MÉTODOS, ATRIBUTOS, CLASSES E MODIFICADORES

SENAI <LAB365>

SUMÁRIO

O QUE SÃO MÉTODOS E FUNÇÕES?

Os métodos são conhecidos como funções, procedimentos ou ações de execução, são pequenas partes de código existentes nas classes e são utilizados para executar algum procedimento, seja ele dar valor a um atributo da classe, realizar algum cálculo ou até mesmo executar uma tarefa complexa.

Sintaxe dos métodos

A sintaxe dos métodos é construída da seguinte maneira:

```
modificador retorno identificador(argumentos) {
 corpo
}
```

A seguir, confira mais sobre modificadores, identificadores, retorno, argumentos e corpo das funções.

Modificadores

Cada método (assim como atributos e classes) possui modificadores de acesso, que indicam sua visibilidade em outras classes. São eles:

Public - O menos restritivo dos modificadores, indica que o método pode ser acessado de forma pública, por qualquer entidade que possa visualizar a classe a que o método pertença, por outras classes derivadas desta e por classes de outros pacotes.

Protected - Mais restritivo que o *public*, indica que o método estará acessível apenas para as classes do mesmo pacote ou através de herança (para suas subclasses).

***Default* (padrão)** - Mais restritivo que o *protected*, também pode ser conhecido como *package*, indica que o método estará acessível apenas para as classes do mesmo pacote. Para que um método seja definido como *default*, não deve ser indicado o modificador de forma explícita, deixando o compilador identificá-lo.

Private - O mais restritivo dos modificadores, indica que o método não estará acessível para outras classes, somente para a sua própria classe.

Você pode conferir pela imagem onde cada modificador se encontra.

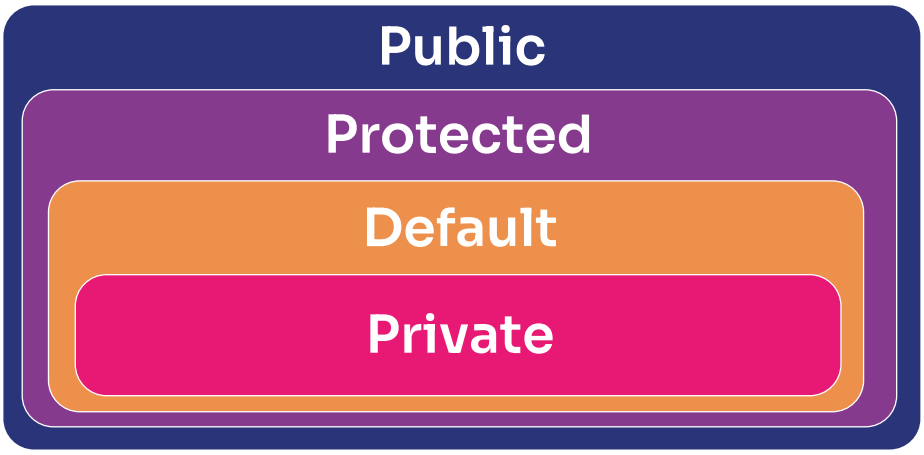

Figura 1 - Modificadores

Fonte: do Autor

Existem também outros modificadores que podem ser utilizados em conjunto com os modificadores descritos acima. São usados para apresentar um comportamento específico do método. Alguns dos mais usados são:

Static - Esse modificador desobriga a criação de uma instância da classe para utilizar o método. Sendo assim, o método pode ser executado sem uma instância da classe. Muito utilizado em atributos para compartilhar a mesma variável entre todas as instâncias da classe.

Abstract - Quando houver um método com este modificador, o método não terá implementação e a classe também precisará ser marcada com o mesmo. Dessa forma a classe não pode ser instanciada e o método precisa ser implementado na classe que herdá-la.

Final - Ao declarar um método com este modificador, a classe que herda fica impedida de sobrepor esse método.

Esses modificadores não serão explorados neste momento, pois o uso baseia-se nos conceitos não abordados de orientação a objetos. Siga estudando agora sobre retornos.

Retornos

Um método contém várias formas de retorno, podendo ser um valor, um objeto ou até mesmo um retorno vazio. Quando o método retorna vazio, utiliza-se a palavra-chave “void”, caso contrário, deve ser indicado o tipo de dado que o método retornará, nesse caso, o método também deverá conter em seu corpo a palavra-chave “return” e na sequência o que representa o tipo de dado indicado no retorno. Por exemplo:

```
// Sem retorno (retorno vazio)
public void setDataAtual() {
    this.dataAtual = new Date();
}

// Com retorno (retorno do tipo “java.util.Date”)
public Date getDataAtual() {
    return this.dataAtual;
}
```

Identificador

O identificador nada mais é do que o nome do método. O nome pode descrever o que o método executa, por exemplo: para um método que salva uma entidade, o nome do método pode ser “salvar”, “guardar” ou até mesmo em inglês “save” (conforme o padrão do projeto). Para Java, existe uma convenção para a nomenclatura dos métodos, abaixo conheça algumas regras da convenção:

- Palavras devem ser separadas por letras maiúsculas e sem espaços entre elas.
- Métodos que inserem valor em atributos iniciam com “set” seguido do nome do atributo. Por exemplo: para o atributo “private String nome” o nome do método fica “setNome”.
- Métodos que leem valores de atributos iniciam com “get” seguido do nome do atributo. Por exemplo: para o atributo “private String nome” o nome do método fica “getNome”.

