**Lista de revisão – Tópicos 1**

**Nome:** Silvia Rosa Curto

**Orientação a objetos**

1. **O que é classe?**

Uma classe é uma estrutura que abstrai um conjunto de objetos com características similares.

1. **O que é objeto?**

Objeto é uma instância da classe. As características do objeto são atributos e seus comportamentos são definidos pelo método.

1. **O que é encapsulamento?**

Encapsulamento vem de encapsular, proteger (proteger “as partes íntimas” do código), significa separar o programa em partes, o mais isolado possível.

A programação orientada a objeto pode encapsular (ou seja, empacotar) atributos e operações (métodos) em objetos. Os objetos podem ter a propriedade de ocultamento das informações.

A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações.

1. **O que é um construtor?**

Cada classe que você declara pode fornecer um método especial chamado construtor que pode ser utilizado para inicializar um objeto de uma classe. Construtores são métodos especiais chamados pelo sistema no momento da criação de um objeto. Eles não possuem valor de retorno, porque você não pode chamar um construtor para um objeto, você só usa o construtor no momento da inicialização do objeto. Construtores representam uma oportunidade de inicializar seus dados de forma organizada, imagine se você esquece de inicializar corretamente ou o faz duas vezes, etc.

Um construtor tem sempre o mesmo nome da classe ao qual pertence.

Por padrão, o compilador fornece um construtor padrão sem parâmetros, em qualquer classe que não inclui explicitamente um construtor.

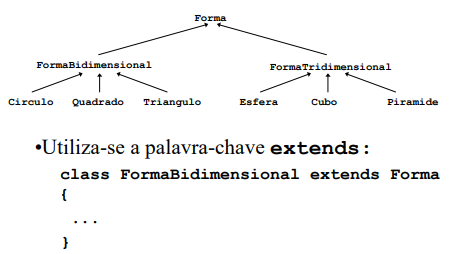
1. **O que é herança? Cite um exemplo de seu uso**

Herança é uma forma de reutilização de software em que uma nova classe é criada, absorvendo membros de uma classe existente e aprimoradas com capacidades novas ou modificadas. Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos. O conceito de herança de várias classes é conhecido como herançamúltipla. A classe existente, denominada superclasse. A nova classe criada a partir da superclasse, chamamos de subclasse. Há um relacionamento entre as classes do tipo “É um”

Exemplo 1:

* Um retângulo é um quadrilátero
* Retângulo é um tipo específico de quadrilátero
* Quadrilátero é a superclasse, retângulo é a subclasse
* Não se pode dizer que um quadrilátero é um retângulo

Exemplo 2:



1. **O que é composição? Cite um exemplo de seu uso**

Quando uma instância da classe existente é usada como componente da outra classe.

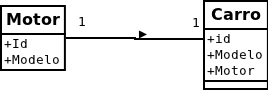
Estende uma classe e delega o trabalho para o objeto desta classe. Uma instância da classe existente é usada como componente da outra classe. Estamos lidando com um relacionamento tem-um.

Usando composição

1. Os objetos que foram instanciados e estão contidos na classe que os instanciou são acessados somente através de sua interface;
2. A composição pode ser definida dinamicamente em tempo de execução pela obtenção de referência de objetos a objetos de do mesmo tipo;
3. A composição apresenta uma menor dependência de implementações;
4. Na composição temos cada classe focada em apenas uma tarefa (princípio SRP);
5. Na composição temos um bom encapsulamento visto que os detalhes internos dos objetos instanciados não são visíveis;

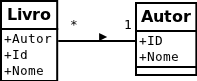
**Exemplo 1:**

Existem duas classes distintas a primeira é a classe Carro e a segunda é a classe Motor. Toda classe Carro tem uma classe Motor, portanto há uma relação entre Carro e Motor de composição, pois o motor está contido na classe Carro.



**Exemplo 2:**

O Livro e o Autor são duas classes distintas, onde cada objeto da classe Livro possui um objeto da classe Autor. Neste caso existirá uma relação de composição, pois a classe Autor faz parte da classe Livro.



1. **No vídeo “ Orientação a Objetos – o que normalmente não se fala por Rinaldi**

**Fonseca”** [**(https://www.youtube.com/watch?v=58wD4rrv5CU ),**](file:///C:\Users\Aluno\Downloads\(https:\www.youtube.com\watch%3fv=58wD4rrv5CU%20),) **o palestrante cita o princípio de SOLID. O que é este princípio? Descreva com suas palavras**

SOLID são princípios básicos para uma boa programação para se ter um código sólido e organizado. Eles devem ser seguidos para que seu código seja de fácil manutenção, adaptação e alterações. Seja testável e de fácil entendimento, forneça o máximo de reaproveitamento e de tempo possível de utilização.

