FIAP GRADUAÇÃO

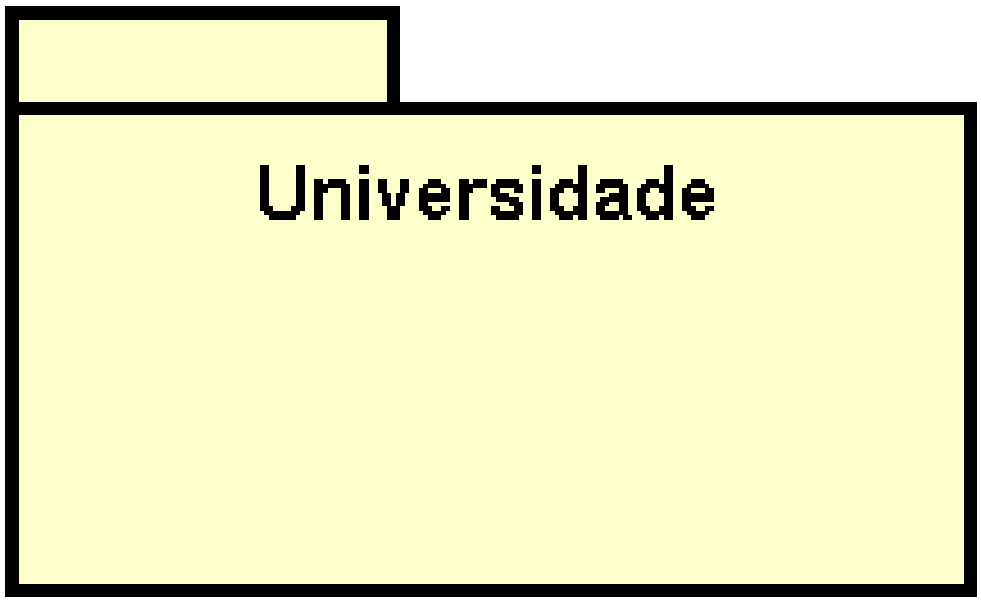
Orientação a Objetos - Agenda

- Pacotes
- Modificador de acesso (visibilidade)
- sobrecarga de métodos
- **this**
- Encapsulamento
- JavaBeans

Pacotes

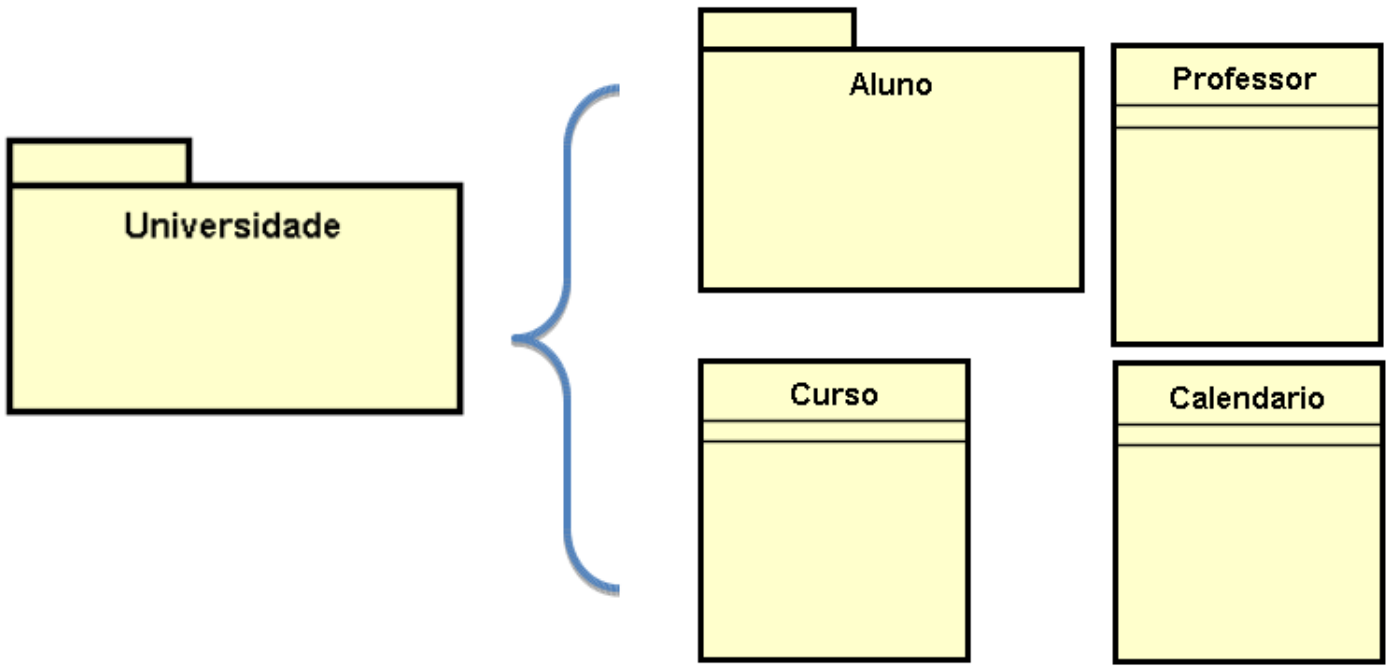
- Além das classes, o Java provê um recurso adicional que ajuda a modularidade: o uso de pacotes;
- Pacotes permitem a criação de espaços de nomes, além de mecanismos de controle de acesso;
- Pacotes são tipicamente implementados como diretórios;
- Os arquivos das classes pertencentes ao pacote devem ficar em seu diretório;
- Hierarquias de pacotes são construídas através de hierarquias de diretórios;