Para conhecer mais a respeito das convenções da linguagem, acesse o documento oficial *Code Conventions for the Java Programming Language*, disponível no site clicando no link a seguir ou aproximando seu celular ao QR code:

https://www.oracle.com/java/technologies/cc-java-programming-language.html

Argumentos

Os argumentos também são conhecidos como parâmetros e são utilizados pelo método como variáveis locais para executar os procedimentos escritos no corpo do método.

O método pode conter vários argumentos separados por vírgula, como pode não conter argumentos, pois são opcionais. Estes argumentos podem ser variáveis ou valores passados diretamente no momento em que o método é chamado. Por exemplo:

```
// Sem argumento
public void escreverConsoleDataAtual() {
     System.out.println(new Date());
}

// Com argumento do tipo “java.util.Date”
public void escreverConsoleData(Date data) {
     System.out.println(data);
}

// Com argumento e verificação se número é par
public boolean numeroPar(int numero) {
 int mod = numero % 2;
 if(modulo == 0) {
    return true;
 } else {
   return false;
 }
}
```

Corpo

O corpo do método é onde fica a lógica implementada. Nele, podem ser declaradas variáveis locais, chamar outros métodos da sua própria classe ou de outras, manipular seus parâmetros, executar operações matemáticas, percorrer laços de repetição, dentre outros procedimentos necessários para cumprir com o propósito do método e devolver o retorno esperado, caso necessário.

EXEMPLOS NA PRÁTICA

Como você conferiu anteriormente, os modificadores indicam padrões de visibilidade e alguns comportamentos específicos de cada método. A seguir vamos praticar um pouco.

Métodos públicos

Para exemplificar o uso dos métodos públicos, vamos criar uma instância de "PublicPessoa" com 15 anos de idade e incrementar sua idade até que a pessoa atinja a maioridade. Para isso, vamos criar classes somente com métodos públicos, uma utilizaremos para instanciação ("PublicPessoa") e outra para acessarmos os métodos criados ("PublicMain").

A classe "PublicPessoa" conterá atributos com dados públicos de uma pessoa (nome e idade) com métodos que retornem seus conteúdos e um atributo utilizado apenas internamente (maioridade) para comparar se a pessoa já atingiu a maioridade. Seus métodos serão utilizados para retornar seus dados públicos, incrementar sua idade e verificar sua maioridade.

A classe "PublicMain" conterá métodos públicos estáticos. Um desses métodos é o método "main", é por ele que uma aplicação java inicia sua execução. O roteiro será o seguinte:

1. O método "main" invocará o método "utilizandoMetodosPublic".
2. O método "utilizandoMetodosPublic" passará parâmetros para o método "criandoAteMaioridade" e aguardará seu retorno.
3. O método "criandoAteMaioridade" utilizará os parâmetros recebidos para criar uma instância da classe "PublicPessoa":
   3.1. Realizar verificações e manipulações na instância até que esteja preparada para retorná-la.
4. Ao retornar a instância de "PublicPessoa" para o método "utilizandoMetodosPublic", ele escreverá no console informações uma frase utilizando outros métodos da instância retornada.

Em seguida confira o código de: PublicPessoa.java

```
public class PublicPessoa {

	private String nome;
	private Integer idade;
	private Integer maioridade = 18;

	// Construtor padrão
	public PublicPessoa() {
		nome = “Nome (public)”;
		idade = 15;
	}

	// Construtor com parâmetros
	public PublicPessoa(String nome, Integer idade) {
		this.nome = nome;
		this.idade = idade;
	}

	// Método que retorna o nome
	public String getNome() {
		return nome;
	}

	// Método que retorna a idade
	public Integer getIdade() {
		return idade;
	}

	// Método que incrementa a idade
	public void fazAniversario() {
		idade = idade + 1;
		System.out.println(nome + “! Parabéns pelos seus “ +
idade + “ anos de vida!”);
	}
```

```
// Método que verifica a maioridade
public boolean verificaMaioridade() {
    boolean maior = idade >= maioridade;
    if (maior) {
              System.out.println("Com " + idade +
              " anos,já atingiu a maioridade!");
    } else {
              Integer diferenca = (maioridade - idade);
              if (diferenca > 1) {
                  System.out.println("Com " + idade + " anos,
faltam " + diferenca + " anos para a maioridade!");
              } else {
                      System.out.println("Com " + idade +
" anos, falta " + diferenca + " ano para a maioridade!");
              }
         }
         return maior;
     }

}
```

Agora confira o código de: PublicMain.java

```java
public class PublicMain {

	public static void main(String[] args) {
		utilizandoMetodosPublic();
	}

	public static void utilizandoMetodosPublic() {

		System.out.println("\n"); // Quebra de linha
		System.out.println("####################");
		System.out.println("# Métodos públicos #");
		System.out.println("####################");
		System.out.println("\n"); // Quebra de linha

		PublicPessoa ana = criandoAteMaioridade("Ana", 15);

		System.out.println("Agora " + ana.getNome() + " tem "
		+ ana.getIdade() + " anos de idade!");

	}

	public static PublicPessoa criandoAteMaioridade(String nome,
	Integer idade) {

		PublicPessoa pessoa = new PublicPessoa(nome, idade);
		// Instanciando uma pessoa (dados públicos)

		// Armazenando retorno do método em uma variável local
		boolean atingiuMaioridade = pessoa.verificaMaioridade();

		while (!atingiuMaioridade) {
			pessoa.fazAniversario();
			atingiuMaioridade = pessoa.verificaMaioridade();
		}

		return pessoa;
	}

}
```

