

Classes e Objetos: atributos e métodos

Programação Orientada a Objetos

- Olhar o mundo como se tudo pudesse ser representado por objetos
- Estruturação do programa é baseada na representação de **objetos** do mundo real (**estados** + **comportamento**)
- Vantagens
 - Facilidade de manutenção
 - Maior extensibilidade
 - Maior reuso

Objeto DVD



voltar()

pausar()

alterarHora()

avancar()

parar()

tocar()

carregarDisco()

tempoDecorrido

horaAtual

tipoSistema

duracaoDisco

Classes e Objetos

- Classes especificam a estrutura e o comportamento dos objetos
- Classes são como "moldes" para a criação de objetos
- Objetos são **instâncias** de classes.

- Um objeto representa uma entidade do mundo real
- Todo objeto tem



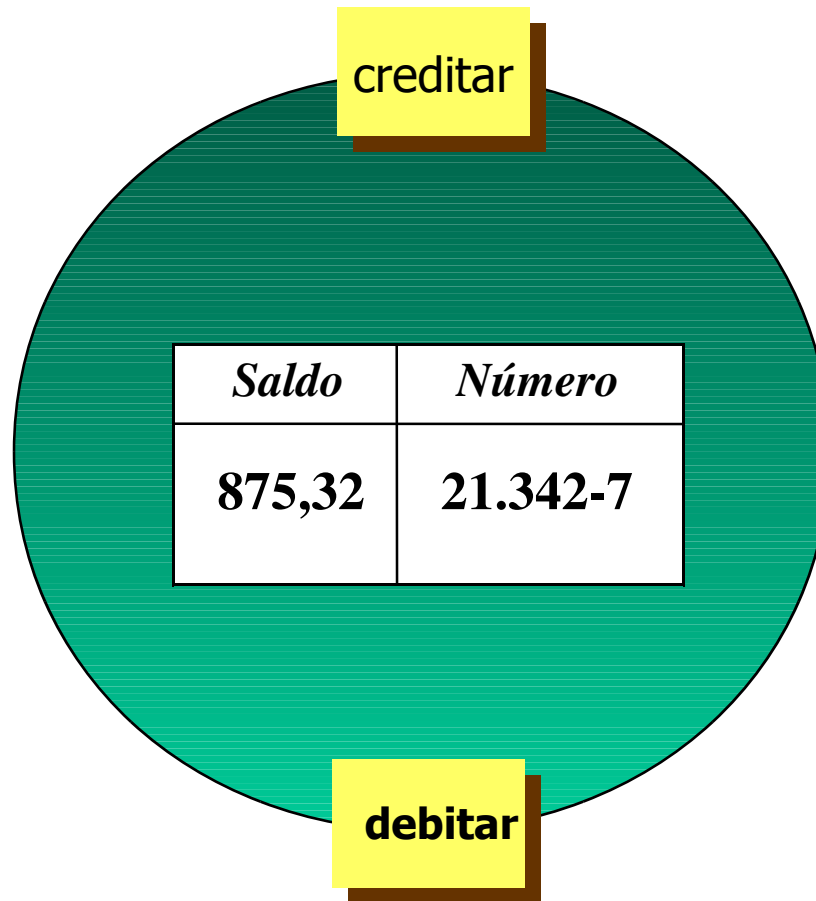
Diagram illustrating the components of an object:

