1. Em linguagens de programação, o termo *objeto* tem um significado específico. Objeto é um tipo de registro que também contém funções.

Em geral, as linguagens de programação têm tanto instruções para declarar e usar estruturas de dados quanto instruções para declarar e usar funções. Nos programas, as declarações das estruturas de dados vêm separadas das declarações das funções. (Obviamente, há uma conexão entre elas na lógica de programação, pois as funções refletem as estruturas.)

DADOS, REGISTROS, ESTRUTURAS

FUNÇÕES, PROCEDIMENTOS

Assim, as funções têm as estruturas de dados como um de seus parâmetros. Exemplo:

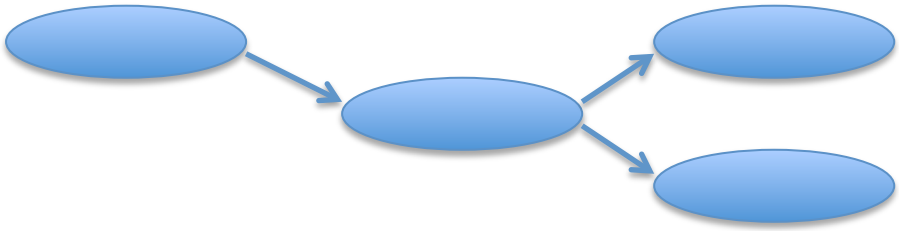
```
push(stack, 13.8)
```

No caso de um objeto, a função lhe pertence, e é acessada como um campo:

```
stack.push(13.8)
```

2. Nomenclatura. Essa programação é chamada de *orientação a objetos*. Funções são chamadas de *métodos* em grande parte das linguagens de programação orientadas a objetos. Através dos seus métodos, o objeto manifesta *comportamento*. Quando um objeto invoca um método em outro objeto, diz-se que um envia uma *mensagem* para o outro. Os dados internos de um objeto estão organizados em *campos*.

3. Programar segundo a orientação a objetos consiste em criar objetos (ou seja, definir campos e métodos) chamando métodos de outros objetos. Um programa orientado a objetos consiste num ecossistema de objetos que invocam métodos uns dos outros (mandam mensagens).



4. Java é uma linguagem de programação orientada a objetos. O mecanismo para definição de objetos em Java é a *classe*. A definição de uma classe parece com a definição de uma *struct* em C, com a inclusão da declaração dos métodos. O exemplo a seguir é a declaração da classe Pessoa:

```
public class Pessoa {
	public String nome;
	public String cpf;
	public int idade;

	public Pessoa () {

	}

	public boolean obrigadoAVotar() {
		return (idade >= 18) && (idade < 70);
	}
}
```

Um objeto da classe Pessoa contido na variável `p1` possui dados para os campos `nome`, `cpf` e `idade`, e o método `obrigadoAVotar` pode ser chamado para ele através da instrução

```
p1.obrigadoAVotar()
```

O qualificador `public` libera o acesso dos campos e métodos a qualquer outro objeto. Há um mecanismo de controle de acesso.

O que parece ser um método, chamado Pessoa, que não retorna valores, é, na verdade, um *construtor*. Construtores são usados para criar um novo objeto, através da palavra `new`. No exemplo, o construtor nada faz, mas esse não é o caso comum.

Classes são tipos em Java. Para declarar uma variável que deve receber um objeto, é preciso indicar o tipo dela como sendo a classe do objeto. Por exemplo,

```
Pessoa p1;
```

É mais comum incluir a criação do objeto junto com a definição da variável, invocando um construtor com `new`:

```
Pessoa p2 = new Pessoa();
```

A classe funciona como uma espécie de molde para os objetos do seu tipo, como uma "lei geral" daqueles objetos. Os objetos são ditos *instâncias* da classe.

