em série

Acoplamento - Tipos

- Acoplamento de Conteúdo
  - Um módulo afeta o trabalho de outro
  - Exemplos: modificar dados de outro módulo, ou controle passa de um módulo para o meio de outro (desvio)
- Acoplamento de Compartilhamento
  - Dois ou mais módulos compartilham dados
- Acoplamento Externo
  - Módulos se comunicam através de meio externo, por exemplo, um arquivo

Acoplamento - Tipos

- Acoplamento de Controle
  - Um módulo conduz a execução de outro, passando informação de controle necessária. Exemplo: uso de flags (inteiros, booleanos)
- Acoplamento “stamp”
  - Estruturas de dados completas são passadas de um módulo para outro
  - Exemplo: “record”
- Acoplamento de Dados
  - Apenas dados simples são passados entre módulos
  - Exemplo: tipos simples e arrays

Acoplamento

- Observações
  - Níveis de acoplamento dependem de tecnologia
  - Exemplos
    - Acoplamento de dados considera tipos simples e arrays, não registros.
    - Acoplamento de controle assume passagem de dados simples

- Contudo
  - Novas tecnologias trouxeram novos níveis e tipos de acoplamento e coesão

Acoplamento

- Atualmente
  - Acoplamento de dados
    - Módulos podem passar estruturas de dados complexas
  - Acoplamento de conteúdo
    - Módulos podem permitir acesso aos seus dados para alguns módulos e proibir para outros (níveis de visibilidade)