Ao executar a classe “PublicMain” você terá o seguinte retorno no console:

```
####################
# Métodos públicos #
####################

Com 15 anos, faltam 3 anos para a maioridade!
Ana! Parabéns pelos seus 16 anos de vida!
Com 16 anos, faltam 2 anos para a maioridade!
Ana! Parabéns pelos seus 17 anos de vida!
Com 17 anos, falta 1 ano para a maioridade!
Ana! Parabéns pelos seus 18 anos de vida!
Com 18 anos, já atingiu a maioridade!
Agora Ana tem 18 anos de idade!
```

Todos os métodos “public” estarão visíveis em todas as classes, independente de herança ou de estarem no mesmo pacote, sempre disponíveis para serem utilizados.

Métodos protegidos

Para exemplificar o uso dos métodos protegidos, vamos duplicar a classe “PublicPessoa”, renomeá-la para “ProtectedPessoa” e movê-la para um pacote “protecteds” (passaremos a utilizar este pacote neste exemplo), altere também o modificador do método “fazAniversario” de “public” para “protected”. Dessa forma, terão acesso a esse método, somente as classes do mesmo pacote ou classes que estendem “ProtectedPessoa”. A classe ficará da seguinte forma:

```java
public class ProtectedPessoa {

	private String nome;
	private Integer idade;
	private Integer maioridade = 18;

	// Construtor padrão
	public ProtectedPessoa() {
		nome = “Nome (protegido)”;
		idade = 15;
	}

	// Construtor com parâmetros
	public ProtectedPessoa(String nome, Integer idade) {
		this.nome = nome;
		this.idade = idade;
	}

	// Método que retorna o nome
	public String getNome() {
		return nome;
	}

	// Método que retorna a idade
	public Integer getIdade() {
		return idade;
	}

	// Método que incrementa a idade
	protected void fazAniversario() {
		idade = idade + 1;
		System.out.println(nome + “! Parabéns pelos seus “ + idade +
		“ anos de vida!”);
	}
```

```
    // Método que verifica a maioridade
    public boolean verificaMaioridade() {
        boolean maior = idade >= maioridade;
        if (maior) {
            System.out.println("Com " + idade + " anos, já atingiu
            a maioridade!");
        } else {
            Integer diferenca = (maioridade - idade);
            if (diferenca > 1) {
                System.out.println("Com " + idade + " anos,
                faltam " + diferenca + " anos para a maioridade!");
            } else {
                System.out.println("Com " + idade + " anos, falta
                " + diferenca + " ano para a maioridade!");
            }
        }
        return maior;
    }

}
```

Também vamos duplicar a classe “PublicMain”, renomeá-la para “ProtectedMain” e movê-la para um pacote “protecteds”. Em seguida vamos substituir nesta classe as palavras “PublicPessoa”, “ana”, “Métodos públicos”, “utilizandoMetodosProtected” respectivamente para “ProtectedPessoa”, “bento”, “Métodos protegidos”, “utilizandoMetodosProtected”. A classe ficará como apresentado a seguir:

```java
public class ProtectedMain {

	public static void main(String[] args) {
		utilizandoMetodosProtected();
	}

	public static void utilizandoMetodosProtected() {
		System.out.println("\n"); // Quebra de linha
		System.out.println("######################");
		System.out.println("# Métodos protegidos #");
		System.out.println("######################");
		System.out.println("\n"); // Quebra de linha

		ProtectedPessoa bento = criandoAteMaioridade("Bento", 15);

		System.out.println("Agora " + bento.getNome() + " tem "
		+ bento.getIdade() + " anos de idade!");
	}

	public static ProtectedPessoa criandoAteMaioridade(String nome,
Integer idade) {

		ProtectedPessoa pessoa = new ProtectedPessoa(nome, idade);
		// Instanciando uma pessoa (dados públicos)

		// Armazenando retorno do método em uma variável local
		boolean atingiuMaioridade = pessoa.verificaMaioridade();

		while (!atingiuMaioridade) {
			pessoa.fazAniversario();
			atingiuMaioridade = pessoa.verificaMaioridade();
		}

		return pessoa;
	}

}
```

Ao executar a classe “ProtectedMain” o processo será executado com sucesso, com o mesmo resultado de quando executado com a classe “PublicMain”. Pois ambas as classes estão no mesmo pacote.

Ao mover ou copiar a classe “ProtectedMain” para outro pacote, a IDE indicará que o método “fazAniversario” não está visível, como na imagem abaixo. Isso acontece porque a classe “ProtectedPessoa” não está no mesmo pacote e também não há uma classe que a estenda para ser utilizada.

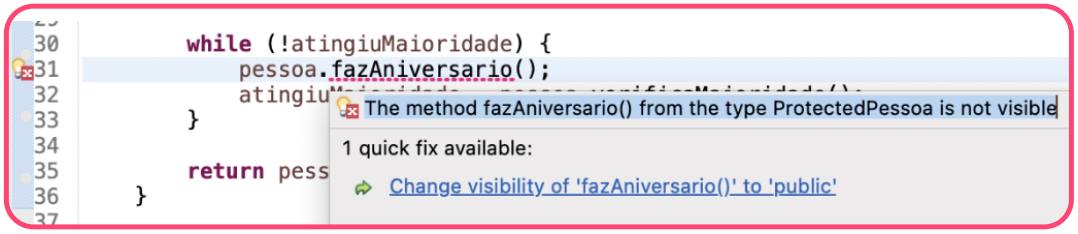

Figura 2 - Métodos protegidos: exemplo de aviso da IDE

Fonte: do Autor (2022)

Dessa maneira, para utilizar este método, podemos criar uma classe que estenda “ProtectedPessoa”, assim podemos sobrepor o método ou criar outro que acesse o “fazAniversario”.

