Fatec Araras

Centro Paula Souza

GOVERNO DO ESTADO SÃO PAULO
Secretaria de Desenvolvimento Econômico, Ciência, Tecnologia e Inovação

Tópicos Especiais Sistemas para Internet II

Ariosmar F. Pegoraro

Lista de Revisão Tópicos I

1. O que é classe?

   É uma estrutura que abstrai um conjunto de objetos com características similares. (*Prof. Me. Orlando Saraiva Jr)*

2. O que é objeto?

   É uma instância da classe. (*Prof. Me. Orlando Saraiva Jr)*

3. O que é encapsulamento?

   É uma codificação que visa proteger partes do código, usado em linguagens de orientação a objetos, podem encapsular atributos e métodos. (*Prof. Me. Orlando Saraiva Jr),* são ações que visam proteger os membros da classe do "mundo externo". (*Prof. Marco Aurélio Regis)*

4. O que é um construtor?

   Os construtores são métodos que facilitam aos programas a inicialização dos dados-membros da classe. Possui algumas características, tal como, um construtor tem o mesmo nome da classe, uma classe pode ter mais de um construtor, um construtor pode ter zero, um ou mais parâmetros e um construtor não tem valor de retorno. (*Prof. Marco Aurélio Regis)*

5. O que é herança?

   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. A classe existente, denominados superclasse. A nova classe criada a partir da superclasse, chamamos de subclasse. Há um relacionamento entre as classe do tipo "É um". (*Prof. Me. Orlando Saraiva Jr)*
   Exemplo:

```
import java.util.Date;
public class Pessoa {
	public String nome;
	public String cpf;
	public Date data_nascimento;

	public Pessoa(String _nome, String _cpf, Date _data) {
	this.nome = _nome;
	this.cpf = _cpf;
	this.data_nascimento = _data;
	}
}
```

   No trecho do código acima, temos a classe Pessoa e como atributos, nome, cpf e data nascimento.

Por exemplo, se fossemos precisar criar classes para professores, alunos e funcionários, entende-se que todos tem cpf, nome e data de nascimento, portanto podemos criar subclasses de Pessoa para representá-los.
Em Java criamos classes derivadas usando a palavra extends seguida do nome da superclasse.

```
public class Aluno extends Pessoa {
    public Aluno(String _nome, String _cpf, Date _data) {
        super(_nome, _cpf, _data);
    }
    public String matricula;
}
public class Professor extends Pessoa {
    public Professor(String _nome, String _cpf, Date _data) {
        super(_nome, _cpf, _data);
    }
    public double salario;
    public String disciplina;
}
public class Funcionario extends Pessoa {
    public Funcionario(String _nome, String _cpf, Date _data) {
        super(_nome, _cpf, _data);
    }
    public double salario;
    public Date data_admissao;
    public String cargo;
}
```

Fonte: https://www.devmedia.com.br/entendendo-e-aplicando-heranca-em-java/24544

6. O que é composição?

Composição é quando uma instância da classe existente é usada como componente da outra classe, nesse caso há um relacionamento tem-um.(*Prof. Me. Orlando Saraiva Jr)*
Um exemplo disso, imaginemos duas classes A e B, com seus respectivos atributos, a pergunta que fazemos para saber se há composição entre elas é: A classe A está contida na classe B? Se a resposta for sim, então existe uma relação de composição entre elas.
Ex.: Classe Livro e Classe Autor
Um livro possui autor? Sim, então podemos usar composição.

7. O que é o princípio de SOLID?

É uma palavra originada das iniciais dos 5 princípios da programação orientada a objetos, cuja teoria é de Robert C. Martin. O autor da criação dessa palavra foi Michael Feathers.
No vídeo o palestrante Rinaldi Fonseca utiliza essa palavra no contexto da explanação sobre open / closed, que ele menciona que o programador pode adicionar novos comportamentos ao sistema.
Os 5 princípios da programação orientada a objetos são:

Single Responsibility Principle (Princípio da Responsabilidade Única)
Open/Closed Principle (Princípio do Aberto/Fechado)
Liskov Substitution Principle (Princípio da Substituição de Liskov)
Interface Segregation Principle (Princípio da Segregação de Interfaces)
Dependency Inversion Principle (Princípio da Inversão de Dependências)

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

   Os oitos tipos primitivos da linguagem Java são:
   Byte, short, int, long, float, double, boolean e char.
   Basicamente os tipos não primitivos são tipos por referência. Os 4 tipos por referencia comumente usados são:
   class types, interface types, type variables, and array types.
   Fonte: https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.3