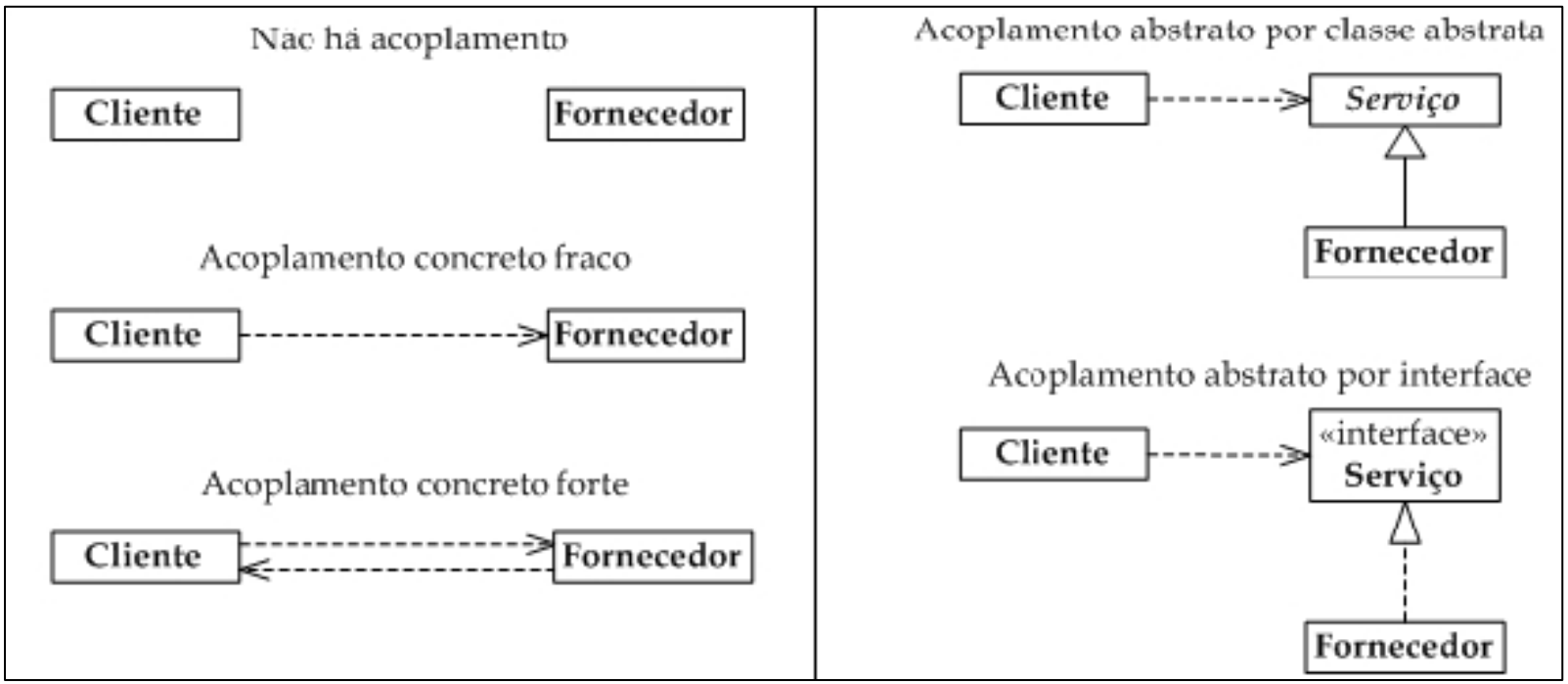
Ideal: Interfaces

- Uso de Interface
  - Coesão forte e Acoplamento Fraco
    - Tornam comunicação entre programadores mais simples
    - Mudanças influenciam menos outros módulos
    - Favorecem a reusabilidade
    - Facilitam a compreensão
    - Tendem a diminuir a ocorrência de erros

Adicionalmente

- Observações
  - Todo módulo deve se comunicar com outros no mínimo possível
  - Se dois módulos se comunicam, eles devem trocar o mínimo de informações

Acoplamento



Herança e Acoplamento

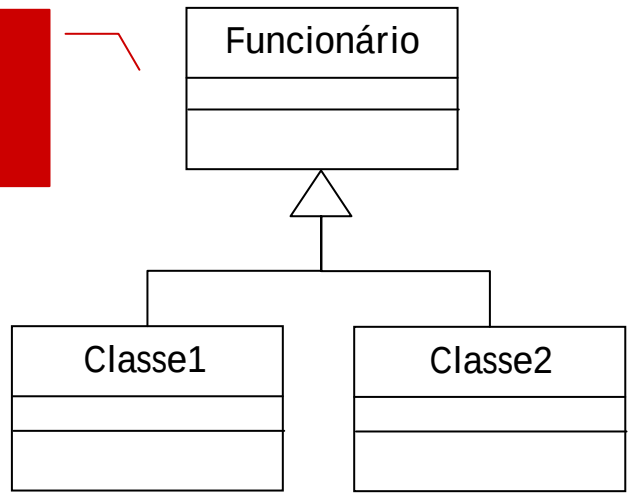
- Herança e Acoplamento

  - Note que o uso de herança aumenta o acoplamento entre as classes, isto é, o quanto uma classe depende de outra. A relação entre classe mãe e filha é muito forte e isso acaba fazendo com que o programador das classes filhas tenha que conhecer a implementação da classe pai e vice-versa - fica difícil fazer uma mudança pontual no sistema.

  - Por exemplo, imagine se tivermos que mudar algo na nossa classe Funcionario, mas não quiséssemos que todos os funcionários sofressem a mesma mudança. Precisaríamos passar por cada uma das filhas de Funcionario verificando se ela se comporta como deveria ou se devemos, agora, sobreescrever o tal método modificado. Esse é um problema da herança, e não do polimorfismo, que resolveremos mais tarde com a ajuda de Interfaces.

Herança e Acoplamento

Uma alteração na superclasse pode provocar efeitos indesejáveis nas subclasses



Herança e Acoplamento – Um pouco Mais

- Acoplamento e Herança
  - Acoplamento fraco é um dos principais requisitos para se construir software orientado a objetos (OO) de qualidade. A capacidade de uma classe em herdar o comportamento de outra(s) é uma das principais características do paradigma OO. A principal vantagem é poder criar novas classes quase de graça, aproveitando o código de outra.