Primeiro, vamos simular esse problema criando um novo pacote “ext” dentro do pacote “protecteds”, copiando a classe “ProtectedMain” para dentro do novo pacote, renomeando-a para “ProtectedExtMain” e, por não estar mais no mesmo pacote, será necessário importar a classe “ProtectedPessoa”. Desta forma, sua IDE estará exibindo o alerta da imagem acima. A classe ficará como expressado a seguir:

```java
public class ProtectedExtMain {

	public static void main(String[] args) {
		utilizandoMetodosProtected();
	}

	public static void utilizandoMetodosProtected() {
		System.out.println("\n"); // Quebra de linha
		System.out.println("##############################");
		System.out.println("# Métodos protegidos (ext) #");
		System.out.println("##############################");
		System.out.println("\n"); // Quebra de linha

		ProtectedPessoaExt claudio = criandoAteMaioridade
		("Claudio", 15);

		System.out.println("Agora " + claudio.getNome() + " tem "
		+ claudio.getIdade() + " anos de idade!");
	}

	public static ProtectedPessoaExt criandoAteMaioridade
	(String nome, Integer idade) {

		ProtectedPessoaExt pessoa = new ProtectedPessoaExt
		(nome, idade);

		// Instanciando uma pessoa (dados públicos)

		// Armazenando retorno do método em uma variável local
		boolean atingiuMaioridade = pessoa.verificaMaioridade();

		while (!atingiuMaioridade) {
			pessoa.fazAniversario();
			atingiuMaioridade = pessoa.verificaMaioridade();
		}

		return pessoa;
	}

}
```

Agora, vamos criar uma classe “ProtectedPessoaExt” e estender “ProtectedPessoa”, criaremos os construtores como os na classe herdada e repassaremos os parâmetros para o construtor da classe pai. Enfim, vamos sobrepor o método “fazAniversario” (utilizaremos a anotação “@Override” no método para indicar a sobreposição) chamando dentro do novo método, o método protegido da classe pai.

Lembre-se de importar a classe “ProtectedPessoa”.

```
public class ProtectedPessoaExt extends ProtectedPessoa {

	public ProtectedPessoaExt() {
		super();
	}

	public ProtectedPessoaExt(String nome, Integer idade) {
		super(nome, idade);
	}

	@Override
	public void fazAniversario() {
		super.fazAniversario();
	}

}
```

Com isso é possível utilizar a nova classe ao invés da “ProtectedPessoa”. Sendo assim, na classe “ProtectedExtMain” vamos alterar o tipo da variável “bento” de “ProtectedPessoa” para “ProtectedPessoaExt”. Vamos também alterar o nome dela de “bento” para “claudio”. A classe ficará como a seguir:

```java
public class ProtectedExtMain {

	public static void main(String[] args) {
		utilizandoMetodosProtected();
	}

	public static void utilizandoMetodosProtected() {
		System.out.println("\n"); // Quebra de linha
		System.out.println("##############################");
		System.out.println("# Métodos protegidos (ext) #");
		System.out.println("##############################");
		System.out.println("\n"); // Quebra de linha

		ProtectedPessoaExt claudio = criandoAteMaioridade
		("Claudio", 15);

		System.out.println("Agora " + claudio.getNome() + " tem "
		+ claudio.getIdade() + " anos de idade!");
	}

	public static ProtectedPessoaExt criandoAteMaioridade
	(String nome, Integer idade) {

		ProtectedPessoaExt pessoa = new ProtectedPessoaExt
		(nome, idade);

		// Instanciando uma pessoa (dados públicos)

		// Armazenando retorno do método em uma variável local
		boolean atingiuMaioridade = pessoa.verificaMaioridade();

		while (!atingiuMaioridade) {
			pessoa.fazAniversario();
			atingiuMaioridade = pessoa.verificaMaioridade();
		}

		return pessoa;
	}

}
```

Nesse momento, ao executar a classe “ProtectedExtMain” você terá o seguinte retorno no console:

```
############################
# Métodos protegidos (ext) #
############################

Com 15 anos, faltam 3 anos para a maioridade!
Claudio! Parabéns pelos seus 16 anos de vida!
Com 16 anos, faltam 2 anos para a maioridade!
Claudio! Parabéns pelos seus 17 anos de vida!
Com 17 anos, falta 1 ano para a maioridade!
Claudio! Parabéns pelos seus 18 anos de vida!
Com 18 anos, já atingiu a maioridade!
Agora Claudio tem 18 anos de idade!
```

