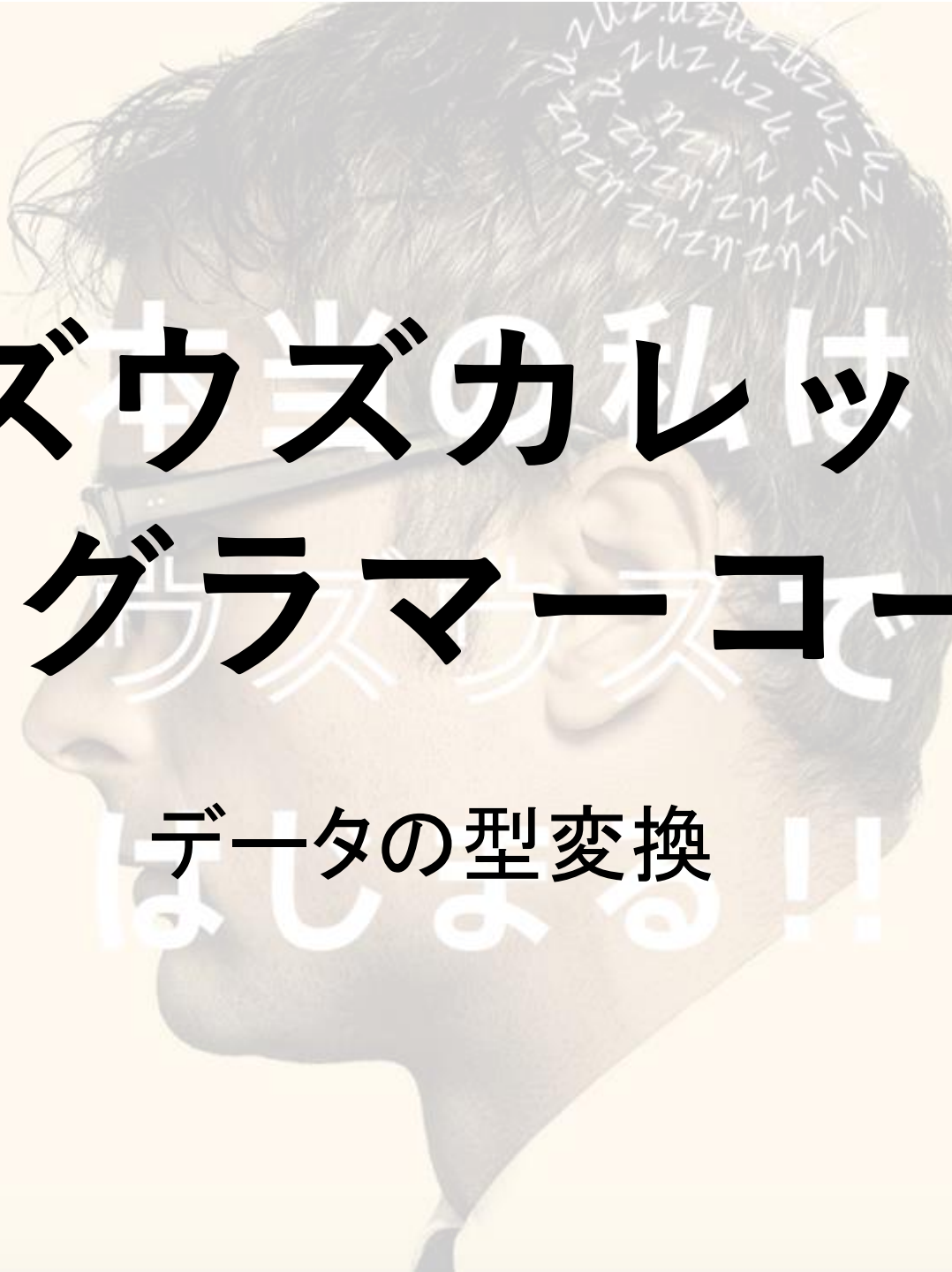


# ウズウズカレッジ
# プログラマーコース

## データの型変換



## ≪代入・算術演算子の活用ルール≫

□代入において、**代入する値の型と代入先の変数の型は必ず同じである必要があります。**
型に食い違いがある場合、一部例外を除いてコンパイルエラーとなります。

□代入は同じ型の**変数同士**で行われる処理です。
マジックナンバーはそのまま値を代入しているわけではなく、**一度対応する型の変数※に格納されてから代入処理が行われています。**
※125→int型、125L→long型、”UZUZ”→String型・・・