Herança e Acoplamento

- Problemas de Acoplamento Forte
  - Uma classe com acoplamento forte depende muito (em geral sem necessidade) de outras. Isso pode conduzir aos seguintes problemas
    - Classes difíceis de aproveitar tendo em vista que sempre que esta for utilizada todas as outras das quais ela depende devem estar presentes;
    - Alterações nas classes relacionadas podem forçar mudanças locais e
    - São difíceis de compreender isoladamente.

Herança e Acoplamento

- Formas de Acoplamento
  - Formas comuns de acoplamento ocorrem através de
    - Variáveis de instância
    - Chamada de serviços em outra classe
    - Uma classe deriva direta ou indiretamente de outra
    - Uma classe implementa uma determinada interface.
    - Etc.
  - Resumindo, sempre que uma classe referencia um outro tipo em qualquer uma das circunstâncias acima está ocorrendo acoplamento

Herança e Acoplamento

```
public class X

{

    private ClasseConcretaY var1;



    void M1(ClasseConcretaW var2 ) { … }

}
```

Existem dois pontos principais de acoplamento, na variável de instância var1, que é do tipo ClasseConcretaY, e no argumento var2, que é do tipo ClasseConcretaW. Nestas duas partes do código a classe X referencia outras duas classes concretas. Isso significa que, sempre que esta classe for utilizada, as outras duas deverão estar disponíveis no espaço de nomes do programa.

Herança e Acoplamento

- Pergunta
  - Referenciar outras classes sempre causa problemas de acoplamento?
    - A resposta é, depende!
    - Referenciar classes estáveis e disseminadas raramente é um problema. Por exemplo, utilizar o pacote java.util num programa em Java dificilmente causará problemas futuros de acoplamento, uma vez que qualquer ambiente de execução Java contém essa biblioteca. O problema está em classes instáveis, pouco conhecidas, ou seja, nas classes que são criadas para atender os problemas específicos dos projetos.

Herança e Acoplamento

- Como diminuir o acoplamento?
  - Uma regra geral para diminuir o acoplamento é "programar para uma interface e não para uma implementação". No exemplo do código da classe X isso significa substituir as declarações das classes concretas por declarações de interfaces. Fazendo isso desacopla-se o código de uma implementação específica, tornando-o dependente apenas de uma interface. Essa não é a solução definitiva, porém ajuda muito.

Herança e Acoplamento

- Sobre herança
  - A herança é a principal característica de distinção entre um sistema de programação orientado a objeto e outros sistemas de programação.
  - O principal benefício da herança é o reaproveitamento de código. A herança permite ao programador criar uma nova classe programando somente as diferenças existentes na subclasse em relação à superclasse. Isto se adeqüa bem a forma como compreendemos o mundo real, no qual conseguimos identificar naturalmente estas relações.
  - A reutilização por meio de subclasses é dito “reutilização de caixa branca”, pois usualmente expõe o interior das classes ancestrais para as subclasses. Qual o problema associado?

Herança e Acoplamento

- Uso de herança
  - A decisão de derivação a partir de uma superclasse precisa ser cuidadosamente considerada, uma vez que ela é uma forma muito forte de acoplamento. As classes ancestrais freqüentemente definem pelo menos parte da representação física das suas subclasses. A implementação de uma subclasse, desta forma, torna-se tão amarrada à implementação da sua classe mãe que qualquer mudança na implementação desta forçará uma mudança naquela.

Herança e Acoplamento

```java
abstract public class X{

    private final int MAX = 100;

    public int CalculaMaximo(int i){
        return  i * MAX;
    }

    public int UsaMaximo(int i){

        int maximo = CalculaMaximo(i);
        //faz alguma coisa de útil com maximo …

        return maximo;
    }
}
```

```java
public class Y extends X{

    public void UsaMetodoDaClasseX(){
        //...
        int aux = UsaMaximo(10);
        //...
    }
}
```

Herança e Acoplamento

```
1  abstract public class X{
2
3      private final int MAX = 100;
4
5      public int CalculaMaximo(int i){
6          return  i * MAX;
7      }
8
9      public int UsaMaximo(int i){
10
11         int maximo = CalculaMaximo(i);
12         //faz alguma coisa de útil com maxi
13
14         return maximo;
15     }
16 }
```