Os princípios são:

* **Single Responsibility Principle (SRP), ou, Princípio da Responsabilidade Única.** Esse princípio diz que as classes devem ser coesas, ou seja, terem uma única responsabilidade. Classes assim tendem a ser mais reutilizáveis, mais simples, e propagam menos mudanças para o resto do sistema.
* **Open Closed Principle (OCP), ou Princípio do Aberto Fechado.** Diz que as classes devem poder ter seu comportamento facilmente estendidas quando necessário, por meio de herança, interface e composição. Ao mesmo tempo, não deve ser necessário abrir a própria classe para realizar pequenas mudanças. No fim, o princípio diz que devemos ter boas abstrações espalhadas pelo sistema.
* **Liskov Substitution Principle (LSP), ou Príncipio da Substituição de Liskov.** Esse princípio diz que precisamos ter cuidado para usar herança. Herança é um mecanismo poderoso, mas [deve ser usado com parcimônia](https://blog.caelum.com.br/como-nao-aprender-orientacao-a-objetos-heranca/), evitando os casos de Gato-estende-Cachorro, apenas por possuírem algo em comum.
* **Interface Segregation Principle (ISP), ou Princípio da Segregação de Interfaces.** Esse princípio diz que nossos módulos devem ser enxutos, ou seja, devem ter poucos comportamentos. Interfaces que tem muitos comportamentos geralmente acabam se espalhando por todo o sistema, dificultando manutenção.
* **Dependency Inversion Principle (DIP), ou Princípio da Inversão de Dependências.** Esse princípio diz que devemos sempre depender de abstrações, afinal abstrações mudam menos e facilitam a mudança de comportamento e as futuras evoluções do código.

Fonte: <https://blog.caelum.com.br/principios-do-codigo-solido-na-orientacao-a-objetos/>

Devemos pensar de maneira diferente => as regras de negócios são objetos.

Ao invés de pensar qual objeto se quer criar, deve-se pensar qual a mensagem eu quero mandar e quem deveria responder essa mensagem. Entender com clareza os fundamentos mindset.

**Java (pesquisar na documentação)**

1. **Quais são os tipos primitivos e tipos por referência do java?**

Os tipos do Java são divididos em tipos primitivos e tipo por referência.

Os tipos primitivos são: boolean, byte, char, short, int, long, float, double.

Todos os tipos não primitivos são tipos por referência, portanto, são as classes, que especificam os tipos de objeto, são tipos por referência.

1. **O que é um pacote?**

Um pacote ou package na tecnologia Java nada mais é do que um conjunto de classes localizadas na mesma estrutura hierárquica de diretórios. Usualmente, são colocadas em um package classes relacionadas, construídas com um propósito comum para promover a reutilização de código; assim, sobre certos aspectos, os packages reproduzem a ideia das bibliotecas de código (libraries e units), de outras linguagens de programação.

O próprio código base da tecnologia Java está todo estruturado em pacotes, como pode ser observado na especificação da API (Application Programming Interface, ou Interface de Programação de Aplicações) da plataforma Java SE

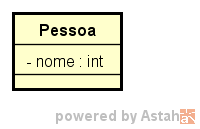
Para indicar que uma classe Java pertence a um determinado pacote, a primeira linha de código deve ser a declaração de pacote implementada através da diretiva package, seguida do nome do pacote.

1. **Quais os modificadores de acesso padrão do Java? Explique-os.**

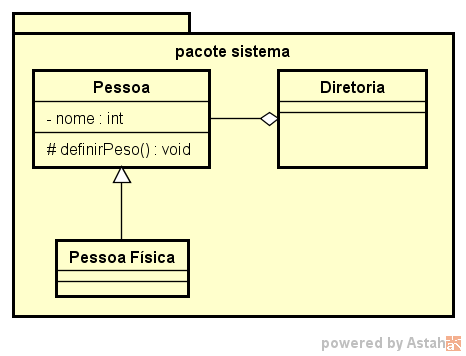
Modificadores de acesso são palavras-chave que garantem níveis de acesso aos atributos, métodos e classes.

Em Java, temos os seguintes modificadores de acesso:

* **Private**É o modificador de acesso mais restritivo que existe. Atributos e métodos declarados como *private* são acessíveis somente pela classe que os declara.   
  Em **UML**, é definido pelo símbolo “***-***”.  
  Métodos e atributos com o modificador *private***não são herdados.**

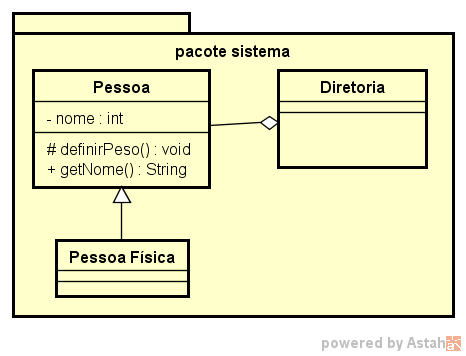
Exemplo de classe com o atributo nome usando o modificador private.