Métodos padrões

Métodos “default” (também podem ser conhecidos como “package”) são utilizados para que sejam visíveis somente no mesmo pacote. Para exemplificar este uso, vamos criar um pacote com nome “defaults”, duplicar a classe “ProtectedPessoa”, renomeá-la para “DefaultPessoa” e movê-la para um pacote “defaults” (passaremos a utilizar este pacote neste exemplo), remova também o modificador “protected” do método “fazAniversario”. A classe ficará da seguinte forma:

```java
public class DefaultPessoa {

	private String nome;
	private Integer idade;
	private Integer maioridade = 18;

	// Construtor padrão
	public DefaultPessoa() {
		nome = “Nome (padrão)”;
		idade = 15;
	}

	// Construtor com parâmetros
	public DefaultPessoa(String nome, Integer idade) {
		this.nome = nome;
		this.idade = idade;
	}

	// Método que retorna o nome
	public String getNome() {
		return nome;
	}

	// Método que retorna a idade
	public Integer getIdade() {
		return idade;
	}

	// Método que incrementa a idade
	void fazAniversario() {
		idade = idade + 1;
		System.out.println(nome + “! Parabéns pelos seus “ + idade
		+ “ anos de vida!”);
	}
```

```
        // Método que verifica a maioridade
        public boolean verificaMaioridade() {
              boolean maior = idade >= maioridade;
              if (maior) {
                    System.out.println("Com " + idade + " anos, já atingiu
                    a maioridade!");
              } else {
                    Integer diferenca = (maioridade - idade);
                    if (diferenca > 1) {
                          System.out.println("Com " + idade + " anos,
                       faltam " + diferenca + " anos para a maioridade!");
                    } else {
                          System.out.println("Com " + idade + " anos,
                       falta " + diferenca + " ano para a maioridade!");
                    }
              }
              return maior;
        }

  }
```

Também vamos duplicar a classe "ProtectedMain", renomeá-la para "DefaultMain" e movê-la para um pacote "defaults". Em seguida, substituiremos nesta classe as palavras "ProtectedPessoa", "bento", "Métodos protegidos", "utilizandoMetodosProtected" respectivamente para "DefaultPessoa", "dafine", "Métodos padrões", "utilizandoMetodosDefault". A classe ficará do seguinte modo:

```java
public class DefaultMain {

	public static void main(String[] args) {
		utilizandoMetodosDefault();
	}

	public static void utilizandoMetodosDefault() {
		System.out.println("\n"); // Quebra de linha
		System.out.println("####################");
		System.out.println("# Métodos padrões #");
		System.out.println("####################");
		System.out.println("\n"); // Quebra de linha

		DefaultPessoa dafine = criandoAteMaioridade("Dafine", 15);

		System.out.println("Agora " + dafine.getNome() + " tem "
		+ dafine.getIdade() + " anos de idade!");
	}

	public static DefaultPessoa criandoAteMaioridade(String nome,
Integer idade) {

		DefaultPessoa pessoa = new DefaultPessoa(nome, idade);
		// Instanciando uma pessoa (dados públicos)

		// Armazenando retorno do método em uma variável local
		boolean atingiuMaioridade = pessoa.verificaMaioridade();

		while (!atingiuMaioridade) {
			pessoa.fazAniversario();
			atingiuMaioridade = pessoa.verificaMaioridade();
		}

		return pessoa;
	}

}
```

Ao executar a classe “DefaultMain” o processo será executado com sucesso, com o mesmo resultado de quando executado com a classe “ProtectedMain”. Pois ambas as classes estão no mesmo pacote.

Ao mover ou copiar a classe “DefaultMain” para outro pacote, a IDE indicará que o método “fazAniversario” não está visível, como na imagem abaixo. Isso acontece porque a classe “ProtectedPessoa” não está no mesmo pacote.

Figura 3 - Métodos padrões: exemplo de aviso da IDE

Fonte: do Autor (2022)

Agora, vamos criar um novo pacote “other” dentro do pacote “defaults”, dentro do novo pacote vamos criar também uma classe “DefaultPessoaOther” e estender “DefaultPessoa”. Na nova classe criaremos os construtores como os na classe herdada e repassaremos os parâmetros para o construtor da classe pai. Enfim, vamos sobrepor o método “fazAniversario” (utilizaremos a anotação “@ Override” no método para indicar a sobreposição) chamando dentro do novo método, o método protegido da classe pai.

Lembre-se de importar a classe “DefaultPessoa”.

A classe ficará ficará como o código apresentado a seguir:

```java
public class DefaultPessoaOther extends DefaultPessoa {

	public DefaultPessoaOther() {
		super();
	}

	public DefaultPessoaOther(String nome, Integer idade) {
		super(nome, idade);
	}

	@Override
	public void fazAniversario() {
		super.fazAniversario();
	}

}
```

Com isso, a IDE indicará que o método “fazAniversario” da classe pai não está visível, como na imagem a seguir. Isso acontece porque a classe “DefaultPessoa” não está no mesmo pacote.

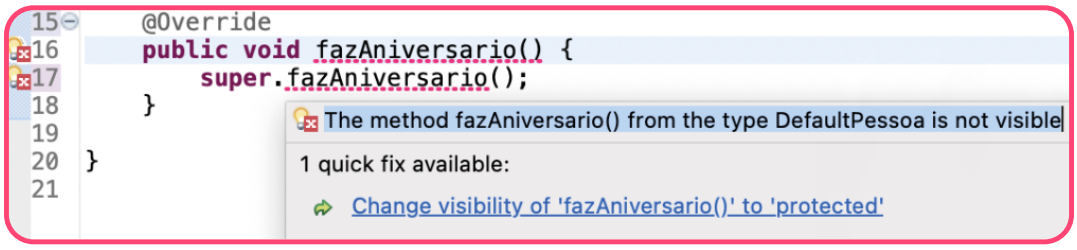

Figura 4 - Métodos padrões: exemplo de aviso da IDE

Fonte: do Autor (2022)

Portanto, este método somente estará visível para classes que estejam no mesmo pacote.