□算術演算子の活用においても、代入と同様に**同じ型の変数同士でしか足したり引いたりすることができません。**

復習

マジックナンバー

```
int teika = 1000;
```

整数は暗黙的にint型

▼マジックナンバーと対応する型

| データ種類 | 書式 |
|---|---|
| int | 5000000 |
| long | 3500000000L<br>※値の末尾に L を付ける |
| double | 6.05 |
| float | 1.35F<br>※値の末尾に F を付ける |
| char | '土'<br>※シングルコーテーションで囲う |
| boolean | true あるいは false<br>※文字列に見えるがダブルコーテーションは使わない |
| String | "Hello World!!"<br>※ダブルコーテーションで囲う |

## ≪変数⑤（マジックナンバー）≫

□プログラム中に記述された具体的な値のことを
マジックナンバーと呼びます。

□整数のマジックナンバーは**暗黙的にint型**として扱われます。

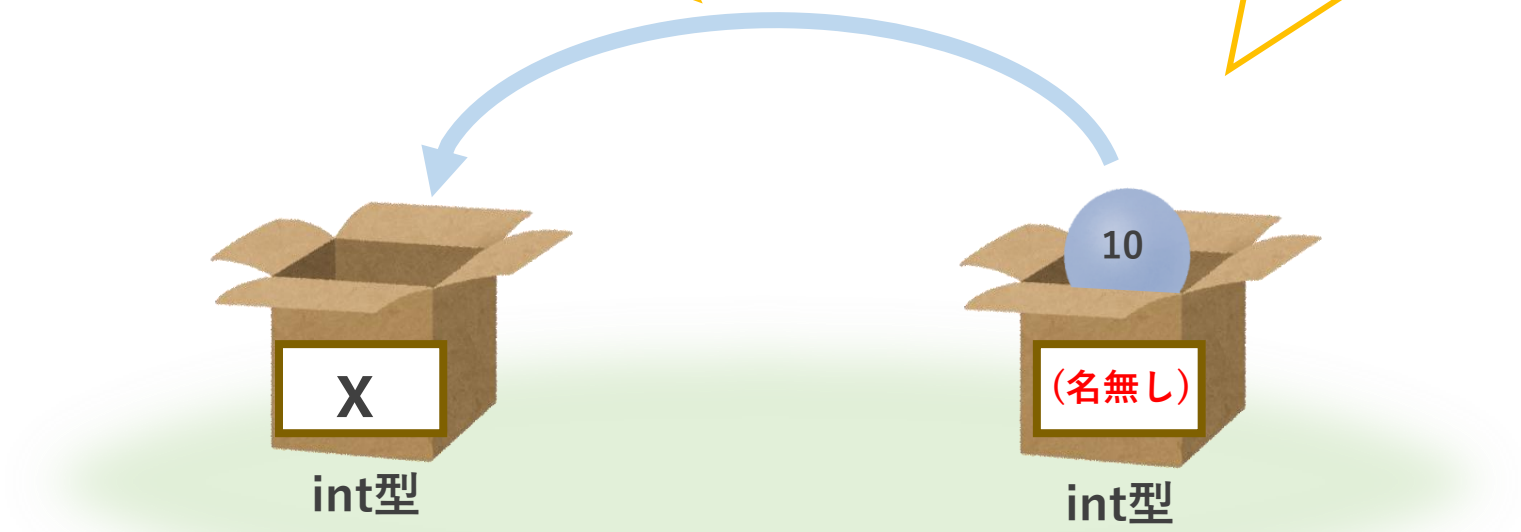
□小数点数のマジックナンバーは**暗黙的にdouble型**として
扱われます。

