VARIÁVEIS E ESTRUTURAS DE CONTROLO

VARIÁVEIS

- Uma variável pode ser vista como um espaço em memória onde um valor de determinado tipo (p.e., inteiro) pode ser guardado

- As variáveis têm três características
  - Nome
  - Tipo
  - Valor

```
int n = 7;
```

- A definição de uma variável indica o seu **tipo**, o seu **nome** e o seu **valor inicial**

VARIÁVEIS: ATRIBUIÇÃO

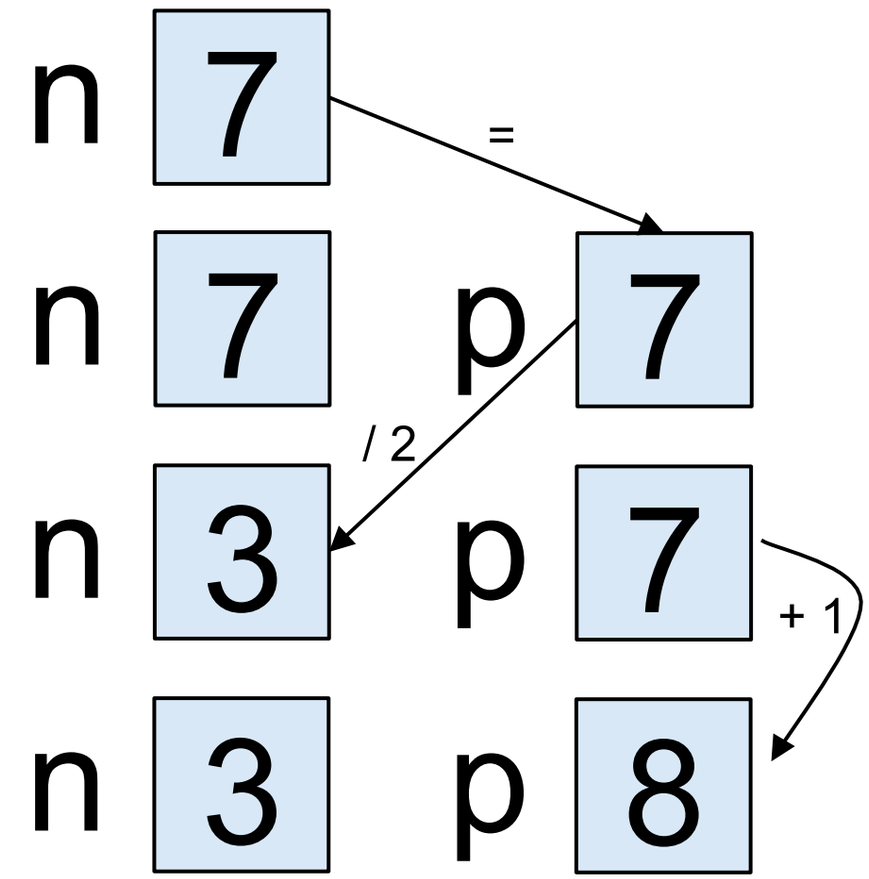
- O valor das variáveis é definido usando o operador de atribuição(=)

```
int n = 7;

int p = n;

n = p / 2;

p = p + 1;
```



INSTRUÇÕES E BLOCOS DE INSTRUÇÕES

- Uma **instrução** é uma ação na execução do programa que pode mudar o seu estado (variáveis). Em Java, uma instrução termina sempre com um ponto-e-vírgula (`;`)

```
int n = 0;
int p = 0;
```

- Um **bloco de instruções** é um conjunto de instruções entre chavetas que será executado sequencialmente (p.e., numa função)

```
{
    n = n + 1;
    p = n % 2;
}
```

ESTRUTURAS DE CONTROLO

- Uma estrutura de controlo é um elemento de um programa que controla a execução de instruções, por exemplo:

  1. Executar uma instrução caso se verifique determinada condição
  2. Executar uma instrução caso se verifique determinada condição e outra se não se verificar essa condição
  3. Executar uma instrução um determinado número de vezes
  4. Executar continuamente um conjunto de instruções enquanto determinada condição se verifica

- De grosso modo, as estruturas de controlo dividem-se em duas categorias: **seleção** (1 e 2) e **repetição** (3 e 4)

ESTRUTURA DE SELEÇÃO (*IF-ELSE*)

- Uma estrutura de seleção permite condicionar a execução de uma ou mais instruções em função de determinada condição booleana (verdadeiro/falso)