A interface pública permanece inalterada

Alterou a implementação do método CalculaMaximo(int i)

```
1  abstract public class X{
2
3      private final int MAX = 100;
4
5      public int CalculaMaximo(int  i){
6
7          return  (int) i * (MAX /100);
8          // Aproxima o resultado com cast
9      }
10     public int UsaMaximo(int i){
11         return 0;
12     }
13 }
```

Herança e Acoplamento

- Problemas com exemplo
  - Essa alteração na maneira como o máximo está sendo calculado pode gerar efeitos colaterais na classe Y. Assim, para que a classe Y continue funcionando, esta precisaria ser adaptada à nova realidade

  - Este exemplo simples ajuda a mostrar como uma alteração na implementação de um método numa classe base pode provocar anomalias nas suas classes derivadas. Observe que não ocorreu uma alteração de interface, o que necessariamente (e notoriamente) implica em alterações nas classes clientes. É por isso que a herança, em particular, revela um alto acoplamento.

    - Além das hierarquias de classes criadas estarem suscetíveis às mudanças de interface, estão suscetíveis também às alterações nas implementações dos métodos.

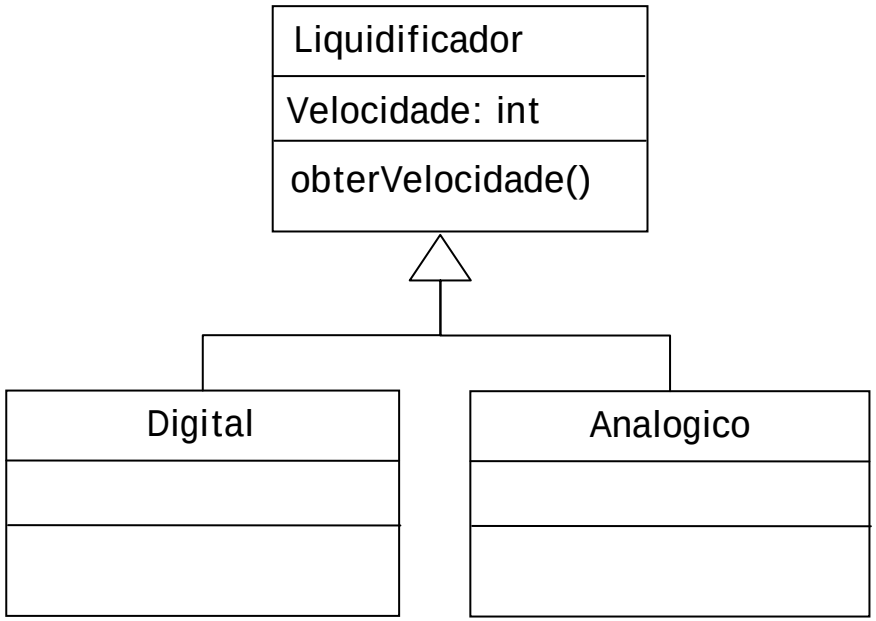
Herança e Acoplamento

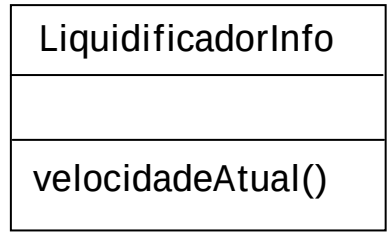
- Herança é ruim?
  - Não necessariamente. Como dito, a decisão de derivação a partir de uma superclasse precisa ser cuidadosamente considerada.
  - Herança favorece reuso de código, mas implica em acoplamento.
  - Pergunta: é possível evitar herança?
    - Sim. Uma outra forma de reaproveitar funcionalidade é através da composição de objetos. Novas funcionalidades são obtidas compondo objetos para obter funcionalidades mais complexas. Uma das vantagens desta abordagem é a flexibilidade em poder selecionar em tempo de execução qual objeto será usado na composição (contanto que este respeite a interface definida)