# ～マジックナンバーのしくみ～

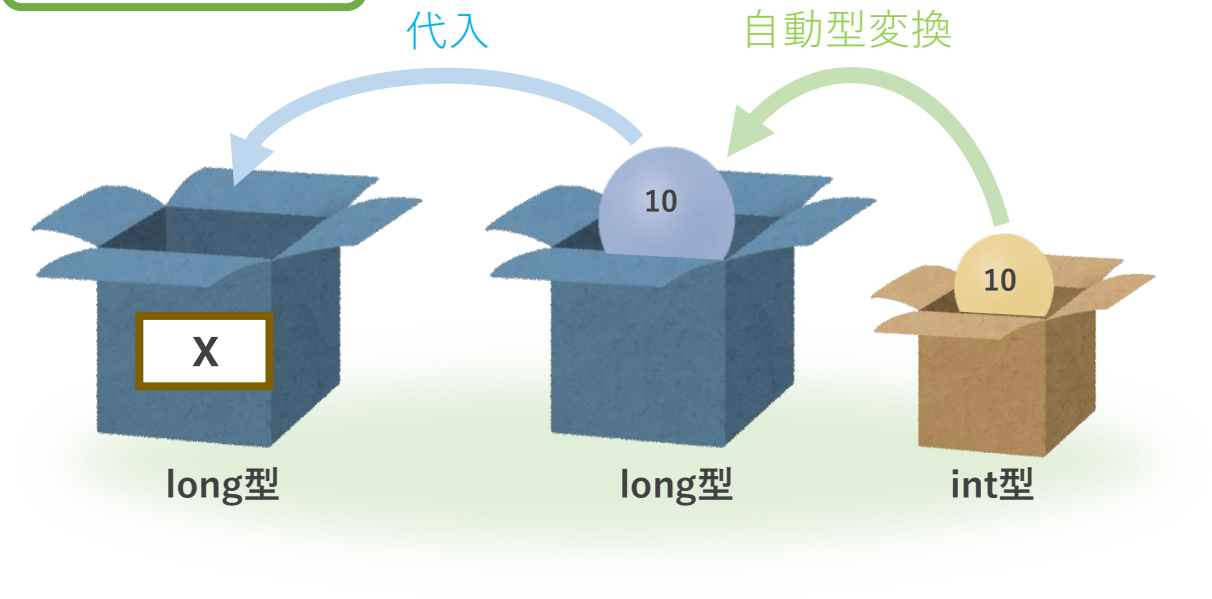
```
int x = 10 ;
```

**STEP2**
マジックナンバーを格納した
名無しの変数から左辺の変数に
代入処理を行う

**STEP1**
マジックナンバーに対応する
名無しの変数を生成し、
そこにマジックナンバーを格納する



long x = 10 ;




Int y = 10L ;