- Em Java, a estrutura de seleção mais comum é o *if-else*

```
if(condição) {
    instrução;
    …
}
```

- Caso a ***condição*** se verifique (i.e., é verdadeira), então o bloco de instruções é executado

ESTRUTURA DE SELEÇÃO (*IF-ELSE*)

- A utilização do *else* é opcional

```
if(condição) {
    instrução;
    …
} else {
    instrução;
    …
}
```

- Caso a *condição* **não** se verifique (i.e., é false), então o bloco de instruções a seguir à palavra *else* é executado

ESTRUTURA DE SELEÇÃO (*IF-ELSE*)

- A estrutura *if-else* pode ser encadeada

```
if(condição) {
    …
} else {
    …
    if(condição) {
        …
    } else {
        …
    }
    …
}
```

EXEMPLO (*IF-ELSE*)

- Função que devolve o mínimo entre dois valores inteiros

```
static int min(int a, int b) {
    if(a < b) {
        return a;
    } else {
        return b;
    }
}
```

ESTRUTURAS DE REPETIÇÃO: O CICLO *WHILE*

- Uma estrutura de repetição permite executar um bloco de instruções enquanto determinada condição é verdadeira

- A estrutura de repetição mais elementar é o ciclo *while*

```
while(condição) {
    instrução;
    …
}
```

- Enquanto a ***condição*** se verificar (i.e., for verdadeira), o bloco de instruções é executado

EXEMPLO (*WHILE*)

- Função que devolve o resultado da multiplicação de dois números inteiros (sem recorrer ao operador *)

```
static int produto(int a, int b) {
    int resultado = 0;
    int somasPorFazer = b;
    while(somasPorFazer != 0) {
        resultado = resultado + a;
        somasPorFazer = somasPorFazer - 1;
    }
    return resultado;
}
```

ESTRUTURAS DE REPETIÇÃO: O CICLO *DO-WHILE*

- O ciclo *do-while* é uma variante do ciclo *while*, onde **o bloco de instruções é executado pelo menos uma vez**. Primeiro o bloco de instruções é executado, e só depois a condição é verificada

```
do {
    instrução;
    …
} while(condição);
```

- O bloco de instruções é executado uma vez. Enquanto a ***condição*** se verificar (i.e., for verdadeira), o bloco de instruções é executado

CICLOS INFINITOS

- Ao utilizar estruturas de repetição um programa válido pode incorrer em situações de **ciclo infinito**. Tal acontece quando o bloco de instruções de uma estrutura de repetição é **executado um número indefinido de vezes (infinitas)**, como consequência da *condição* do ciclo ser sempre verdadeira

```
int n = 2;
while(n%2 == 0) {
   n = n + 2;
}
```

A *condição* (que verifica se *n* é par) é sempre verdadeira e o bloco de instruções é executado infinitamente. Logo, as instruções do bloco de instruções têm que garantir que o ***progresso*** do ciclo é tal que a determinada altura a *condição* se torna falsa.

PAPÉIS DAS VARIÁVEIS: ACUMULAÇÃO

- Um padrão comum na forma de utilizar variáveis consiste em efectuar **acumulações**. Por exemplo:

```
static int produto(int a, int b) {
    int resultado = 0;
    int somasPorFazer = b;
    while(somasPorFazer != 0) {
        resultado = resultado + a;
        somasPorFazer = somasPorFazer - 1;
    }
    return resultado;
}
```

O papel da variável `resultado` é guardar um valor que vai sendo **acumulado** ao longo da execução do ciclo.

PAPÉIS DAS VARIÁVEIS: CONTAGEM

- Um padrão comum na forma de utilizar variáveis consiste em efectuar **contagens**. Por exemplo:

```
static int produto(int a, int b) {
    int resultado = 0;
    int somasPorFazer = b;
    while(somasPorFazer != 0) {
        resultado = resultado + a;
        somasPorFazer = somasPorFazer - 1;
    }
    return resultado;
}
```

O papel da variável `somasPorFazer` é guardar um valor que representa o **número de vezes** que o parâmetro `a` deve ser acumulado em `resultado`. Quando `somasPorFazer` chega a zero, já não se devem realizar mais acumulações.

A RETER

- Variáveis
  - Tipo
  - Nome
  - Valor
- Estruturas de controlo
  - Seleção
  - Repetição
- Papeis das variáveis
  - Acumulação
  - Repetição