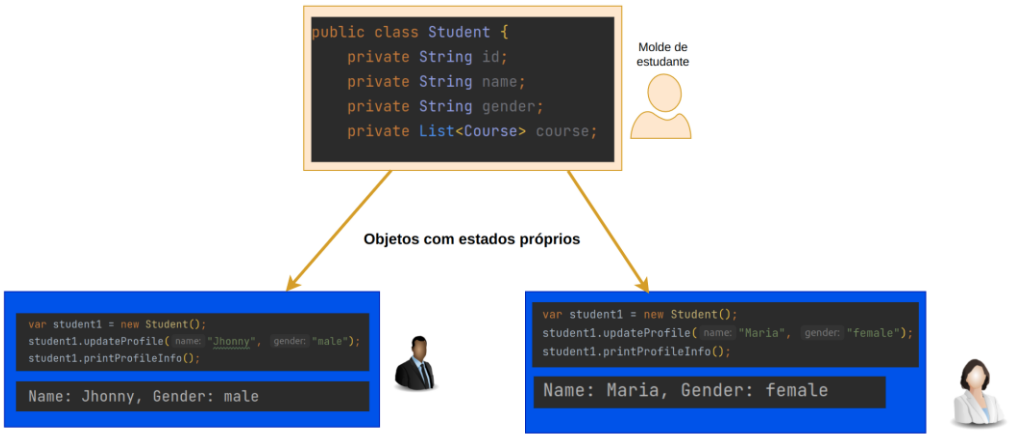
Classes e Objetos – Java

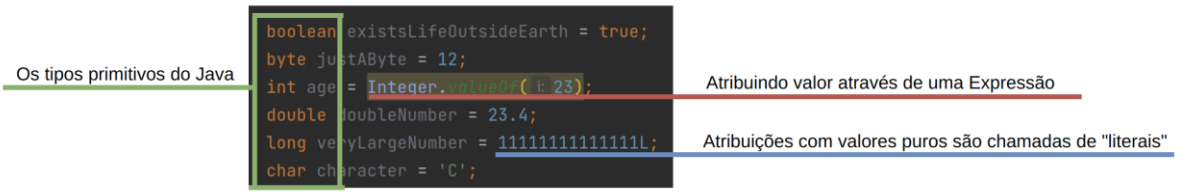
Classes e **Objetos** são **conceitos diferentes embora tenham uma relação direta.** Uma **Classe representa a forma como um objeto é representado**, já um **objeto é uma instância concreta de uma classe com seu próprio estado:**



*"Pra fazer um bolo em formato de coração você usa uma **forma(classe)** em formato de coração, você vai comer o **bolo(objeto)** que pode ter qualquer **sabor(estado)** mas foi feito na forma de coração".*

Sintaxe de variáveis – Java

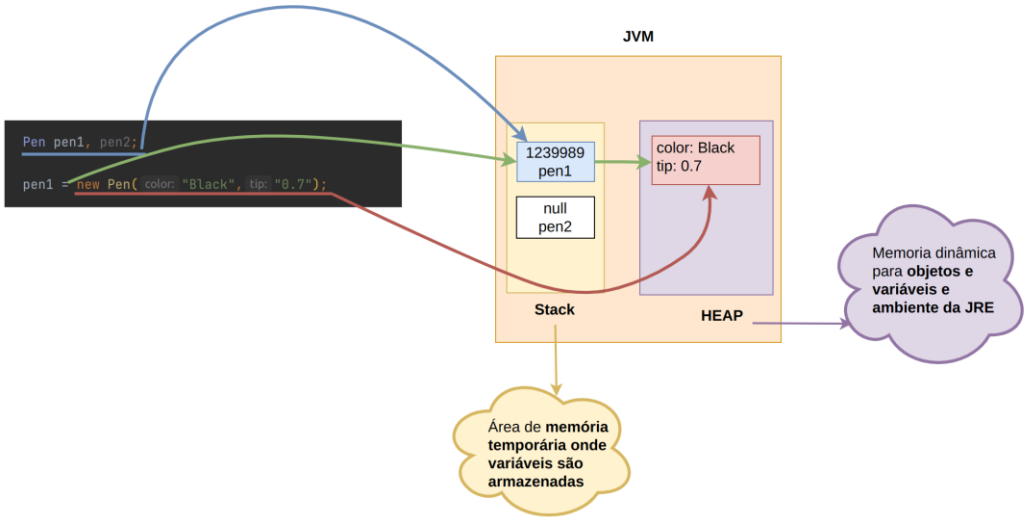
O estado dos objetos é representado por variáveis. **Variáveis em java são estaticamente tipadas**, ou seja, uma vez definidas seu tipo, não pode ser sobrescrito com outro tipo. Por exemplo, tentar atribuir um Integer a uma variável tipada como String resultaria em erro.