2. O que é um pacote ?

   Conceitualmente podemos pensar que pacotes são semelhantes as pastas que criamos no computador, que servem para organizar, relacionar e implementar as classes de bibliotecas, etc. A plataforma Java fornece uma vasta quantidade de pacotes que podem ser usadas nas aplicações.
   Fonte: https://docs.oracle.com/javase/tutorial/java/concepts/package.html

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

   Public, private e protected.
   Quando uma classe é declarada com modificador public, ela pode ser visível para todas as classes e partes do código.
   O modificador de acesso privado especifica que o membro só pode ser acessado na sua própria classe.
   O modificador protected especifica que o membro só pode ser acessado em seu próprio pacote (como no pacote private) e, além disso, por uma subclasse de sua classe em outro pacote.
   A tabela abaixo mostra o acesso permitido por cada modificador:

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

   Fonte: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

4. Quais os métodos da classe LinkedList?

• Method Summary

Methods

Modifier and Type	Method and Description
boolean	**add**(**E** e) Appends the specified element to the end of this list.
void	**add**(int index, **E** element) Inserts the specified element at the specified position in this list.

boolean	**addAll**(**Collection**<? extends **E**> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	**addAll**(int index, **Collection**<? extends **E**> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**addFirst**(**E** e) Inserts the specified element at the beginning of this list.
void	**addLast**(**E** e) Appends the specified element to the end of this list.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this LinkedList.
boolean	**contains**(**Object** o) Returns true if this list contains the specified element.
Iterator<**E**>	**descendingIterator**() Returns an iterator over the elements in this deque in reverse sequential order.
E	**element**() Retrieves, but does not remove, the head (first element) of this list.
E	**get**(int index) Returns the element at the specified position in this list.
E	**getFirst**() Returns the first element in this list.
E	**getLast**() Returns the last element in this list.
int	**indexOf**(**Object** o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
int	**lastIndexOf**(**Object** o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<**E**>	**listIterator**(int index) Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
boolean	**offer**(**E** e) Adds the specified element as the tail (last element) of this list.
boolean	**offerFirst**(**E** e) Inserts the specified element at the front of this list.
boolean	**offerLast**(**E** e) Inserts the specified element at the end of this list.
E	**peek**() Retrieves, but does not remove, the head (first element) of this list.
E	**peekFirst**() Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
E	**peekLast**() Retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
E	**poll**() Retrieves and removes the head (first element) of this list.
E	**pollFirst**() Retrieves and removes the first element of this list, or returns null if this list is empty.

E	**pollLast**() Retrieves and removes the last element of this list, or returns null if this list is empty.
E	**pop**() Pops an element from the stack represented by this list.
void	**push**(**E** e) Pushes an element onto the stack represented by this list.
E	**remove**() Retrieves and removes the head (first element) of this list.
E	**remove**(int index) Removes the element at the specified position in this list.
boolean	**remove**(**Object** o) Removes the first occurrence of the specified element from this list, if it is present.
E	**First**() s and returns the first element from this list.
boolean	**FirstOccurrence**(**Object** o) s the first occurrence of the specified element in this list (when traversing the list from head to tail).
E	**Last**() s and returns the last element from this list.
boolean	**LastOccurrence**(**Object** o) s the last occurrence of the specified element in this list (when traversing the list from head to tail).
E	ıdex, **E** element) s the element at the specified position in this list with the specified element.
int	 the number of elements in this list.
Object[]	() an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	(T[] a) an array containing all of the elements in this list in proper sequence (from first to last element); the runtime he returned hat of the specified array.

Fonte: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

5. Qual o papel do pacote java.io ?

Este pacote fornece para o sistema input e output através do fluxo de dados, serialization e o file system, para evitar um argumento nulo para um constructor ou método e assim evitar um erro de NullPointerException.
Fonte: https://docs.oracle.com/javase/7/docs/api/

6. O que é uma anotação ?

   É um recurso que pode ser usado para anotar, isto é marcar, classes, campos e métodos e essas marcações podem ser lidas pelo compilador, ferramentas de desenvolvimento e bibliotecas.
   Os tipos de anotação são:
   @Deprecated, @Override, @SuppressWarning, @SafeVarargs, @FunctionalInterface, @Retention, @Documented, @Target, @Inherited e @Repeatable.

   “Annotations have a number of uses, among them:

- **Information for the compiler** — Annotations can be used by the compiler to detect errors or suppress warnings.
- **Compile-time and deployment-time processing** — Software tools can process annotation information to generate code, XML files, and so forth.
- **Runtime processing** — Some annotations are available to be examined at runtime."

  Fonte: https://docs.oracle.com/javase/tutorial/java/annotations/

7. Quais os dois principais pacotes no Java para criação de interface gráficas ?

   java.awt e `javax.swing`