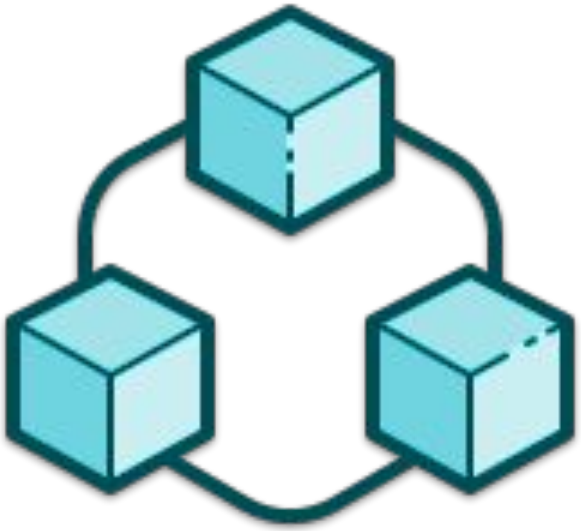


Aula 6

Abstração e Modularização

Rafael Geraldeli Rossi

Introdução

- Normalmente **um problema pode ser dividido em subproblemas**
- Fazendo isso, tornamos o problema **menos complexo** (filosofia Jack Estripador)
- Cada especialista em uma subparte poderá desenvolver sua parte na qual é especialista
- Vale ressaltar que conforme o problema começa a crescer, aumenta a **dificuldade de lidar com todos os detalhes ao mesmo tempo**

Introdução

- Por exemplo, no desenvolvimento de um relógio analógico e digital, o correto seria:
  - Um especialista em design desenvolver o modelo do relógio
  - Um especialista em eletrônica desenvolver a parte do *display* digital
  - Um especialista desenvolver o maquinário para a parte analógica
  - Um especialista em metalurgia desenvolver a carcaça do relógio
  - ...
- No final, todas essas partes serão acopladas em um único componente, a classe `Relógio`

Introdução

- Se deixássemos a cargo de uma pessoa muito boa mecanicamente ou eletronicamente o desenvolvimento de todo o relógio, isso poderia resultar em relógios do tipo:






Introdução

- No caso do desenvolvimento de um carro, normalmente não atribui-se a uma pessoa só o desenvolvimento do design, do motor, da parte dos freios, ...
- Se fosse, provavelmente teríamos um carro com um motor muito bom, mas feio de doer, ou ainda um carro muito bonito mas com um freio que poderia falhar a qualquer momento
- O correto seria:
  - Um engenheiro ou grupo de engenheiros determinar as medidas do carro (espaçamento entre os eixo, espaçamento entre os acentos, localização do motor, do tanque de gasolina, ...)
  - Um engenheiro ou grupo de engenheiros desenvolver o motor (cilindros, mecanismo de injeção, carburador, ...)

Introdução

- O correto seria:
  - Um outro engenheiro ou grupo, inclusive de outra companhia para desenvolver a parte elétrica
  - Um engenheiro, grupo, ou mesmo uma outra companhia para desenvolver o estofado
  - Um engenheiro, grupo, ou mesmo uma outra empresa para desenvolver ou fabricar o sistema de freios
  - Uma empresa para fabricar os pneus
  - ...
- No final, essas partes são combinadas para formar o carro completo
- Além disso, uma parte precisar ser alterada/modificada, não é preciso refazer o carro todos

Introdução

- Como já fizemos na aula passada, uma conta bancaria, pode ter, por exemplo, um campo Pessoa, o qual conterá as informações de uma pessoa que é o titular da conta

- Ao invés de um especialista no funcionamento da conta bancária ficar preocupado com como a Pessoa irá se comportar (atributos e métodos), este poderá se focar apenas nas operações pertinentes à conta bancaria (mensalidade, débito, crédito, empréstimo, investimentos, ...) e assumir que a classe Pessoa está funcionando corretamente ou que será desenvolvida corretamente

Abstração e Modularização

- A solução para lidar com um problema complexo é a **abstração** e a **modularização**
- O problema é dividido em sub-problemas, os sub-problemas são divididos em sub-sub-problemas, e assim por diante, **até que os problemas sejam pequenos o suficiente para serem tratados** $\rightarrow$ **dividir e conquistar**
- Casa sub-problema que funciona individualmente mas que pode ser combinado com outros problema também é conhecido como **módulo**
- Além disso, na abstração, devemos nos importar com os **aspectos relevantes do problema em questão**

Abstração e Modularização

- **Os princípios de abstração e modularização são também empregados no desenvolvimento de um software**
- Primeiro identifica-se os **subcomponentes** do programa que podem ser **programados de maneira independente**
- Depois, **os subcomponentes são combinados em um componente maior**, o qual será responsável por "encaixar" esses componentes e fazê-los cada um executar o seu papel
- Quem irá utilizar ou combinar os subcomponentes, só sabe que eles funcionam e sabem como interagir com eles $\rightarrow$ não necessariamente sabem o seu funcionamento interno

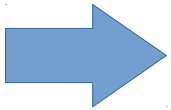
Abstração e Modularização

- Além disso, a modularização aumenta a potencialidade do reuso de código
  - Uma classe `Pessoa` que é desenvolvida e utilizada para uma classe `ContaBancaria` pode ser utilizada em outra classe, por exemplo, em uma classe `Aluno`, `Funcionario`, `Firma`, `Escola`, ...
  - Uma classe `Motor` pode ser utilizada em diferentes tipos de carros ou mesmo outros veículos automotores
  - ...