Pacotes - Representação Gráfica



Pacotes

- Um pacote pode conter classes e outros pacotes
- O pacote Universidade contém um pacote e três classes
- Veja a figura abaixo:



Pacotes

- "Empacotando"uma Classe:
  - Para declararmos uma classe como pertencente a um pacote, devemos:
    - declarar em um arquivo dentro do diretório que representa o pacote;
    - declarar, na primeira linha do arquivo, que a classe pertence ao pacote;
    - Por exemplo a classe Pessoa na pasta br/com/fiap/modelo:

      ```
      package br.com.fiap.modelo
      ```
- Importação de Pacotes:
  - Podemos usar o nome simples (não qualificado) de uma classe que pertença a um pacote se importarmos a classe;
  - A importação de uma classe (ou classes de um pacote) pode ser feita no início do arquivo, após a declaração do pacote (se houver);
  - As classes do pacote padrão java.lang não precisam ser importadas (Ex.: String)

Modificadores de acesso

Modificador de acesso é um elemento da orientação a objetos que possibilita formas de controlar o acesso aos membros (atributos e métodos) de uma classe. No mínimo, podemos controlar o que é público ou privado. Também chamado de **visibilidade**.

- O Java disponibiliza quatro modificadores de acesso:
  - private
  - protected
  - public
  - Quando nenhum modificador é utilizado, dizemos que o membro está com o nível de acesso default, também conhecido como package;
- Membros públicos podem ser acessados por todos, enquanto os privados só podem ser acessados pela própria classe

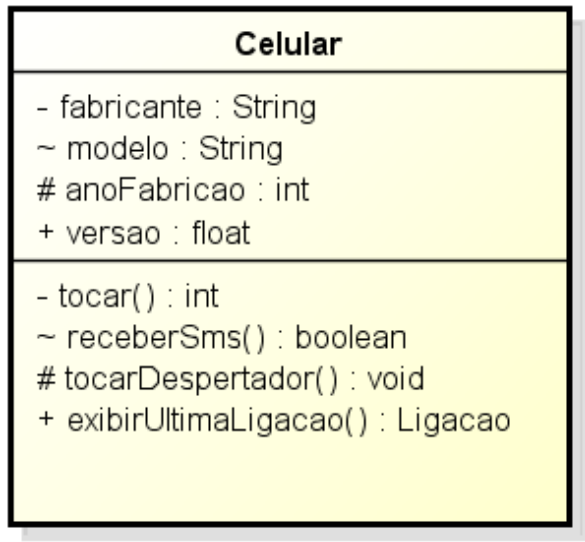
Modificadores de acesso

Símbolo	Palavra-chave	Descrição
-	**private**	Atributos e métodos são acessíveis somente nos métodos da própria classe. Este é o nível mais rígido de encapsulamento.
~		Atributos e métodos são acessíveis somente nos métodos das classes que pertencem ao pacote em que foram criados.
#	**protected**	Atributos e métodos são acessíveis nos métodos da própria classe e suas subclasses.
+	**public**	Atributos e métodos são acessíveis em todos os métodos de todas as classes. Este é o nível menos rígido de encapsulamento.