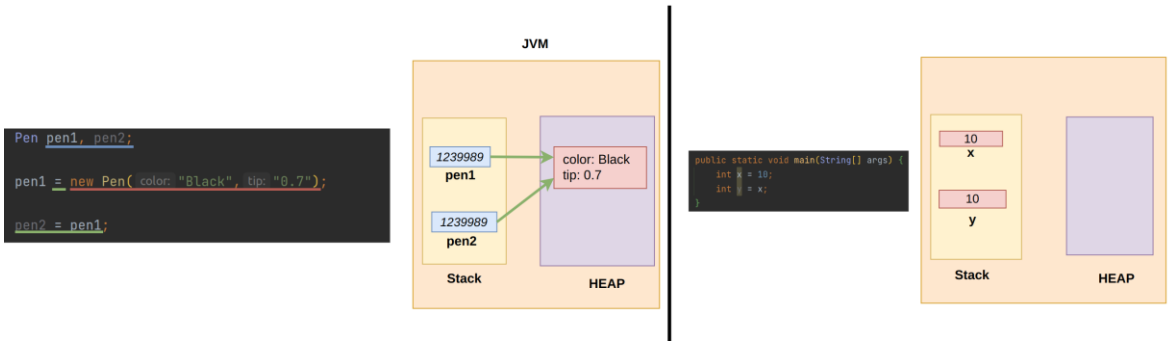
Tipos primitivos: Primitivo nesse cenário significa que é um dado que não pode ser decomposto em algo menor, eles são o tipo mínimo e a partir deles outros tipos são construídos. Outro pensamento pode ser de que um **tipo primitivo não é construído a partir de outros objetos, ele simplesmente é construído pelo valor que ele representa**.

Referência e valor – Java

Um tipo primitivo não aponto para alguma coisa, ele simplesmente é um valor guardado na memória, variáveis de objetos complexos são referências apontando pra um objeto. Então **quando você declara uma variável de objeto no Java, na verdade você está fazendo uma referência:**



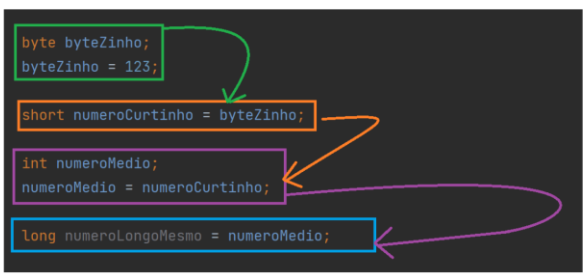
Se eu fazer **minha segunda referência de objeto receber a primeira referência** eu **vou estar apontando PARA O MESMO ENDEREÇO que aponto para o objeto**, se eu **fizer isso com um tipo primitivo eu vou criar uma cópia**. Isso ajuda a entender ainda mais a dinâmica de variáveis de referência e "valor":

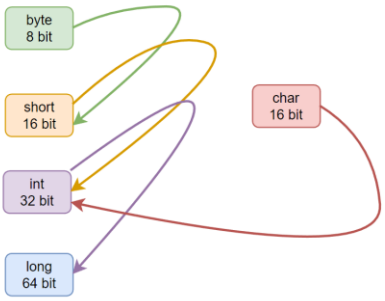


Casting – Java

Casting é um modo de você **converter uma variável de um tipo para outro**. Nos tipos primitivos, números só podem ser "castados" para outros números, não dá pra fazer casting em boolean.

Os tipos ***primitivos numéricos*** são um bom exemplo de casting implícito, que funciona bem devido ao tamanho **deles**:





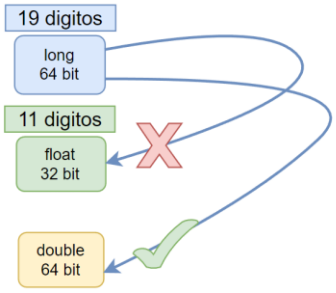
Um tipo **long pode ser castado para float ou double**. No caso do double é ok, mas pra float pode ser perigoso:

```
long numeroLongo = 1212312312224233412L;

float numeroQuebradoCurto = numeroLongo;

System.out.println(numeroQuebradoCurto);
1.21231231E18

Process finished with exit code 0
```



O caminho inverso dessa situação **requer castings explícitos**, mas por alguma razão parece loucura fazer um tipo maior caber um menor (e de fato é), e isso pode gerar perda de informação:

```
long numeroLongo = 1212312312224233412L;
int numeroMedio = (int) numeroLongo;
short numeroCurto = (short) numeroMedio;
char caractere = (char) numeroCurto;
byte numeroPequeno = (byte) caractere;

double decimalLongo = 231.321321323231231232;
float decimalMedio = (float) decimalLongo;
long numeroIncrivelmenteLongo = (long) decimalMedio;
```

```
public static void main(String[] args) {
    byte byteEsquisito = (byte) 1234566;
    System.out.println(byteEsquisito);
}
-122
```

out of range: Um **byte tem um range entre -128 e 217**. Obviamente 1234566 não vai caber, então ele encaixa o que cabe (os primeiros 8 bits) e descarta o resto.

Ao **converter tipos flutuantes para inteiros, você perde as casas decimais**. Se precisar **fazer uma divisão entre inteiros** e as casas decimais forem importantes, então **certifique-se de fazer o casting**:

```
public static void main(String[] args) {
    float numeroQuebrado = 12.5f;
    float numeroQuebrado2 = 3.6f;

    int numeroInteiro = (int) numeroQuebrado;
    int numeroInteiro2 = (int) numeroQuebrado2;

    System.out.println("Numero inteiro: "+numeroInteiro+" | Numero inteiro2: "+numeroInteiro2);
}
Numero inteiro: 12 | Numero inteiro2: 3
```

```
public static void main(String[] args) {
    int x = 10; int y = 3;
    double divisaoSemCasting = x/y;

    double divisaoComCasting = (double) x / y;

    Resultado sem casting: 3.0 | Resultado com casting: 3.3333333333333335
}
```