* **Protected**  
  É um modificador de acesso um pouco mais permissivo que o *private.*Atributos e métodos declarados como *protected* são acessíveis pela classe que os declara, suas *subclasses* em outros pacotes e outras classes dentro do mesmo pacote.  
  Em **UML**, é definido pelo símbolo “**#**”.  
  Métodos e atributos declarados com modificador *protected* numa superclasse devem ser definidos como *protected* ou *public* em suas *subclasses* e nunca *private.*



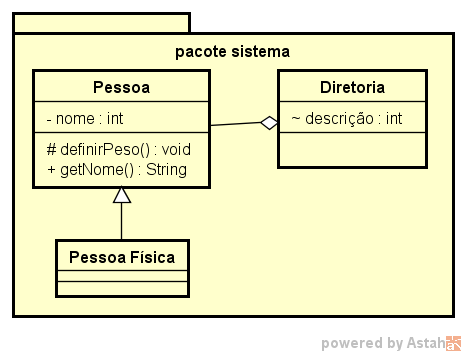
Neste exemplo o método definirPeso( ) é visível na subclasse “Pessoa Física” e na classe “Diretoria” que está no mesmo pacote.

* **Public**Modificador de acesso mais permissivo que existe. Atributos, métodos e classes declarados como *public*são acessíveis por qualquer classe do *Java.*Em **UML**, é definido pelo símbolo “**+**”.  
  Todos os métodos e atributos declarados como *public* são herdados pelas *subclasses*.  
  Métodos e atributos declarados como *public*devem se manter *public*em todas as *subclasses*.



Neste exemplo, o método getNome() é visível por qualquer outra classe.

* **Default**Modificador de acesso padrão, usado quando nenhum for definido. Neste caso os atributos, métodos e classes são visíveis por todas as classes dentro do mesmo pacote.  
  Em **UML**, é definido pelo símbolo “**~**” (*package visibility*).



Neste exemplo, o atributo descrição da classe “DIretoria” é visível por todas as classes dentro do mesmo pacote.

1. **Quais os métodos da classe LinkedList?**

public LinkedList() É um método que constrói uma lista vazia.

LinkedList usa uma lista-ligada para implementar List.

**Module** [java.base](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/module-summary.html)

**Package** [java.util](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/package-summary.html)

## Class LinkedList<E>

Parâmetro E – tipos de elementos mantidos nesta coleção.

* [java.lang.Object](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Object.html)
* [java.util.AbstractCollection](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/AbstractCollection.html)<E>
* [java.util.AbstractList](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/AbstractList.html)<E>
* [java.util.AbstractSequentialList](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/AbstractSequentialList.html)<E>
* java.util.LinkedList<E>

Todas as interfaces implementadas Interfaces:

[Serializable](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/io/Serializable.html), [Cloneable](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Cloneable.html), [Iterable](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Iterable.html)<E>, [Collection](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Collection.html)<E>, [Deque](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Deque.html)<E>, [List](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/List.html)<E>,

[Queue](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Queue.html)<E>

1. **Qual o papel do pacote java.io?**

Fornece entrada e saída do sistema por meio de fluxos de dados, serialização e sistema de arquivos. Salvo indicação em contrário, passa um argumento nulo para um construtor ou método em qualquer classe ou interface, neste pacote fará com que um NullPointerException seja lançado.

A linguagem Java utiliza classes e instâncias delas (objetos) para lidar com a saída e entrada de dados (input/output, abreviado: io). O java trata a entrada e saída como fluxos de dados (Streams), que você tem pleno controle sobre eles. As classes ligadas a io estão nos pacotes java.io e java.nio (NEW io).

1. **O que é uma anotação?**

Anotação é um recurso usado para anotar classes, campos e métodos, de tal maneira que essas marcações podem ser tratadas pelo compilador, ferramentas de desenvolvimento e bibliotecas.

Fornece suporte de biblioteca para o recurso de anotação da linguagem de programação Java.

Interface comum estendida para todos os tipos de anotação.

Um objeto Anotação é usado como um wrapper para um valor de atributo de texto, se o atributo tiver características de anotação.

public class **Annotation**

extends [Object](https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Object.html)

1. **Quais os dois principais pacotes no Java para a criação de interface gráficas?**

Interface Gráfica com Usuário (Graphical User Interface – GUI), é onde os resultados são apresentados em modo gráfico. Os componentes GUI Swing estão dentro do pacote javax.swing que são utilizados para construir as interfaces gráficas. Alguns componentes não são do tipo GUI Swing e sim componentes AWT

Aplicações gráficas são criadas através da utilização de componentes gráficos e recursos associados, que em Java estão agrupados em dois grandes pacotes, java.awt e javax.swing.

AWT é o Abstract Windowing Toolkit, sendo definido através das classes do pacote java.awt e seus subpacotes, tais como java.awt.event e java.awt.color. As classes desses pacotes agrupam as funcionalidades gráficas que estão presentes desde as primeiras versões de Java, que operam tendo por base as funcionalidades do gerenciador de janelas e bibliotecas nativas do sistema onde a aplicação é executada.

Já o framework Swing, associado ao pacote javax.swing e seus subpacotes, é uma extensão definida a partir de AWT que define componentes gráficos que utilizam exclusivamente Java (lightweight components), com funcionalidades e aparência independentes do sistema onde a aplicação é executada. Na plataforma Java 2, Swing passou a fazer parte da distribuição padrão.