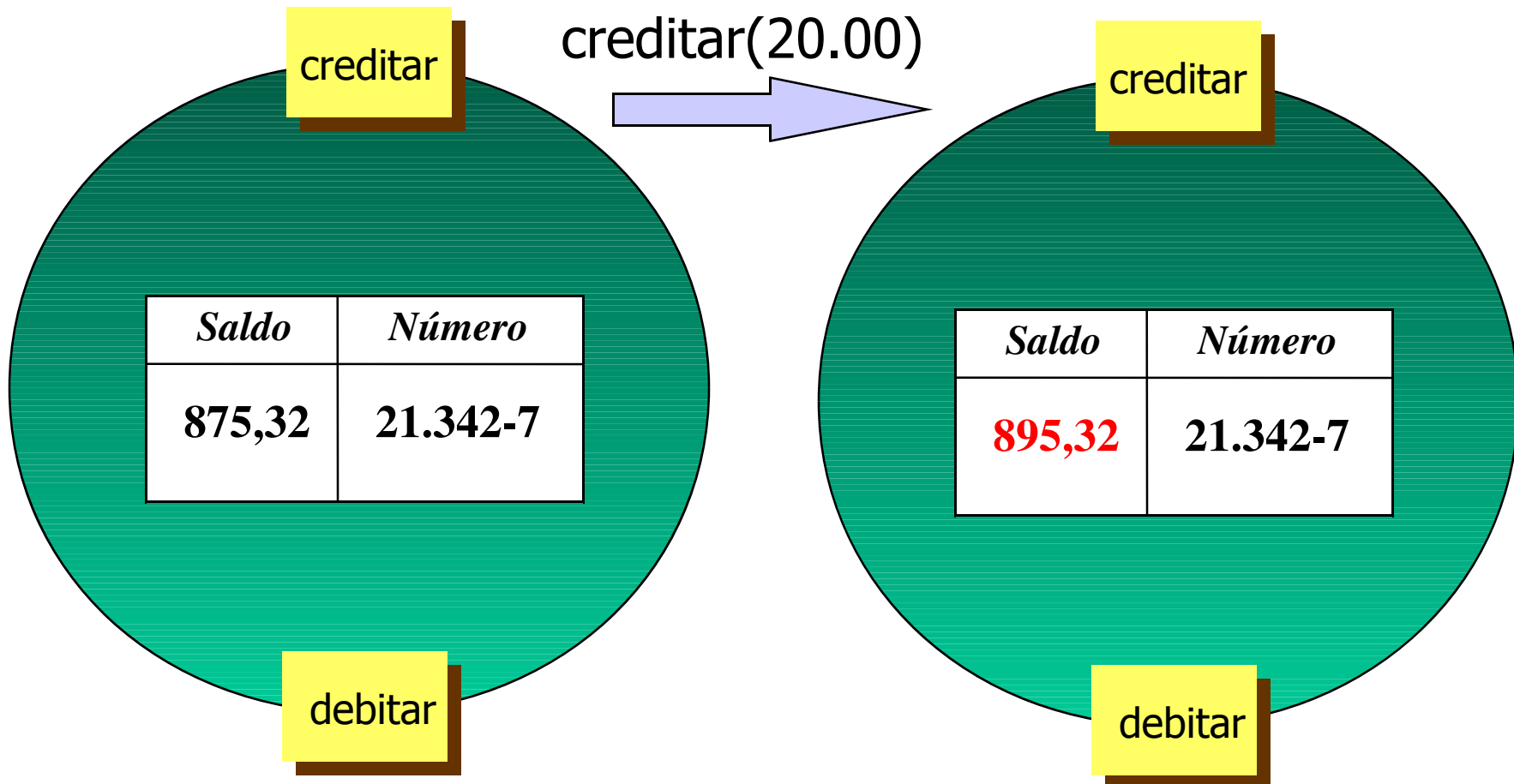
- Identity
- Estado
- Comportamento

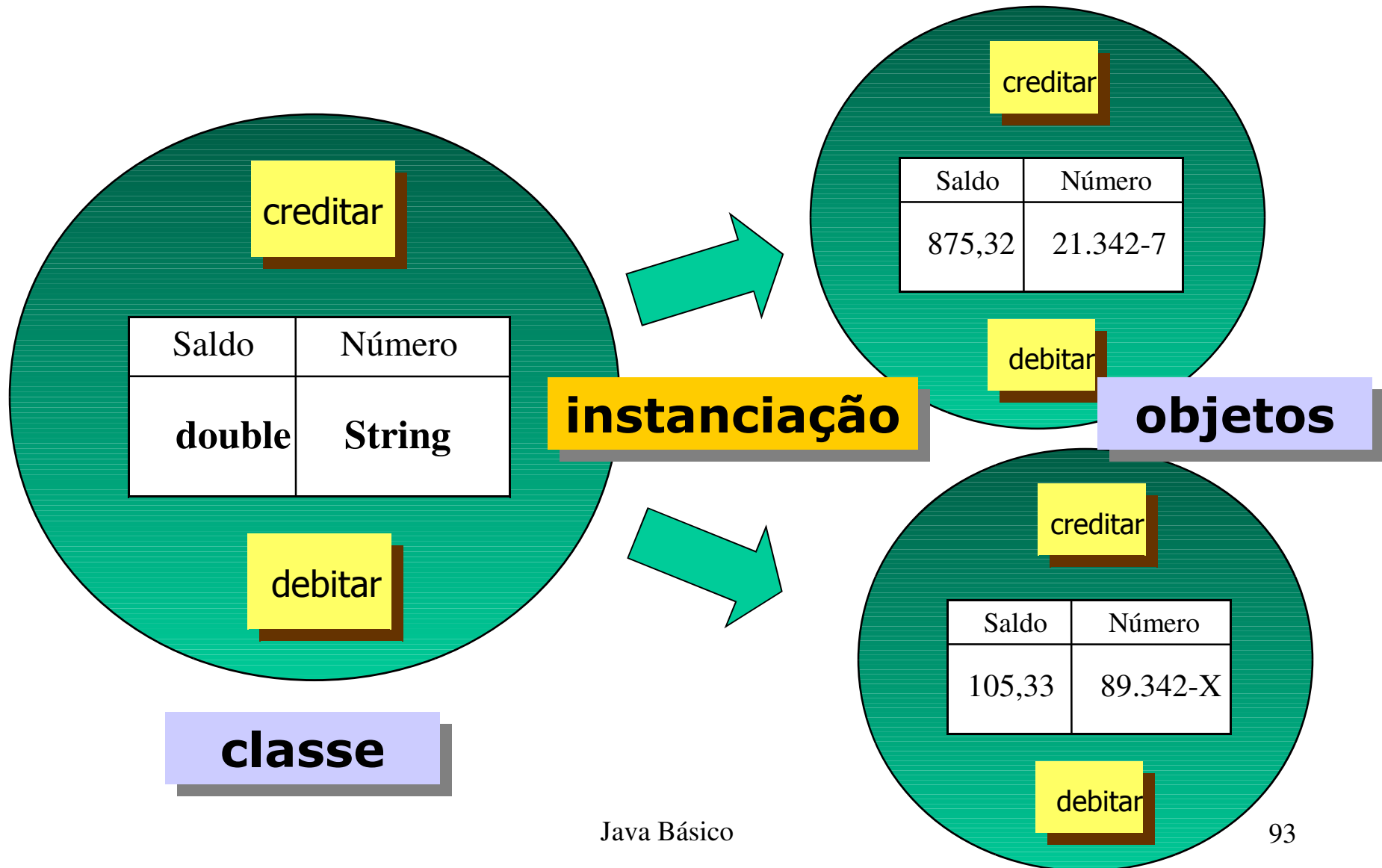
Estado

Comportamento

- Identidade
 - Todo objeto é único e pode ser distinguido de outros objetos
- Estado
 - Todo objeto tem estado, que é determinado pelos dados contidos no objeto
- Comportamento
 - O comportamento de um objeto é definido pelos serviços/operações que ele oferece







```
class NomeDaClasse {  
    CorpoDaClasse  
}
```