Métodos privados

Métodos “private” são visíveis apenas na sua própria classe. Para exemplificar este uso, vamos criar um pacote com nome “private” e no mesmo criaremos uma classe “*PrivatePessoa*” com um atributo “documento”.

Confira um passo a passo para implementar alguns métodos:

1. Pegar documento.
2. Alterar documento:
    2.1. Verificar se documento é válido.
    2.2. Realizar tratamentos no documento.

A classe aparecerá assim:

```
public class PrivatePessoa {

	private String documento;

	public String getDocumento() {
		return documento;
	}

	public void alterarDocumento(String novoDocumento) {
		System.out.println("\n");
		if (verificarDocumento(novoDocumento)) {
			documento = tratarDocumento(novoDocumento);
			System.out.println("Documento alterado com sucesso!");
		} else {
			System.out.println("Documento inválido!");
		}
	}

	private boolean verificarDocumento(String novoDocumento) {
		if (novoDocumento == null) {
			return false;
		}
		novoDocumento = tratarDocumento(novoDocumento);
		if (novoDocumento.trim().equals("")) {
			return false;
		}
		if (novoDocumento.length() < 11) {
			return false;
		}
		return true;
	}

	private String tratarDocumento(String novoDocumento) {
		return novoDocumento.trim(); // Elimina espaços vazios
	}

}
```

Para utilizar essa classe vamos criar uma classe “PrivateMain”, nela conterá o método “main” que chamará o método “utilizandoMetodosPrivate”, no mesmo haverá uma instância de “PrivatePessoa” que irá alterar o documento com documento inválido, válido e também tentará acessar um método private. A classe ficará dessa forma:

```
public class PrivateMain {

	public static void main(String[] args) {
		utilizandoMetodosPrivate();
	}

	public static void utilizandoMetodosPrivate() {
		PrivatePessoa pessoaComDocumento = new PrivatePessoa();

		pessoaComDocumento.alterarDocumento(“00000000000”);
		System.out.println(“Documento Atual: “ + pessoaComDocumento.
		getDocumento());

		pessoaComDocumento.alterarDocumento(“ 111111111    “);
		System.out.println(“Documento Atual: “ + pessoaComDocumento.
		getDocumento());

		pessoaComDocumento.alterarDocumento(“22222222222”);
		System.out.println(“Documento Atual: “ + pessoaComDocumento.
		getDocumento());

		pessoaComDocumento.alterarDocumento(null);
		System.out.println(“Documento Atual: “ + pessoaComDocumento.
		getDocumento());

		pessoaComDocumento.verificarDocumento(“33333333333”);
		pessoaComDocumento.tratarDocumento(“44444444444”);

	}

}
```

Ao tentar acessar os métodos privados, sua a IDE indicará que os métodos “verificarDocumento” e “tratarDocumento” não estão visíveis, como na imagem abaixo. Isso acontece porque os métodos só podem ser usados pela própria classe.

Figura 5 - Métodos privados: exemplo de aviso da IDE

Fonte: do Autor (2022)

Ao comentar as linhas com este problema, o retorno no console será:

```
Documento alterado com sucesso!
Documento Atual: 00000000000


Documento inválido!
Documento Atual: 00000000000


Documento alterado com sucesso!
Documento Atual: 22222222222


Documento inválido!
Documento Atual: 22222222222
```

Portanto, métodos com modificador “private” estão impedidos de serem usados fora de sua própria classe.

Passagem de parâmetros

No exemplo de método público, mais especificamente na classe “PublicMain”, podemos perceber que quando a instância da classe “PublicPessoa” foi criada com 15 anos e com a uma variável nomeada “pessoa”, ao retornar estava com 18 anos e com a variável renomeada para “ana” (conforme imagem abaixo). Isso porque, quando se trata de objetos, o que é passado por parâmetro, nada mais é do que a referência do objeto. No caso de variáveis dos tipos primitivos o que é passado por parâmetro é o valor da variável que o armazena. Confira.

```
9     public static void utilizandoMetodosPublic() {
10
11        System.out.println("\n"); // Quebra de linha
12        System.out.println("####################");
13        System.out.println("# Métodos públicos #");
14        System.out.println("####################");
15        System.out.println("\n"); // Quebra de linha
16
17        PublicPessoa ana = criandoAteMaioridade("Ana", 15);
18
19        System.out.println("Agora " + ana.getNome() + " tem " + ana.getIdade() + " anos de idade!");
20
21    }
22
23    public static PublicPessoa criandoAteMaioridade(String nome, Integer idade) {
24
25        PublicPessoa pessoa = new PublicPessoa(nome, idade); // Instanciando uma pessoa (dados públicos)
26
27        // Armazenando retorno do método em uma variável local
28        boolean atingiuMaioridade = pessoa.verificaMaioridade();
29
30        while (!atingiuMaioridade) {
31            pessoa.fazAniversario();
32            atingiuMaioridade = pessoa.verificaMaioridade();
33        }
34
35        return pessoa;
36    }
```

Figura 6 - Exemplo de passagem de parâmetros

Fonte: do Autor (2022)

Em seguida, comece a colocar em prática seus conhecimentos.

