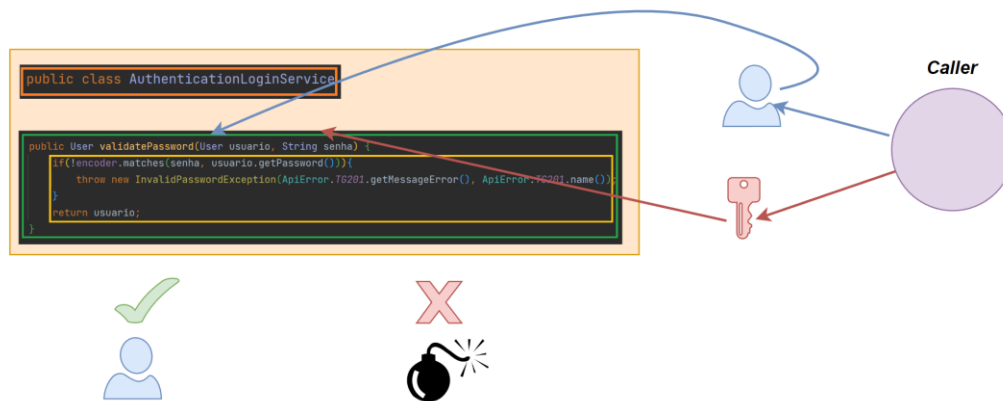




Métodos – Java

Métodos representam comportamentos de um **Objeto de uma determinada classe**. Métodos possuem uma **lógica autocontida** com o propósito de serem usados várias vezes. Por exemplo, um **objeto de autenticação de login** tem o **comportamento de validar uma senha**:

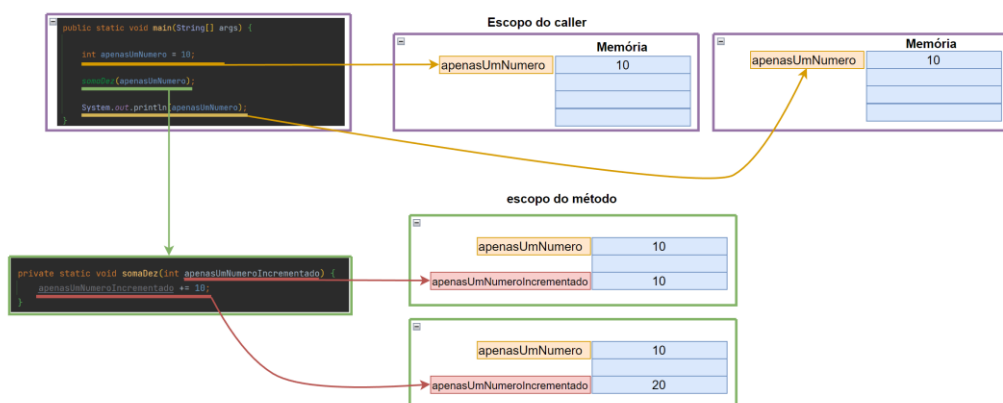


Passagem de parâmetros – Java

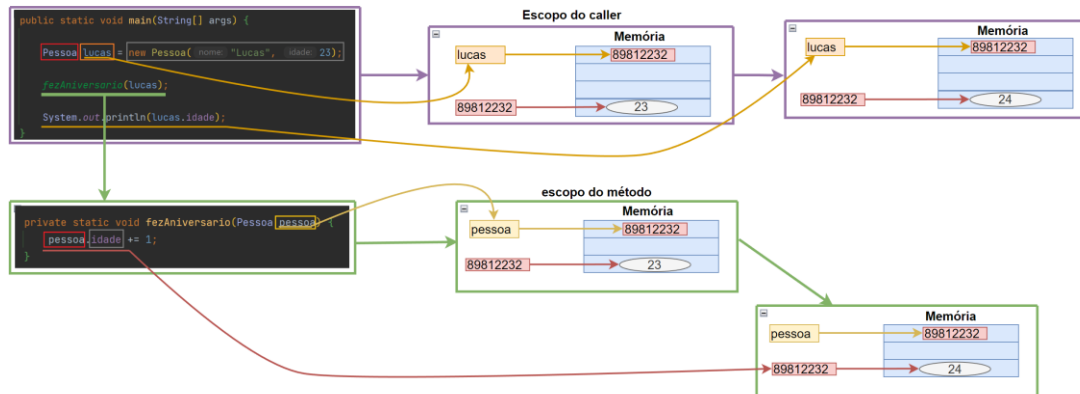
Em runtime os **nomes lógicos das variáveis** (*nome no código fonte*) se transformam em **endereço na memória** que aponto para o **valor delas**.