O corpo de uma classe pode conter

- atributos
- métodos
- construtores
- outras classes...

```
class NomeDaClasse {  
    atributo1;  
    atributo2;  
    ...  
    método1 {  
        Corpo do método1  
    }  
    método2 {  
        Corpo do método2  
    }  
}
```

```
class Conta {  
    CorpoDaClasse  
}
```

Atributos

Declaração de atributos

modificadores tipo nome ;



```
private static String numero;
```

```
double LIMITE_MAXIMO;
```

```
double saldo;
```

```
private String nome, sobrenome;
```

Vários atributos podem ser declarados na mesma linha

```
int numEstoque = 8;
```

Um atributo pode ser inicializado na declaração

Exemplos de atributos

```
class Livro {  
    int anoDePublicacao;  
    int numeroDePaginas;  
    String titulo;  
    ...  
}
```

```
class Conta {  
    String numero;  
    double saldo;  
    ...  
}
```

```
class Cabine {  
    int nivel;  
    String codigo;  
    int codCategoria;  
    int lotacaoMaxima;  
    ...  
}
```

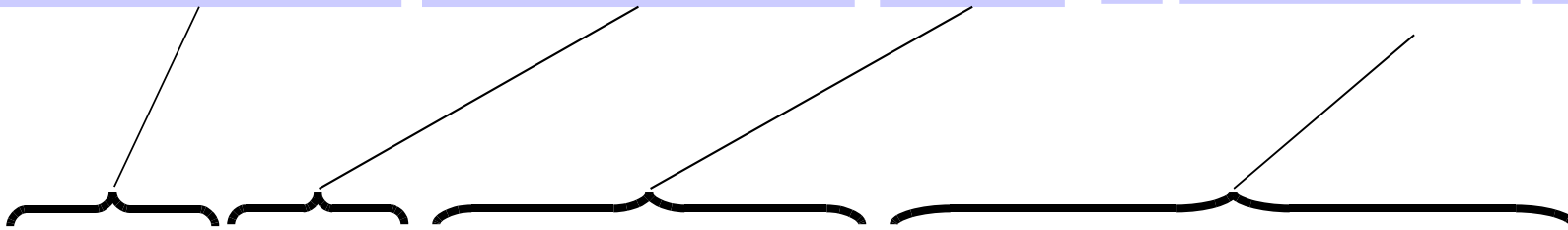
Métodos

O que são métodos?

Métodos são operações que realizam ações e modificam os valores dos atributos do objeto responsável pela sua execução

Declaração de métodos

modificadores tipo de retorno nome (parâmetros) ;



```
private double obterRendimento(String numConta, int mes);
```

```
nomeDoMetodo() {  
    Corpo do método  
}
```

O corpo do método

- O corpo do método contém os comandos que determinam as ações do método
- Esses comandos podem
 - realizar simples atualizações dos atributos de um objeto
 - retornar valores
 - executar ações mais complexas como chamar métodos de outros objetos
- O corpo do método também pode conter declarações de variáveis
 - Variáveis cuja existência e valores são válidos somente dentro do método em que são declaradas.

Exemplo de Método

```
class Conta {  
    String numero;  
    double saldo;  
  
    void creditar(double valor) {  
        saldo = saldo + valor;  
    }  
    ...  
}
```

Métodos e tipo de retorno

```
class Conta {  
    String numero;  
    double saldo;  
  
    String getNumero() {  
        return numero;  
    }  
    double getSaldo() {  
        return saldo;  
    }  
    ...  
}
```