Adicionais

- Acoplamento
  - Sempre devemos expôr o mínimo possível de funcionalidades, para criar um baixo acoplamento entre as nossas classes. Isso pode ser obtido com restrições de visibilidade. Neste caso, devemos tornar visível somente o necessário
  - Polimorfismo ajuda a diminuir o acoplamento entre classes
    - Exemplo no próximo slide

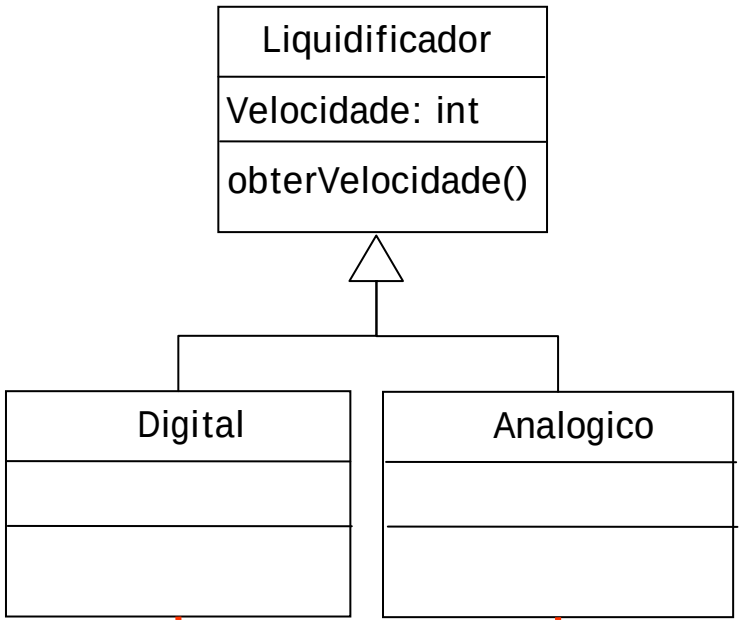
Polimorfismo e Acoplamento





Imaginemos que seja necessário implementar numa classe LiquidificadorInfo um método que seja capaz de imprimir a velocidade atual de objetos liquidificador os quais podem ser tanto do tipo digital como analógico.

Polimorfismo e Acoplamento

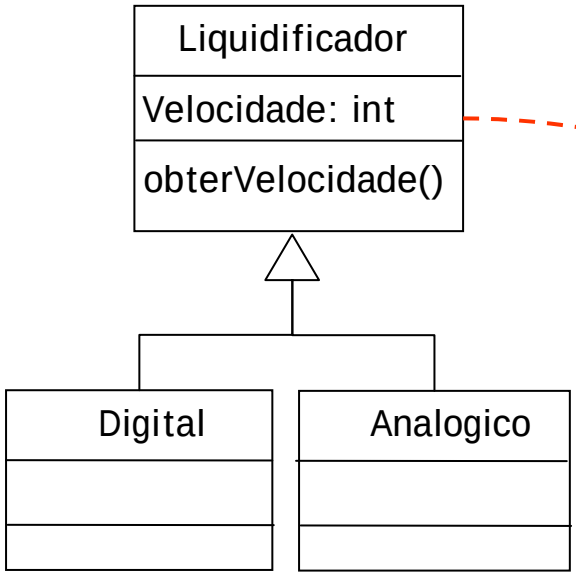



```
public class LiquidificadorInfo {

  public void velocidadeAtual(LiquidificadorAnalogico l) {
    System.out.println("Veloc. Atual: "+l.obterVelocidade());
  }

  public void velocidadeAtual(LiquidificadorDigital l) {
    System.out.println("Veloc. Atual: "+l.obterVelocidade());
  }
}
```

Polimorfismo e Acoplamento




```
1  public class LiquiInfo {
2
3      public void velocidadeAtual(Liquidificador l) {
4          System.out.println("Veloc. Atual: "+l.obterVelocidade());
5      }
6  }
```

Acoplamento e Coesão

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3