
JAVA™

Mais sobre Loops

for

```
for(inicialização ;condição ;expressão ){
    // código a repetir;
}
```

```
for(int i=1;i<=10;i++)
    System.out.println(i+" ");
```

Loops ***for***

Loops for comuns (não aperfeiçoados)

```
for (int i; i < 10; i++){
    // faz algo 10 vezes
}
```

Diferença entre for e while

Um loop while apresenta apenas o teste booleano; nao tem uma expressão interna de inicialização ou iteração.
Ele será útil quando você não souber quantas vezes o loop será executado e quiser continuar a execução apenas enquanto alguma condição for verdadeira. Mas se você souber quantas vezes o loop será executado, um loop fora será mais simples. Veja o loop anterior usando while:

```
int i = 0;
while (i < 10){
    // faz algo 10 vezes
    i++;
}
```

Mais sobre Loops

do-while

```
do{
    // código a repetir;
}while(condição);
```

Controle de fluxo

if

if (*condição*){
    caso a condição seja verdadeira
    esse bloco de código será executado
}

Controle de fluxo

else

if (*condição*){
    caso a condição seja verdadeira
    esse bloco de código será executado
} else {
    caso a condição seja falsa
    esse bloco de código que será executado
}

Variáveis primitivas e de referência

Até agora usamos variáveis em duas situações: como estado do objeto (variáveis de instancia) e como variáveis locais (variáveis declaradas dentro de um método). Também usaremos variáveis como argumentos (valores enviados para um método pelo código que o chamou) e como tipos de retorno (valores retornados ao código que chamou o método).

Variáveis primitivas e de referência

As variáveis primitivas contêm valores básicos, que incluem inteiros, booleanos e números de ponto flutuante.

As variáveis de referência contêm referências a objetos.

Tipos primitivos

TIPO	QTD DE BITS	INTERVALO DE VALORES
boolean	Especifica da JVM	verdadeiro ou falso
byte	8 bits	-128 a 127
short	16 bits	-32768 a 32767
int	32 bits	-2147483648 a 2147483647
long	64 bits	-muita coisa a muita coisa
float	32 bits	varia
double	64 bits	varia

Não se pode despejar uma quantidade grande de café em uma xícara pequena.
Bom, poder até pode, mas vai se perder uma parte devido ao derramamento.
No Java, o compilador tentará ajudar a impedir isso se conseguir perceber que algo em seu código não caberá no conteiner (variável/xícara) que você estiver usando.

Por exemplo, não se pode despejar muitos inteiros em um conteiner de tamanho byte, como abaixo:

```
int x = 24;
byte b = x;
// isso não funcionará
```

Orientação a Objetos

Na primeira aula, colocamos todo o código no método main(). Essa não é exatamente uma abordagem orientada a objetos. Na verdade ela definitivamente não é orientada a objetos.

Usamos alguns objetos, como as matrizes de strings, mas não desenvolvemos nenhum tipo de objeto por nossa própria conta. Vamos agora deixar esse universo procedimental pra trás, sair de main() e começar a criar alguns objetos.

O que pensar quando projetar uma classe

Quando projetar uma classe, pense nos objetos que serão criados com esse tipo de classe. Considere:

As coisas que o objeto **conhece**

As coisas que o objeto **faz**

O que pensar quando projetar uma classe

As coisas que um objeto conhece sobre si mesmo se chamam:

variáveis de instância

As coisas que um objeto pode fazer se chamam:

métodos

Qual a diferença entre uma classe e um objeto?

Uma classe não é um objeto. Uma classe é o projeto de um objeto. Ela informa a maquina virtual como criar um objeto desse tipo especifico. Cada objeto criado a partir dessa classe terá seus próprios valores para as variáveis de instancia da classe. Por exemplo, você pode usar a classe Button para criar vários botões diferentes, e cada botão poderá ter sua própria cor, tamanho, forma, rótulo e assim por diante.

Criando o primeiro objeto

Para criação e uso de um objeto são necessárias duas classes. Uma para o tipo de objeto que deseja usar e outra para testar sua classe. É na classe de teste que devera ser inserido o método principal e nesse método, main(), serão criados e acessados os objetos da classe. A classe de teste terá apenas uma tarefa: testar os métodos e variáveis da classe de objeto.

O operador ponto (.)

O operador ponto (.) da acesso ao estado e comportamento (variáveis de instância e métodos) de um objeto.

c.latir();

c.tamanho = 40;

Como os objetos se comportam. Como o estado e o comportamento estão relacionados.