Os métodos que retornam valores como resultado usam o comando **return**

Mais sobre métodos

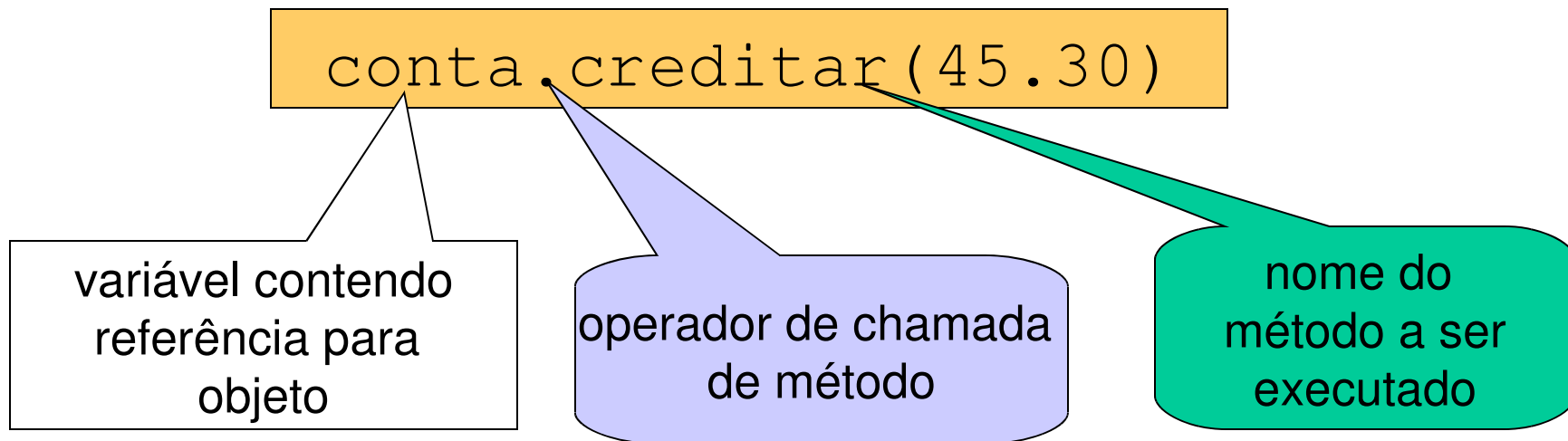
```
class Conta {  
    ...  
  
    void debitar(double valor) {  
        saldo = saldo - valor;  
    }  
}
```

Usa-se **void** para indicar que o método não retorna nenhum valor, apenas altera os valores dos atributos de um objeto

Por que o debitar não tem como parâmetro o número da conta?

Chamada de métodos

- Métodos são invocados em instâncias (objeto) de alguma classe.
 - Podem também ser invocados a partir da própria classe (métodos estáticos).
- Os objetos se comunicam para realizar tarefas
- Parâmetros são passados por “cópia”
- A comunicação é feita através da **chamada de métodos**

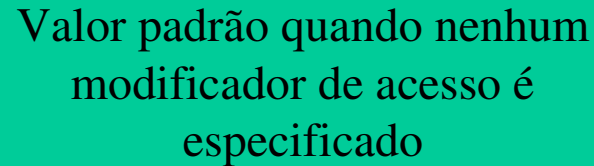


Modificadores

Modificadores

- Modificadores de acesso

- public
- protected
- private
- default (friendly)



Valor padrão quando nenhum
modificador de acesso é
especificado

- Outros modificadores

- static
- final
- native
- transient
- synchronized

Modificadores de Acesso

- Controlam o acesso aos membros de uma classe
- Membros de uma classe:
 - A própria classe
 - Atributos
 - Métodos e construtores (um tipo especial de métodos)
- Não são aplicados à variáveis

public

- Uma classe **public** pode ser instanciada por qualquer classe
- Atributos **public** podem ser acessados (lidos, alterados) por objetos de qualquer classe
- Métodos **public** podem ser chamados por métodos de qualquer classe

```
public class Conta {  
    public String numero;  
    public double saldo;  
  
    public void creditar(double valor) {  
        saldo = saldo + valor;  
    }  
    ...  
}
```

protected

- Usado somente para atributos e métodos
- Atributos **protected** podem ser acessados (lidos, alterados) por objetos de classes dentro do mesmo **pacote** ou de qualquer **subclasse** da classe ao qual ele pertence
- Métodos **protected** podem ser chamados por objetos de classes dentro do mesmo **pacote** ou de qualquer **subclasse** da classe ao qual ele pertence

```
public class Conta {  
    protected String numero;  
    protected double saldo;  
  
    protected void creditar(double valor) {  
        saldo = saldo + valor;  
    }  
    ...  
}
```

default (friendly)

