PROGRAMAÇÃO ORIENTADA A OBJETOS
CONCEITOS

No dia a dia vemos diversos tipos de objetos. Nossa vida é rodeada de objetos. É natural que pensemos em objeto quando programamos. Mas os objetos do mundo real não são todos iguais. Existe um conceito que todos compartilham, mas cada um tem sua característica distinta. Na figura ao lado vemos pães de queijo, mas podemos notar que não são todos iguais. Eles se parecem, mas não são idênticos. Poderíamos considerar, inclusive, que fossem de origens (fabricantes) diferentes. Ou seja, podem ter gostos diferentes. Mas nem por isso deixam de ser pães de queijo.
Apesar de serem diferentes, compartilham de uma propriedade comum.

Poderíamos ter utilizado outro exemplo: maçãs. Apesar de não serem moldadas com formas de padeiro, foram moldadas com a forma da natureza. Esta forma define o que é uma maçã. Quando elas surgem, são únicas, com suas cores, tamanhos e sabores variados.

Em programação orientada a objetos também pensamos em formas. Quando pensamos em um objeto, pensamos em suas propriedades (atributos) e seus métodos (ações) que realiza ou estados em que se transformam. Uma maçã pode estar verde, madura, estragada, podre... pode ser do tipo verde ou do tipo vermelha... pode ser pequena, média ou grande.

Um livro também tem seus atributos: pequeno, médio ou grande; nacional ou estrangeiro; de ficção ou não-ficção; etc.

Você conseguiria pensar nos atributos e nas ações de um carro?

Para simplificar nosso raciocínio, vamos iniciar nossas reflexões com um objeto bem simples: uma lâmpada.

Uma lâmpada pode ser incandescente ou florescente. Por isso, devemos criar um atributo chamado **tipo**.

Ela, normalmente, tem uma tensão de trabalho. Pode ser **110v** ou **220v**. Assim, outro atributo importante é a **tensão**.

Ela também tem uma **potência** de trabalho. Pode ser de **40w**, **60w**, **100w**, **120w** etc.

Uma lâmpada pode estar em 3 **estados**: **ligada**, **desligada** ou **queimada**. E pode ter duas **aparências**: **acesa** ou **apagada**.

Podemos executar as ações de trocar uma lâmpada, apaga-la ou acende-la.

Para simplificar nossa reflexão, assumiremos que sempre existe energia.

Lâmpada
Tipo Tensão Potência Estado Aparência
Trocar Apagar Acender

O quadro ao lado representa a classe lâmpada. Servirá como modelo ou forma para gerar outras lâmpadas. Elas não serão idênticas (assim como não são idênticos os pães de queijo ou as maçãs...) mas respeitarão as definições definidas em seu modelo (classe).

Quando uma lâmpada é gerada, ela é única. Você pode, inclusive, atribuir um nome para ela:

Lâmpada Josie;

A Josie (nossa lâmpada) é uma concretização do modelo que definimos. Tem seu tipo, sua tensão, sua potência etc.

Quando criamos outra lâmpada (Rosie), ela poderá ter características diferentes das de Josie. Nem por isso deixará de respeitar as definições estabelecidas por sua classe.

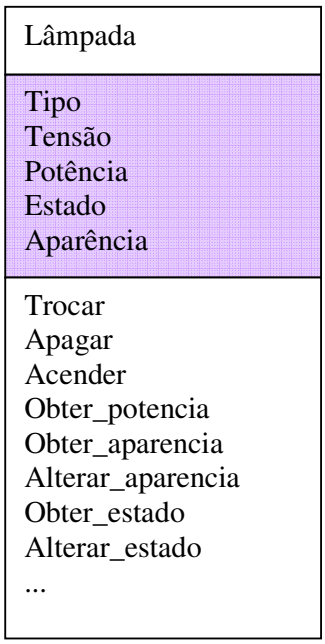
Cada vez que criamos algo que respeite as definições da classe, na verdade criamos um objeto. Assim, Josie é um objeto (uma instância) da classe Lâmpada. Rosie é outra instância (outro objeto).

Se os atributos da classe não forem protegidos (private ou protected) um programador poderá altera-los quando desejar. Mais, ainda, ele poderá atribuir qualquer valor para um atributo.

Para acessar um dado de uma classe, utilizamos o operador . (ponto), da mesma forma que fazíamos quando utilizávamos structs em C. Assim, com dados não protegidos, poderíamos ter o seguinte comando:

```
Rosie.potencia = -50;
```

Isso não existe, mas o programador não pensou nesta possibilidade quando elaborou sua classe. Mas você não quer que isso aconteça, quer? Para proteger seus dados é preciso defini-los como private ou protected (no quadro ao lado, representamos isso com uma cor diferente.



Mas isso impedirá que acessemos os dados diretamente (que é exatamente o que queríamos). Assim, como fazemos para acessa-los (saber quais valores possuem) e/ou modifica-los?

A resposta é simples: criamos métodos (ações) que realizam tais operações. No diagrama ao lado criamos alguns métodos adicionais àqueles que havíamos criado originalmente. Agora é possível saber a potência da lâmpada Rosie através da chamada de um método:

```
P = Rosie.Obter_potencia( );
```

P armazenará o valor da potência de Rosie. E a potência de Josie? O processo é semelhante, mas o valor armazenado não precisa, necessariamente, ser o mesmo! Afinal de contas, são lâmpadas diferentes!

Para acender a Josie, podemos acionar o método acender:

```
Josie.acender();
```

Isso fará mais sentido dentro de uma estrutura condicional ou um loop:

```
Se ! Josie.acender( ) então
      Josie.trocar();
Fim_se
```

O método acender deverá, primeiramente, verificar se a lâmpada está ligada (conectada à linha de tensão). Se a lâmpada não estiver conectada, nada deverá acontecer.

Se a lâmpada estiver conectada à linha de tensão (ligada), então poderemos alterar seu estado ou sua aparência (se estiver apagada). A transformação da lâmpada para o estado queimada pode ser feita de forma aleatoria (através de um sorteio). Para tanto, estipulamos um sorteio de um valor entre 0 e 100 e se o valor for menor que 5 (por exemplo), então a lâmpada poderá se queimar. Caso contrário, deverá alterar sua aparência para acesa:

MÉTODO ACENDER

```
Se estado == "ligada" então
        Se aparência == "apagada" então
                N = sorteio(0..100)
                Se N < 5 então
                        estado = "queimada"
                Senão
                        aparência = "acesa"
                Fim_se
        Fim_se
Fim_se
```

O método acender pode acessar os atributos de sua classe diretamente. Assim, com uma maneira de proteger seus dados o programador que criou a classe lâmpada garante sua qualidade e qualquer outro programador que fizer uso da classe não terá receio em utiliza-la.