Abstração e Modularização

- Na programação orientada à objetos, **os componentes e os subcomponentes são as classes**
- Se formos fazer um carro utilizando programação a orientada a objetos
  - Ao invés de construir um carro como sendo um componente único, primeiro seriam projetados e construídos os subcomponentes de maneira separada (motor, caixa de marchas, direção, assentos, pneus, ...)
  - Depois, o componente carro seria montado considerando os subcomponentes individuais

Abstração e Modularização



Abstração e Modularização

- Além disso, ao programar de maneira modular, você pode rapidamente dar manutenção no módulo ou trocar os módulos por um mais novo ou mais adequado
- Em um exemplo da vida real, você poderia trocar um componente de um carro se ter que trocar o carro inteiro
- Em um exemplo de programação, você poderia trocar uma classe por outra que realize a mesma função, porém, que apresente mais campos ou mais funcionalidades (Ex: uma classe Pessoa, uma clases Menu, uma classe para realizar operações matemáticas)

Exemplo: Relógio Digital

- Vamos considerar um relógio digital bem simples $\rightarrow$ irá exibir apenas as informações das horas, minutos e segundos
- Uma maneira é considerar que o relógio consiste em um único *display* com 6 dígitos
- Portanto, a classe relógio teria que se preocupar em controlar os *displays*, inclusive com as regras de atualização dos *displays*
- Porém, é possível quebrar o desenvolvimento desse problema em sub-partes?

Exemplo: Relógio Digital

- Se abstrairmos para um nível bem mais baixo, podemos enxergar um relógio como sendo composto por 3 *displays* de dois dígitos cada $\rightarrow$ um para as horas, um para os minutos e um para os segundos

- Cada *display* tem seu próprio comportamento
  - Os *displays* dos segundos e minutos, ao atingir o valor 60, eles voltam para o valor 00
  - O *display* das horas, ao atingir 12 (ou 24), volta para 00

Exemplo: Relógio Digital

- Podemos entender esses *displays* como objetos que incrementam seus valorem em uma unidade, e, ao atingirem os valores limites (específicos para cada unidade de tempo), têm seus valores zerados

- Obviamente é preciso combinar esses *displays* em um relógio de forma que eles possam trabalhar em conjunto

Classe `Display`

- Vamos então nos preocupar com a menor parte do problema: o *display*
- Voltando ao raciocínio anterior, o *display* é algo (objeto) que tem se valor incrementado até um limite e depois seu valor é zerado
- Vamos considerar também que o *display* sempre exibirá um número composto por dois dígitos $\rightarrow$ 01, 09, 15, 35, 59, ...

Classe Display

```
1   public class Display {
2
4       private int limite;
5       private int valor;
6
7       public Display(int limite){
8           this.limite = limite;
9           valor = 0;
10      }
11
12      public int getValor(){
13          return this.valor;
14      }
15
16      public void setValor(int valor){
17          this.valor = valor;
18      }
19
21      public String exibir(){
22          String display = "";
23          if(this.valor < 10){
24              display = "0" + valor;
25          }else{
26              display = "" + valor;
27          }
28          return display;
29      }
30
31      public void incrementar(){
32          valor = (valor + 1) % limite;
33      }

}
```

Classe `Relogio`

- Uma vez tendo os módulos dos *displays* desenvolvidos, podemos utilizá-los no relógio

- A função da classe `Relogio` então será combinar esses *displays* e interagir com eles
  - Atualizar a hora
  - Fazê-los exibir a hora atualizada

Classe Relogio

```
1   public class Relogio {
2
3       private Display horas;
4       private Display minutos;
5       private Display segundos;
6
7       public Relogio(){
8           horas = new Display(24);
9           minutos = new Display(60);
10          segundos = new Display(60);
11      }
12
13      public Relogio(int hora, int minuto, int segundo){
14          horas = new Display(24);
15          minutos = new Display(60);
16          segundos = new Display(60);
17
18          horas.setValor(hora);
19          minutos.setValor(minuto);
20          segundos.setValor(segundo);
21      }
22
23      public void atualizar(){
24          segundos.incrementar();
25          if(segundos.getValor() == 0){
26              minutos.incrementar();
27              if(minutos.getValor() == 0){
28                  horas.incrementar();
29              }
30          }
31      }
32
33      public void exibirHora(){
34          System.out.print("\r" + horas.exibir() + ":"
35          + minutos.exibir() + ":" + segundos.exibir());
36      }
37
38  }
```

Classe Programa

- Uma vez desenvolvida a classe relógio, podemos instanciá-la e usá-la

```
1   public class Programa {
2
3       public static void main(String[] args){
4
5           Relogio relogio = new Relogio();
6
7           while(true){
8               relogio.atualizar();
9               relogio.exibirHora();
10              try {
12                  Thread.sleep(1000);
13              } catch (Exception e) {
14                  System.err.println("Erro ao atualizar o relógio");
15                  System.exit(0);
16              }
17          }
18
19      }
20  }
```

Exercício

- Vamos complementar o exercício realizado na última aula (Aula 5)

- O complemento se dará da seguinte forma:
  - Separar os componentes do menu em funções

Material Complementar

- Programação Orientada a Objetos

  `http://www.linhadecodigo.com.br/artigo/506/programacao-orientada-a-objetos.aspx`

Imagem do Dia

Programação Orientada a Objetos
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010] e [Barnes and Kolling, 2016]

Referências Bibliográficas I


Barnes, D. and Kolling, M. (2016).
Objects First with Java: A Practical Introduction Using BlueJ.
Pearson Education.


Deitel, P. and Deitel, H. (2010).
Java: How to Program.
How to program series. Pearson Prentice Hall, 8th edition.