OO - Lorenzon

Abstração, classe e objeto.

O primeiro pilar da OO, a abstração é a capacidade da linguagem de programação possuir suporte à criação de classes que definem (tipificam) um objeto.

A abstração se caracteriza pela classe que é uma estrutura de programação que descreve uma abstração através de suas propriedades (atributos) e comportamento (métodos).

O objeto é uma instância da classe em memória. Possui as características do molde, mas seus valores individuais que denota o estado interno do objeto.

Atributos

São as características da abstração. Denotadas por variáveis primitivas ou do tipo Classes. São definidas, criadas na Classe. A informação de cada variável, seu valor, compõe parte dos valores internos do objeto. O conjunto de valores dos atributos para cada objeto, baseado nos atributos da classe, refere-se ao estado interno do objeto.

Métodos

É o comportamento da classe. Construído como funções membros da classe. Podem ou não possuir assinatura. Podem ou não retornar valores. O comportamento quando acionado, internamente na classe ou fora dela, índice sobre os valores dos atributos da classe: recuperando os valores ou atribuindo novos valores.

Instanciação, construção do objeto

Partindo da classe, existirá um objeto pela instanciação da classe em memória. Para instanciar uma classe, criando um objeto a partir dela, é preciso do construtor da classe.

Na linguagem Java toda classe possui, se não criado outro, um construtor padrão. Este construtor é o método com o nome da classe seguido de (), ou seja, assinatura vazia.

A ideia do construtor é instanciar o objeto em memória e dar valores ao seu estado interno inicial, portanto, os valores de cada atributo serão definidos neste momento. Valores que podem vir externamente serão colocados na assinatura do método construtor, reescrevendo então o construtor padrão da classe.

A classe pode ter vários construtores, mantendo o nome do método e diferenciando as assinaturas destes.

O construtor é publico e não possui nenhum retorno, nem mesmo void.

Um exemplo para criação de um objeto a partir de uma classe com uma referência em Java.

```
NomeClasse nomeObjeto;
nomeObjeto = new NomeClasse ();
```

A instanciação é a invocação de um construtor da classe. A instanciação se faz com o uso de new, que devolve um endereço de memória para ser alocado em uma referência da classe.

Resumo do código exemplificado:

nomeObjeto	Nome do objeto, um "ponteiro" ou referência a ele.
=	Operador de atribuição.
new	Uma palavra reservada que invoca a construção de Objeto.
NomeClasse	O construtor de uma classe (padrão);
()	Uma indicação de assinatura (vazia) de método construtor;
;	Encerramento de uma instrução.

Na instanciação, a chamada ao construtor da classe inicializa o objeto em memória.

Exemplos:

```
public class A {
   int x;
   //reescrita do construtor padrão por explicitação
   public A(){
       x = 0;
   }
   //alternativa de construtor recebendo valores externos
   public A(int x){
       this.x = x;
   }
}
```

A referência this

`this` é uma palavra usada em Java para referenciar um elemento, atributo da classe. Refere-se ao valor do estado interno para aquele objeto.

Evita conflito de nomes de assinaturas e atributos. Pode-se ser evitado se não possuir mesma nomenclatura.

`this` também invoca um construtor dentro de outro. Fazendo uma cascata de código quando há vários construtores dentro de uma classe.

Java tem capacidade de saber quem é o método `this` pela sua passagem de parâmetros.

Do exemplo anterior, evoluindo.

```
public class A {
   int x, y;


   public A(){
       x = 0;
       y = 0;
   }
   //chamando um construtor dentro de outro
   public A(int x){
       this();
       this.x = x;
   }
}
```

A notação ponto

Para acessar qualquer elemento de uma classe, seja atributo ou método, partindo de uma classe ou referência para a mesma (objeto), usa-se a notação referencia.elemento, onde elemento é um atributo ou um método.

```
        // Define "classeObjeto" como uma referência de "Classe"
        Classe classeObjeto = new Classe ();


        // Acessa (chama) o método m1() usando classeObjeto
        classeObjeto.m1();
```

A organização das classes e pacotes

Cada arquivo de código de uma classe deve ser depositado em um arquivo.Java.

Tendo uma classe pública no arquivo, esta deve ser única. O nome do Arquivo.Java deve ser igual ao nome da classe pública nele codificada, seguindo a notação.

Podemos definir pacotes de organização de classes. Esta organização se dará por criação de pastas e subpastas no projeto de seu workspace.

Quando precisamos de uma classe que não esteja no mesmo pacote, mesma pasta, da classe que estamos codificando, precisamos fazer uso de `import` para passar o caminho da classe.

Para organizar as classes usamos a palavra `package` e o nome do pacote que a classe pertence.

Java fornece acesso implícito ao pacote java.lang. Todas as classes deste pacote estão disponíveis pra uso. Exemplo: String e Math – classe para operações de métodos matemáticos.

```
import utilitários.Teclado; //fazendo ligação para usar
package um; //organização desta classe que está sendo criada
public class mesmoNomeArquivo{
      //..
}
```

Associação entre classes

Existem 3 formas de relação entre as classes: **associação**, agregação e composição.

Diferença básica das relações:

- **Associação**: Característica de uma classe TER uma referência de outra em si. Exemplo: Um **Livro** tem um **Autor**;
- **Agregação**: Característica de uma classe TER uma ou mais referências de outra classe em si. Uma **Biblioteca** possui **Livros**. Um Livro tem vários Autores;
- **Composição**: Característica de uma classe obrigatoriamente TER um ou mais referências de outra classe em si. Um **Livro** possui um ou mais **Capítulo**. O Capítulo não tem sentido de existir sem um Livro.

Toda e qualquer relação é definida por uma referência de uma classe em outra classe.

Qualquer associação de mais do que um elemento será um arranjo ou uma coleção de objetos.

```
class Autor {
  String nome;
}

class Livro {
  String titulo;
  Autor autor; //associação com a classe Autor
}
```