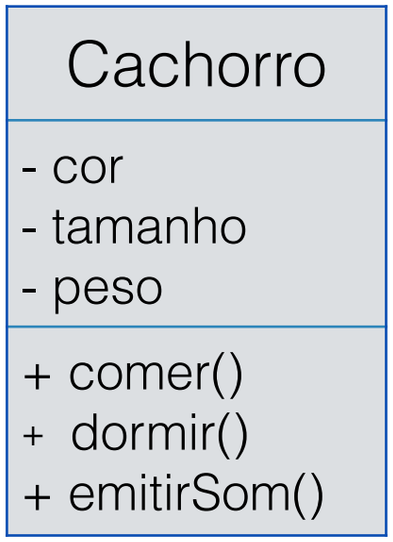
O estado afeta o comportamento, o comportamento afeta o estado. Sabemos que os objetos têm **estado** e **comportamento**, representados pelas **variáveis de instância** e **métodos**.

O que há de importante sobre um objeto é que ele tem um comportamento que atua sobre o seu estado. Em outras palavras, os métodos usam os valores das variáveis de instância.

Como os objetos se comportam. Como o estado e o comportamento estão relacionados.

Todas as instâncias de uma classe especifica têm os mesmos métodos, mas eles podem se comportar diferentemente com base no valor das variáveis de instância.

Criando o primeiro objeto




```
class CachorroTeste{
    public static void main(String[] args){
        Cachorro um = new Cachorro();
        um.tamanho = 70;
        Cachorro dois = new Cachorro();
        dois.tamanho = 8;
        Cachorro tres = new Cachorro();
        tres.tamanho = 35;

        um.emitirSom();
        dois.emitirSom();
        tres.emitirSom();
    }
}
```

```
class Cachorro{
    int tamanho;
    String cor;

    void emitirSom(){
        if (tamanho > 60){
            System.out.println("Woof! Woof!");
        } else if (tamanho > 14){
            System.out.println("Ruff! Ruff!");
        } else {
            System.out.println("Yip! Yip!");
        }
    }
}
```

O que vimos até aqui

- A programação orientada a objetos permite estender um programa sem ser preciso mexer em código funcional já testado.
- Todo código Java é definido em uma classe.
- Uma classe descreve como criar um objeto desse tipo de classe. Uma classe é como um projeto.
- Um objeto conhece coisas e faz coisas.
- As coisas que um objeto conhece sobre si próprio se chamam variáveis de instancia. Elas representam o estado de um objeto.
- As coisas que um objeto faz são chamados de métodos. Eles representam o comportamento de um objeto.

Como os objetos se comportam. Como o estado e o comportamento estão relacionados.

Podemos enviar valores para um método

d.emitirSom(3);

Um método usa parâmetros. Um chamador passa argumentos.

Os argumentos são os valores que passamos para os métodos.

Como os objetos se comportam. Como o estado e o comportamento estão relacionados.

Mas aqui está a parte importante:

se um método usar um parâmetro, você terá que passar algo para ele. E esse algo deve ser um valor do tipo apropriado.

```
void emitirSom(int numeroDeLatidos){
    while (numeroDeLatidos > 0){
        System.out.println("Ruff!");
        numeroDeLatidos = numeroDeLatidos - 1;
    }
}
```

Como os objetos se comportam. Como o estado e o comportamento estão relacionados.

valores podem ser retornados por um método

Os métodos retornam valores. Todos os métodos são declarados com um tipo de retorno, mas até agora criamos todos os nossos métodos com o tipo de retorno void, o que significam que não retornam nada.

```
void ir(){

}
```

Podemos declarar um método que retorne um tipo especifico de valor para o chamador

```
int meRetorme(){
    return 42;
}
```

Encapsulamento

Até esse momento, cometemos uma das piores falhas na OO

Expor nossos dados!

Precisamos construir métodos de configuração para todas as variáveis de instancia em vez de acessar os dados diretamente.

O método ***Getter*** (de captura) e ***Setter*** (de configuração) permitirão que você capture e configure coisas.

Encapsulamento

Oculte os dados!

Modificadores de acesso public e private.

Marque as variáveis de instância com **private**.
Marque os métodos de configuração e captura com **public**.

Encapsulamento
Encapsulando a classe Cachorro

```
class Cachorro{
    private int tamanho;

    public int getTamanho(){
        return tamanho;
    }

    public void setTamanho(int s){
        tamanho = s;
    }

    void emitSom(){
        if (tamanho > 60){
            System.out.println("Woof! Woof!");
        } else if (tamanho > 14){
            System.out.println("Ruff! Ruff!");
        } else {
            System.out.println("Yip! Yip!");
        }
    }
}
```

```
class CachorroTeste{
    public static void main(String[] args){
        Cachorro um = new Cachorro();
        um.setTamanho(70);
        Cachorro dois = new Cachorro();
        dois.setTamanho(8);
        System.out.println("Cachorro um: " + um.getTamanho());
        System.out.println("Cachorro dois: " + dois.getTamanho();
        um.emitirSom();
        dois.emitirSom();
    }
}
```