Métodos

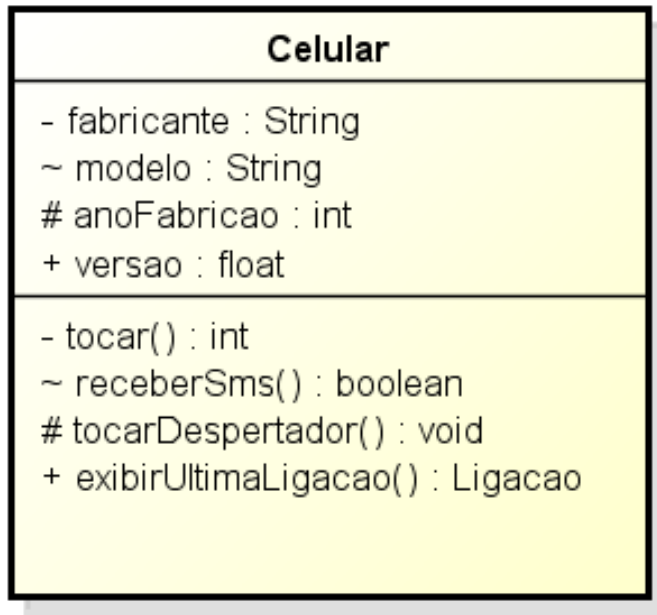
Os modificadores de acesso podem ser representados no diagrama de classes através dos símbolos:

- (private)
~ (default)
# (protected)
+ (public)



Modificadores de acesso

```
public class Celular {

    private String fabricante;
    String modelo;
    protected int anoFabricacao;
    public float versao;


    private int tocar(){ ... }

    boolean receberSms(){ ... }

    protected void tocarDespertador(){
        ... }

    public Ligacao exibirUltimaLigacao()
        { ... }
}
```



Sobrecarga de métodos

A **sobrecarga** é uma característica da orientação a objetos que permite a criação de dois ou mais métodos dentro da mesma classe com diferenças nos tipos e ordem dos parâmetros.

Vamos ilustrar esse conceito fazendo alguns exemplos.

Sobrecarga - Exemplos

```
1  public class Teste {
2
3      public int soma(int x, int y) {...}
4
5      //ok - tipos dos parametros diferentes
6      public double soma(double x, double y) {...}
7
8      //ok - tipos dos parametros diferentes
9      public double soma(int x, double y) {...}
10
11     //ok - numero de parametros diferentes
12     public int soma(int a, int b, int c) {...}
13
14     //nao ok - nome dos parametros nao implica em assinatura
           diferente
15     public int soma(int a, int b) {...}
16
17     //nao ok - mudanca no retorno do metodo nao implica em
           assinatura diferente
18     public void soma(int x, int y) {...}
19 }
```

Operador de autoreferência: **this**

Algumas vezes sofro de criatividade para nomear parâmetros de métodos, por exemplo, no método `setIdade` da classe Pessoa, usamos o parâmetro escrito em inglês (age) seria ótimo usar `idade` A solução seria usar o operador de autoreferência `this`, vejamos como fica o método:

```
1     public void setIdade(int idade) {
2         this.idade = idade;
3     }
```

- podemos considerar que o `this` representa o próprio objeto
- usamos ele para referenciar um atributo de forma não ambígua
- ou seja, na linha 2 do lado esquerdo, o `this.idade` representa o atributo do objeto

this usado em método ou construtor

- usamos o **this** na sobrecarga de construtores ou métodos
- dentro de um construtor, quando queremos chamar um outro construtor, usamos apenas this com os parâmetros do construtor que queremos invocar
- dentro de um método, funciona da mesma forma, porém, adicionamos o nome do método ao this.

Construtores: Exemplos

```
1   public class Carro {
2
3       //considere atributos declarados
4
5       public Carro(String modelo) {
6           this.modelo = modelo;
7       }
8
9       public Carro(String modelo, String montadora) {
10          this(modelo);
11          this.montadora = montadora;
12      }
13
14      public Carro(String modelo, String montadora, int km)
            {
15          this(modelo, montadora);
16          this.quilometragem = km;
17      }
18  }
```

Encapsulamento

Consiste em **proteger** os atributos de uma classe a fim de evitar algum uso indevido. Apenas os métodos da própria classe podem alterar ou recuperar o valor de um atributo!