- A classe é visível somente por classes do mesmo **pacote**
- Se um atributo não tem nenhum modificador de acesso associado, ele é “implicitamente” definido como **friendly**, e só é visível para objeto de classes do mesmo **pacote**
- Se um método não tem nenhum modificador de acesso associado, ele é “implicitamente” definido como **friendly**, e só pode ser chamado a partir de objetos de classes do mesmo **pacote**

```
class Conta {  
    String numero;  
    double saldo;  
  
    void creditar(double valor) {  
        saldo = saldo + valor;  
    }  
    ...  
}
```

private e encapsulamento

atributos **private** podem ser acessados somente por **objetos da mesma classe**

```
class Pessoa {  
    private int anoDeNascimento;  
    private String nome, sobrenome;  
    private boolean casada = false;  
    ...  
}
```

- Java não obriga o uso de **private**, mas vários autores consideram isso essencial para a programação orientada a objetos
- Impacto em coesão e acoplamento
- Use **private** para atributos!

private

- Métodos **private** só podem ser chamados por métodos da classe onde são declarados

```
class Pessoa {  
    private int anoDeNascimento;  
    ...  
    int getAnoDeNascimento() {  
        return formatarAno();  
    }  
  
    private int formatarAno() {  
        //código para formatar o ano  
    }  
  
    ...  
}
```

Outros modificadores

- **final**
 - Pode ser utilizado para em qualquer membro de uma classe
 - Torna o atributo **constante**
 - O atributo não pode ser alterado depois de inicializado
 - Métodos **final** não podem ser **redefinidos**
 - Classes **final** não podem ser **estendidas**

Outros modificadores

- **final**

```
class ConstantesBanco {  
    public static final int NUM_MAXIMO_CONTAS;  
    private static final double LIMITE_MIN_CHEQUES;  
    private static final int NUM_CHEQUES_TALAO;  
    ...  
}
```

Atributos **final**
geralmente são
declarados como
static também

Pelo padrão de codificação, constantes devem
ter seus nomes em **maiúsculas**

Outros modificadores

- **static**
 - Pode ser usado somente em atributos e métodos
 - Atributos static pertencem à classe e não aos objetos
 - Só existe uma cópia de um atributo static de uma classe, mesmo que haja vários objetos da classe
 - Atributos static são muito usados para constantes
 - O acesso é feito usando o nome da classe:

```
int numCheques = numTaloes *  
    ConstantesBanco.NUM_CHEQUES_TALAO;
```

Outros modificadores

- **static**

- Métodos **static** pertencem a classes e não a objetos
- Podem ser usados mesmo sem criar os objetos
- Só podem acessar diretamente atributos estáticos
- O acesso é feito usando o nome da classe:

```
x = Math.random();
```

```
media = Funcoes.calcularMedia(valores);
```

Outros modificadores

- **native**
 - Usado somente em métodos
 - Código nativo
- **transient**
 - Usado somente em atributos
 - Não é armazenado como parte persistente do objeto
- **synchronized**
 - Usado somente em métodos
 - Acesso concorrente

Criação e remoção de objetos

Criação de objetos

- Objetos precisam ser criados antes de serem utilizados
- Construtores precisam ser definidos na classe
- A criação é feita com o operador **new**