VAMOS PRATICAR

Nada como a boa e velha mão na massa para exercitar seus conhecimentos. Para organizarmos nosso projeto, criaremos alguns pacotes:

Figura 7 - Exemplo de pacotes

Fonte: do Autor (2022)

- **modelos:** armazenaremos aqui nossas classes que representam nossas entidades;
- **repositórios:** ficarão nossas classes de armazenamento, busca e persistência;
- **serviços:** estarão nossas classes de serviço. Essas serão as classes que acessarão diretamente as classes de repositório, fazendo validações, processamentos e demais tratamentos necessários tanto para salvar nossas entidades quanto para consultá-las.

A classe criada no pacote modelos será “Pessoa”, ela terá os atributos nome, sobrenome, data de nascimento e país. Vamos criar também os métodos “getters” e “setters” e o “toString” para exibirmos a instância no console da melhor forma possível. A classe ficará dessa forma:

```
public class Pessoa {

	private String nome;
	private String sobrenome;
	private Date dataNascimento;
	private String pais;

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getSobrenome() {
		return sobrenome;
	}

	public void setSobrenome(String sobrenome) {
		this.sobrenome = sobrenome;
	}

	public Date getDataNascimento() {
		return dataNascimento;
	}
```

```
	public void setDataNascimento(Date dataNascimento) {
		this.dataNascimento = dataNascimento;
	}

	public String getPais() {
		return pais;
	}

	public void setPais(String pais) {
		this.pais = pais;
	}

	@Override
	public String toString() {
		return "\n" + nome + " " + sobrenome + "\nSeu nascimento foi
		" + dataNascimento + " (" + pais + ")\n";
	}
}
```

A próxima classe que criaremos será no pacote repositórios. A classe "PessoaRepositorio", será responsável por armazenar e buscar as pessoas cadastradas. Sendo assim precisaremos criar uma variável para armazenar as pessoas e métodos para adicionar, remover e buscar pessoas. A classe ficará do modo apresentado a seguir:

```
public class PessoaRepositorio {

	// Armazenará as pessoas cadastradas
	private static List<Pessoa> PESSOAS = new ArrayList<>();

	public Pessoa salvar(Pessoa pessoa) {
		PESSOAS.add(pessoa);
		return pessoa;
	}

	public void apagar(Pessoa pessoa) {
		PESSOAS.remove(pessoa);
	}

	public List<Pessoa> buscarTodasPessoas() {
		return PESSOAS;
	}
}
```

Entenda abaixo os métodos criados:

- Método “salvar(Pessoa pessoa)” receberá uma instância de “Pessoa”, incluirá na lista “PESSOAS” e retornará a mesma instância da classe pessoa.
- Método “apagar(Pessoa pessoa)” receberá uma instância de “Pessoa”, removerá a instância da lista “*PESSOAS*” e não terá retorno (***void***).
- Método “buscarTodasPessoas()” não receberá parâmetros e retornará a lista “PESSOAS”.

Em nosso pacote de serviços criaremos a classe “PessoaServico”, ela será a classe que intermediará o acesso à classe “PessoaRepositorio”. Sendo assim, inicialmente, criaremos uma variável “repositorio” para acessar a classe “PessoaRepositorio” e os métodos para adicionar e buscar pessoas. A classe será da seguinte forma:

```java
public class PessoaServico {

	private PessoaRepositorio repositorio = new PessoaRepositorio();

	public Pessoa salvar(Pessoa pessoa) {
		return repositorio.salvar(pessoa);
	}

	public List<Pessoa> buscarTodasPessoas() {
		return repositorio.buscarTodasPessoas();
	}
}
```

Entenda abaixo os métodos criados:

- Método “salvar(Pessoa pessoa)” receberá uma instância de “Pessoa”, que imediatamente repassará esta instância para o método salvar do “repositorio” e retornará a mesma instância retornada pelo repositório.
- Método “buscarTodasPessoas()” não receberá parâmetros e retornará a lista que o repositório devolver.

Com tudo isso, você pode utilizar tranquilamente a classe que contém o método “main”. Vamos criá-la ao lado dos pacotes especificados anteriormente, confira na figura:

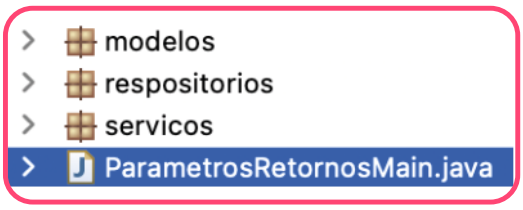

Figura 8 - Exemplo de novo pacote criado

Fonte: do Autor (2022)

A classe “ParametrosRetornosMain” terá o método “parametrosRetornos()” que o método “main” chamará. Nesse método criaremos instâncias da classe “Pessoa”, preenchendo com os dados que preferir. Também precisaremos criar uma instância de “PessoaServico” para salvar nossa instância de “Pessoa” e consultá-la. A cada instância criada, salvaremos ela e no final do método vamos buscar todas as instâncias salvas. A classe ficará como a seguir:

```
public class ParametrosRetornosMain {
	public static void main(String[] args) {
		parametrosRetornos();
	}
	public static void parametrosRetornos() {
		PessoaServico pessoaServico = new PessoaServico();

		Pessoa novaPessoa = new Pessoa();
		novaPessoa.setNome("Pessoa");
		novaPessoa.setSobrenome("Nova");
		novaPessoa.setDataNascimento(new Date());
		novaPessoa.setPais("Brasil");

		pessoaServico.salvar(novaPessoa);

		novaPessoa = new Pessoa();
		novaPessoa.setNome("Novíssima");
		novaPessoa.setSobrenome("Pessoa");
		novaPessoa.setDataNascimento(new Date());
		novaPessoa.setPais("Argentina");

		pessoaServico.salvar(novaPessoa);

		System.out.println(pessoaServico.buscarTodasPessoas());
	}
}
```