- para limitar ou controlar o conteúdo de um atributo, métodos devem ser utilizados para colocar ou alterar valores dos atributos de um objeto;
- Para limitar o acesso a um método, métodos devem ser utilizados para acessar o método com visibilidade restrita;
- O uso de atributos diretamente pelos clientes de uma classe é desencorajado:
- Quaisquer mudanças na estrutura interna da classe acarretariam em mudanças nos clientes (alto acoplamento);
- Dependendo da visibilidade, o acesso aos atributos não podem ser feito diretamente;

Encapsulamento

- Esconde os detalhes da implementação de uma classe;
- Força o usuário a usar um método para acesso aos dados;
- Permite definir o modo de acesso aos dados: Leitura, Escrita ou Leitura/Escrita;
- Proteger os dados que estão dentro dos objetos, evitando assim que os mesmos sejam alterados de forma errada;
- Por exemplo, uma classe Pessoa com atribuito inteiro idade. Teoricamente, poderíamos colocar um valor negativo e "quebrar"o objeto;
- ou pior, podemos mudar a classe Pessoa e substituir o atributo idade por dataNascimento

Modificador static

- O modificador static pode ser utilizado na declaração de atributos e métodos.
- Pode ser utilizado em classes, mas não é comum devido a aplicação (classes internas).
- Sintaxe para atributos:

```
1   <visibilidade> static <tipo> <nomeDoAtributo>;
```

- Sintaxe para métodos:

```
1   <visibilidade> static <tipo> <nomeDoMetodo>(<param>) {
2       ....
3   }
```

Modificador static em atributos

- Utilizado quando o atributo está logicamente associado a classe e não os objetos. Por exemplo: o atributo rm de um aluno não pode ser estático porque cada aluno tem um rm único (cada objeto aluno deve ter uma cópia do atributo).
- O atributo para armazenar o valor da passagem do metrô pode ser estático porque o valor padrão é o mesmo para todos os objetos.
- Outro exemplo pode ser a taxa selic para uma classe que representa os vários papeis do tesouro direto ou a cotação do dólar do dia
- Uma atributo estático é instanciada apenas uma vez por classe e não por objeto. Você não precisa de um objeto para acessar esse atributo.

Exemplo de static em atributos

```
1  public class Conta {
2
3      private String cliente;
4      private double saldo;
5      static double limiteCredito = 3000;
6
7
8      public static void main(String[] args)
           {
9          Conta c = new Conta();
10         Conta.limiteCredito = 3500;
11
12         //ou c.limiteCredito = 3500;
13         //apesar de funcionar nao e o
14         //mais correto
15
16         System.out.println(c.limiteCredito);
17   }
18 }
```

Quando a classe Conta é carregada na memória, todas as suas variáveis static são inicializadas (já estão prontas para uso, sem que a classe seja instanciada)

A variável static é acessada usando o nome da classe. Não há a necessidade de instanciar a classe. Todos objetos passarão a ter acesso a esse novo valor

Modificador static em métodos

- O modificador *static* é sempre utilizado quando um método terá a sua execução independente dos atributos de um objeto, ou seja, são métodos e variáveis que não estão logicamente associados aos objetos que foram instanciados.
- Métodos estáticos servem para trabalhar com variáveis definidas como estáticas.
- Definir classes cuja finalidade é reunir métodos globais, que devam estar disponíveis para todos os outros objetos da aplicação
- Por exemplo, a classe `Math` do Java

Exemplo de static em métodos

```
1  import java.util.LocalDate;
2
3  public class Util {
4
5      public static double calculaIdade(LocalDate nascimento) {
6          //Pega a data de agora e realiza as contas para
               calcular
7          //a idade de algo. Para pegar a Data de agora,
               pesquise
8          //sobre a classe Calendar.
9
10         //Observe que, se uma pessoa possui 38 anos e seis
               meses,
11         //seu metodo devera retornar 38.5
12     }
13 }
```

Procure na API do Java a documentação da classe LocalDate.

Exemplo de static em métodos

```
1  public class Conta {
2
3   private String dono;
4   private double saldo;
5   static double limiteCredito = 3000;
6
7   public void aumentaCredito() {
8      limiteCredito = limiteCredito * 1.2;
9   }
10
11  public static void main(String[] args) {
12     Conta c = new Conta();
13     Conta.aumentaCredito();
14
15     //ou c.aumentaCredito();
16     //novamente, mesmo argumento
17
18     System.out.println(c.limiteCredito);
19  }
20 }
```