5. Uma classe pode ser definida com base em uma outra classe existente. Diz-se que a nova classe *herda* os campos e os métodos da outra classe (*herança*). A nova classe introduz modificações como novos campos ou métodos ou mudança dos métodos existentes (*sobrescrever*). No exemplo abaixo, a classe Funcionário é definida a partir da classe Pessoa, adicionando um campo para salário e um método para pagamento:

```
public class Funcionario extends Pessoa {
	public float salario;

	public Funcionario() {
		super();
	}

	public float pagamento() {
		return salario;
	}
}
```

A palavra reservada `extends` sinaliza que a classe está sendo criada pelo mecanismo de herança. Quando ela não é utilizada, está implícito que a classe herda de `java.lang.Object`, a classe a partir da qual todas as demais classes são criadas. Ou seja, a sentença `public class Pessoa {...` na verdade é `public class Pessoa extends java.lang.Object{...` (`lang` é um subpacote do pacote `java`. Em Java, as classes são organizadas em pacotes.)

A classe Funcionário é dita uma subclasse da classe Pessoa. Inversamente, Pessoa é a superclasse de Funcionário.

A primeira ação no construtor de classe criada por herança é invocar o construtor da sua superclasse através da palavra `super`.

Mais um exemplo:

```
public class Vendedor extends Funcionario {
	public float comissao;

	public Vendedor () {
		super();
	}

	public float pagamento() {
		return salario + comissao;
	}
}
```

O mecanismo de herança fica claro na lista de campos e métodos disponíveis para um objeto da classe Vendedor:

Campos: `nome`, `cpf`, idade (de Pessoa), `salario` (de Funcionário) e `comissao`.

Métodos: `obrigadoAVotar` (de Pessoa), `pagamento` (de Vendedor).

O método pagamento de Vendedor sobrescreveu o de Funcionário.

6. Um método marcado com a palavra `static` deixa de ser um método de instância e passa a ser um método de classe. Métodos de instância são o caso comum em orientação a objetos, em que o método se aplica a um objeto, como no exemplo `p1.obrigadoAVotar()`. Métodos de classe parecem com funções em C, isto é, apenas os parâmetros são relevantes para o entendimento da ação da função. Por exemplo, a classe Contador:

```
public class Contador {
	public Contador () {

	}

	public static int conta(java.util.List<String> lista) {
		return lista.size();
	}
}
```

contém o método `conta`. Usos permitidos e não permitidos de `conta`:

```
q1 = c1.conta(lis1);   // ERRADO!!!
q2 = Contador.conta(lis2);  // Correto
```

A presença do nome da classe no início da expressão indica ela funciona como uma espécie de contêiner (um módulo, uma biblioteca) no qual o método está definido.

Exemplo baseado em classes da API Java (Application Programming Interface):

```
public class Integer {
...
      public static int compare(int x, int y) ...

      public int compareTo(Integer anotherInteger) ...
}
```

Antes de prosseguir com o exemplo, é preciso explicar que em Java os tipos básicos da linguagem (`boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short`), chamados tipos primitivos, possuem equivalentes como objetos. Ou seja, é possível representar um número por dois caminhos. O primeiro caminho é similar ao que se faz em C. O segundo caminho é usar uma classe já existente para o tipo em uso. Esse esclarecimento se fez necessário porque o método `compare` usa o tipo primitivo `int` nos parâmetros, e compareTo usa um objeto da classe `Integer`. Mas não é essa a diferença relevante aqui.

```
int i = 10;   // tipo primitivo
int j = -3;   // tipo primitivo
System.out.print( Integer.compare( i , j ) ); //  metodo de classe

Integer k = new Integer(10);  // objeto
Integer m = new Integer(-3);  // objeto
System.out.print( k.compareTo( m ) ); // metodo de instancia
```

Os dois blocos são equivalentes. Mas `compare` se parece mais com uma função em C.

7. Uma classe é executável se ela contém o método de classe `main` como segue:

```
public static void main(String[] args) {
      ...
}
```

Dessa forma, a classe pode ser invocada diretamente pela máquina virtual Java, que o método `main` passa a ser executado imediatamente. (Em Alice, seria o método `myFirstMethod`).

8. Polimorfismo. Considerando um método hipotético `swimm()` definido em uma classe chamada `Swimmer` é sobrescrito nas subclasses `Fish` e `MarineMammal`. (Os peixes batem a cauda na vertical e os cetáceos, na horizontal.)

À primeira vista, tudo se passa como se houvesse dois métodos diferentes com o mesmo nome. Visto de outra forma, pode-se dizer que, na verdade, há uma ideia única (isto é, nadar) realizada de duas formas diferentes. Daí o termo *polimorfismo*.