Quando uma variável é passada por parâmetro, você passa o **valor do endereço delas**. Em caso de primitivo vai ser a **cópia do valor**, e em caso de objeto vai ser o **valor da referência que aponta para o objeto**.

Quando você passa um tipo primitivo, você **está criando uma cópia do valor que só existe no escopo do método, portanto a variável original não é afetada no escopo do caller**:



Quando você passa um objeto como parâmetro, você está utilizando o **valor na memória** que aponta para o **objeto**, então a **variável do escopo do método** vai simplesmente passar a apontar para a **referência** do objeto que já existe:



Sobrecarga – Java

Uma conveniência que java oferece é a sobrecarga de métodos. Basicamente falando são métodos que possuem o mesmo nome, mas que tem **parâmetros diferentes**, seja em **quantidade, tipo ou ordem**.

A conveniência da sobrecarga é poder fazer a mesma ação de forma semântica, mas alterando o comportamento, é uma flexibilidade maior:

```

public void add(int index, Object element) {
    this.rangeCheckForAdd(index);
    ++this.modCount;
    int s;
    Object[] elementData;
    if ((s = this.size) == (elementData = this.elementData).length) {
        elementData = this.grow();
    }
    System.arraycopy(elementData, index, elementData, (index + 1), (s - index));
    elementData[index] = element;
    this.size = s + 1;
}

private void add(Object e, Object[] elementData, int s) {
    if (s == elementData.length) {
        elementData = this.grow();
    }
    elementData[s] = e;
    this.size = s + 1;
}

public boolean add(E e) {
    ++this.modCount;
    this.add(e, this.elementData, this.size);
    return true;
}

```

```

List<Pen> penList = new ArrayList<ArrayList<
    new Pen("Black", "B", "0.7"),
    new Pen("Blue", "B", "0.7")
>());

penList.add(new Pen());
add(int i, Pen p) void
addAll(Collection c, boolean Pen collection) boolean
addAll(int i, Collection c, boolean Pen collection) boolean

```



Var args – Java

É uma sintaxe especial que permite o método receber diversas variáveis, essas variáveis são convertidas em um array. Um método **só pode ter um argumento como var args**, e ele **sempre deve**

ser o último parâmetro:

```
public static void main(String[] args) {  
    String course1 = "Java";  
    String course2 = "Design Patterns";  
    studs( flag: true, course1, course2);  
}
```

```
String[] courses = {"Java", "Design Patterns"};
```

```
public static void studs(boolean flag, String ... courses){...}
```



Construtores – Java

Métodos construtores servem para construir um objeto de uma classe, geralmente já atribuindo um estado ao objeto. A sintaxe do **construtor é simplesmente o nome da classe, sem retorno**.

Mesmo **quando não declarado explicitamente, uma classe ainda tem um construtor, que é o default**. Sendo esse o “**default**”, quando instanciada pelo construtor default, o **objeto** vai **atribuir valores default** para suas **propriedades**:



Um uso mais real do construtor é **obrigar que te passem determinados valores para que o objeto funcione corretamente (injeção de dependência)**. É algo como “esse objeto só vai funcionar se você me fornecer alguns valores, então passa os valores ou nem vai instanciar o objeto”.

Pra obter esse comportamento é necessário apenas **não declarar o construtor default**, apenas os construtores especificados por você. Dessa maneira só é possível instanciar um objeto usando os construtores disponíveis:

