Não é uma linguagem OOP pura, porque as variáveis podem ter valores primitivos ou ser referenciadas como objetos.

Não há apontadores, mas variáveis primitivas são guardadas como valores e objetos são guardados como referências.

Literals: são representações sintáticas de variáveis (Boolean, Character, String, Integer, ...).

Operador == compara tipos primitivos pelo seu valor, mas compara objetos pela sua referência.

Input/Output:

- System.out.println (...);
- Scanner scanner = new Scanner(System.in);
  String line = scanner.nextLine();

Strings de Java são imutáveis. Para as comparar usa-se o método equals().

OOP: providencia uma abstração onde os elementos do problema são objetos no espaço solução; permite descrever o problema em termos do problema.

Pilares da Orientação por Objetos (A PIE)

- Data **A**bstraction: separação entre interface pública de um tipo de dados e a sua implementação
- **P**olymorphism: um único símbolo pode representar muitos tipos
- **I**nheritance: objetos podem herdar propriedades e comportamentos de outros objetos
- **E**ncapsulation: acesso restrito a algumas componentes de um objeto

Visibilidade:

- Classes:
  - public
  - protected (mesmo package)
  - private (dentro de outras classes)
- Variáveis e Métodos:
  - public
  - protected
  - private
  - package

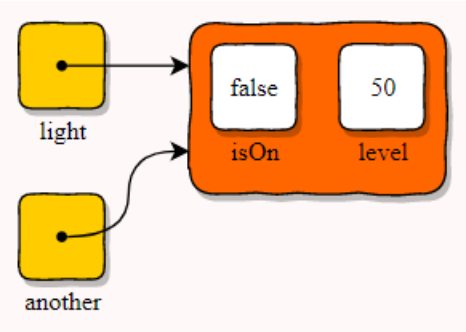

Figura 1: Criação de novo objeto

Para criar um novo objeto é necessário usar new:

```
Light light = new Light();
Light another = light;
```

Para termos duas instâncias do mesmo objeto, a classe tem que implementar a interface Cloneable e o seu método clone();

final: variável que não pode ser alterada.

Herança deve ser usada para estabelecer uma relação de is-a (é um/a). Só é possível estender uma classe.

Métodos final, static e private não podem ser reescritos.

Deve-se reescrever o método equals() para todas as classes que vão ser comparadas, o método hashCode() quando usamos HashSet e o método toString().

O bloco finally executa sempre que se sai de um bloco try. Assegura que este código é sempre corrido mesmo que ocorra uma exceção.

```
try {
    ...
} catch {
    ...
} finally { ... }
```

Coleções: Set, List, Queue, Deque, Map

- São parameterizadas
- Exemplo: List<Animal> list = new ArrayList<>();
  - Princípio "Retorna o tipo mais específico, aceito o tipo mais genérico"