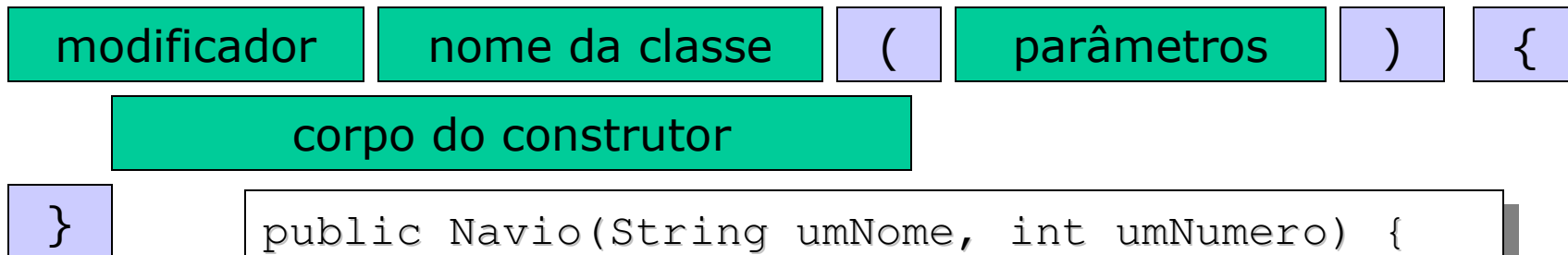
```
Conta c = new Conta();
```



construtor

Construtores

- Além de métodos e atributos, uma classe pode conter **construtores**
- Construtores definem como os atributos de um objeto devem ser inicializados
- São semelhantes a métodos, mas não têm tipo de retorno
- O nome do construtor deve ser exatamente o nome da classe.
- Uma classe pode ter diversos construtores, diferenciados pelos parâmetros



```
public Navio(String umNome, int umNumero) {  
    nome = umNome;  
    numCabines = umNumero;  
}
```

Construtor default

- Caso não seja definido um construtor, um construtor default é fornecido implicitamente
- O construtor default inicializa os atributos com seus valores default
- O construtor default não tem parâmetros:

```
public Conta() {  
}
```

Valores default para atributos

Tipo	Valor Default
byte, short, int, long	0
float	0.0f
double	0.0
char	`\u0000`
Tipos referência (Strings, arrays, objetos em geral)	null
boolean	false

Outros construtores

- Podem ser criados novos construtores, com parâmetros

```
class Conta {  
    String numero;  
    double saldo;  
    public Conta(String numeroConta, double  
        saldoInicial){  
        numero = numeroConta;  
        saldo = saldoInicial;  
    }  
    ...  
}
```

Quando é definido um construtor com parâmetros, o construtor default não é mais gerado

Mais sobre criação de objetos

```
Conta c;
```

```
...
```

```
c = new Conta("12345", 100);
```

Atribui à variável
c a referência
criada para o
novo objeto

responsável por
criar um objeto do
tipo Conta em
memória

responsável por
inicializar os
atributos do
objeto criado

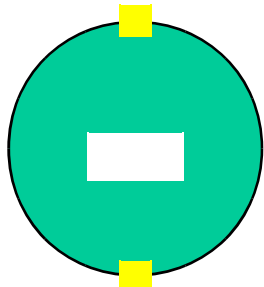
Remoção de objetos

- Não existe mecanismo de remoção explícita de objetos da memória em Java (como o `free()` de C++)
- O *Garbage Collector* (coletor de lixo) de Java elimina objetos da memória quando eles não são mais referenciados
- Você não pode obrigar que a coleta de lixo seja feita
- A máquina virtual Java decide a hora da coleta de lixo

Referências

Objetos são manipulados através de referências

`Conta c = null;` → `c == null`

`c = new Conta("1287", 0);` → `c` → 

`c.getSaldo();`

chama o método `getSaldo()` do
objeto referenciado pela
variável `c`

Referências

Mais de uma variável pode armazenar referências para um mesmo objeto (*aliasing*)

```
Conta a = new Conta("123-4", 340.0);  
Conta b;
```

```
b = a;
```

a e b passam a referenciar a mesma conta

```
b.creditar(100);  
System.out.println(a.getSaldo());
```

qualquer efeito via b
é refletido via a