Ao executar a classe "ParametrosRetornosMain", você terá o seguinte retorno no console:

```
[
Pessoa Nova
Seu nascimento foi Sat May 28 17:29:45 BRT 2022 (Brasil)
,
Novíssima Pessoa
Seu nascimento foi Sat May 28 17:29:45 BRT 2022 (Argentina)
]
```

Como a lista de pessoas é uma lista de objetos, ao passar esse objeto por várias classes e métodos, você estará enviando a referência de memória deste objeto, ou seja, se alterar algum atributo da instância desse objeto, em qualquer lugar (seja na classe “main”, na de serviço ou até mesmo na classe repositório), essa alteração terá efeito em todos os lugares onde a instância esteja.

Para conferir isso na prática, vamos incrementar nosso código no método “parametrosRetornos()” da classe “ParametrosRetornosMain”. Abaixo de onde consultamos todas as pessoas, vamos apenas alterar o nome da última pessoa adicionada e logo abaixo consultar novamente. Adicione também alguma marcação para separar a primeira consulta da última.

Perceba que não vamos salvar a instância novamente, apenas consultar.

A classe ficará da seguinte forma:

```java
public class ParametrosRetornosMain {
	public static void main(String[] args) {
		parametrosRetornos();
	}
	public static void parametrosRetornos() {
		Pessoa novaPessoa = new Pessoa();
		novaPessoa.setNome("Pessoa");
		novaPessoa.setSobrenome("Nova");
		novaPessoa.setDataNascimento(new Date());
		novaPessoa.setPais("Brasil");

		PessoaServico pessoaServico = new PessoaServico();
		pessoaServico.salvar(novaPessoa);

		novaPessoa = new Pessoa();
		novaPessoa.setNome("Novíssima");
		novaPessoa.setSobrenome("Pessoa");
		novaPessoa.setDataNascimento(new Date());
		novaPessoa.setPais("Argentina");

		pessoaServico.salvar(novaPessoa);

		System.out.println("Primeira consulta");
		System.out.println(pessoaServico.buscarTodasPessoas());

		novaPessoa.setNome("Frederico");

		System.out.println("\n\nÚltima consulta");
		System.out.println(pessoaServico.buscarTodasPessoas());

	}
}
```

Ao executar a classe “ParametrosRetornosMain”, você conferirá o seguinte retorno no console:

```
Primeira consulta
[
Pessoa Nova
Seu nascimento foi Sat May 28 17:36:41 BRT 2022 (Brasil)
,
Novíssima Pessoa
Seu nascimento foi Sat May 28 17:36:41 BRT 2022 (Argentina)
]


Última consulta
[
Pessoa Nova
Seu nascimento foi Sat May 28 17:36:41 BRT 2022 (Brasil)
,
Frederico Pessoa
Seu nascimento foi Sat May 28 17:36:41 BRT 2022 (Argentina)
]
```

Conforme o console, você pode perceber que o nome da última pessoa foi alterado, mesmo sem executar o método salvar. Interessante, não é mesmo? Continue e a seguir, confira um desafio baseado em seus estudos até aqui.

DESAFIO

O objetivo é criar uma forma de instanciar inúmeros objetos “Pessoa” com informações mais diferentes possíveis, salvá-los no repositório que criamos anteriormente e exibir no console todos os registros salvos.

Dicas:

- Pode ser aproveitado o código já criado.
- Lembre-se: métodos podem usar outros métodos em seu corpo.
- Podem ser usados laços de repetição, números randômicos, bibliotecas de terceiros e o que mais preferir.
- Pesquisas e códigos da internet estão liberados, mas lembre-se que pode sempre melhorá-los.
- Use sua criatividade, não há uma forma correta para resolver o desafio.

Boa prática!

REFERÊNCIAS

CARVALHO, T. L. **Orientação a objetos**: aprenda seus conceitos e suas aplicabilidades de forma efetiva, 2020.

MAUDA, E. **OO + Java básico: modificadores de acesso.** Disponível em: http://www.mauda.com.br/?p=1433. Acesso em: 22 maio 2022.

RICARTE. I. L. M. **Definição do corpo de método.** Disponível em: https://www.dca.fee.unicamp.br/cursos/PooJava/classes/corpomet.html. Acesso em: 21 maio 2022.

SACURAI, R. G. **Modificadores de visibilidade**. Disponível em: http://www.universidadejava.com.br/java/java-modificador/. Acesso em: 20 maio 2022.

SILVEIRA, G. **Algoritmos em Java**: busca ordenação e análise, 2017.

TURINI, R. **Desbravando Java e orientação a objetos**: um guia para o iniciante da linguagem, 2014.

ZUNIC, N. **Java non access modifiers:** static, abstract, final. Disponível em: https://medium.com/java-vault/java-non-access-modifiers-static-abstract-final-64e8518ca9d. Acesso em: 20 maio 2022.

Gabriel Augustin

Experiência em desenvolvimento de aplicações Web com Java e Spring. Atualmente lidera equipes de desenvolvimento back-end e front-end, em projetos com foco em gateways de pagamento e criptomoedas. Atualmente é Tech Leader na KahshPay e Mentor Educacional no LAB365 do SENAI/SC.

SENAI <LAB365>