## ≪代入における自動型変換≫

□代入処理において以下の条件を満たしている場合、
代入元の値の型と代入先の型に食い違いがあっても
**内部的に自動型変換が行われて**代入が成功します。

・代入元と代入先が共に整数型（byte ,short ,int ,long）
あるいは小数点数型（double ,float）である。

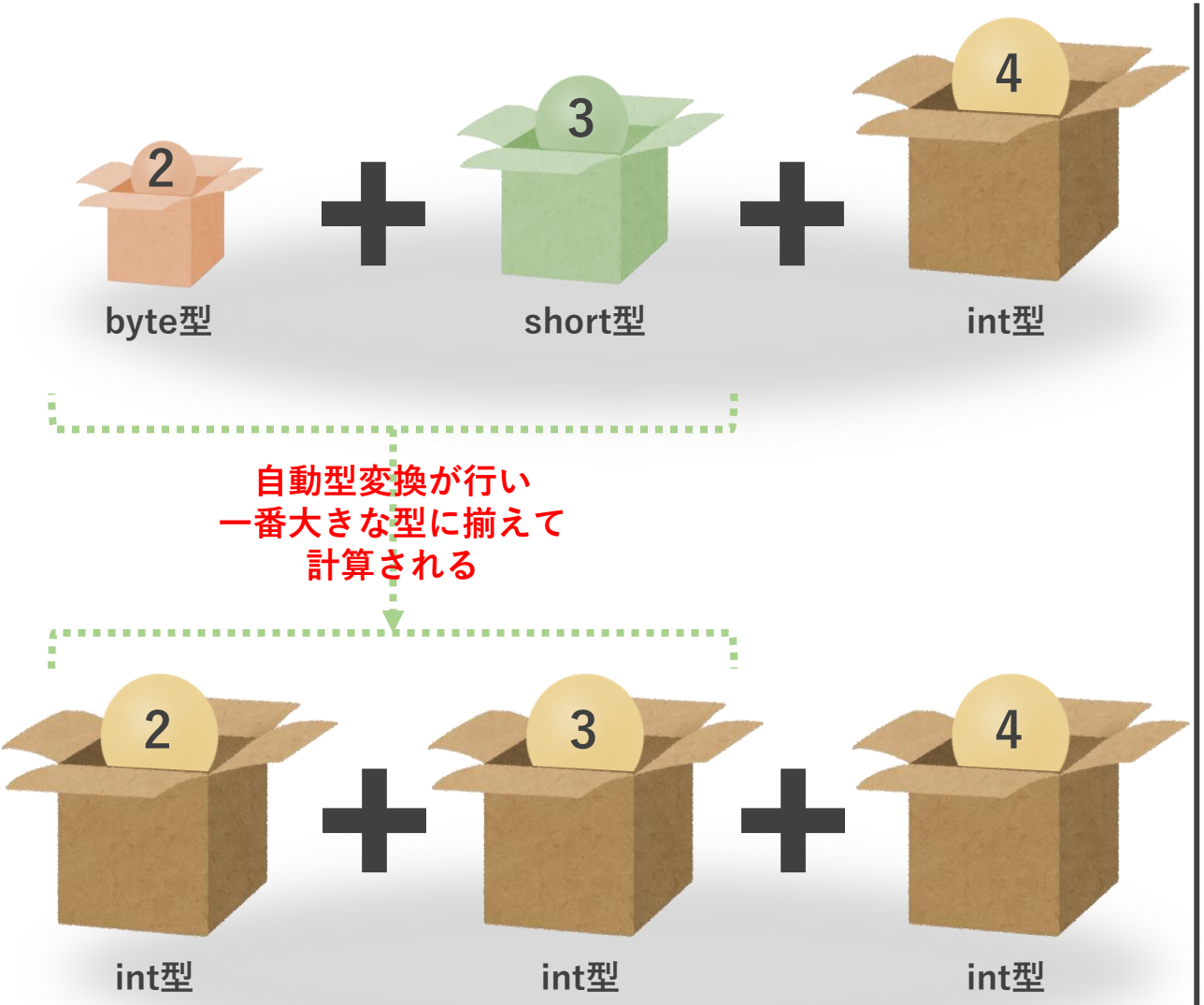
・変数の大きさ（許容メモリサイズ）が以下を満たす。
**代入先の型（左辺）＞ 代入元の型（右辺）**

□代入における自動型変換は以下の順序で行われます。

①代入先の型と同じ型の変数を作り、代入元のデータを
詰め替える。（**この工程が自動型変換**）

②詰め替えた変数を代入元とすることで同じ型同士の
代入処理が行われる。

□許容メモリサイズの小さな変数に大きな型のデータを
代入しようとすると**桁あふれ（オーバーフロー）**が
起きてエラーになります。
小さい箱に大きい箱は入らないのと同じです。




## ≪算術演算子の活用における自動型変換≫

□算術演算子で扱われる変数の型が食い違っていても
多くの場合**内部的に自動型変換が行われて**演算が成功します。

**＜演算時の自動型変換ルール＞**

・整数型同士であれば一番大きな型に揃えて演算される

・doubleを含む小数点数の演算はdoubleに揃えて演算
整数型が混じっていてもdoubleに揃えられる

・文字列を含む足し算は文字列の連結となる

## ≪キャスト≫

□**キャスト演算子**を使うと**一時的な型変換**を行うことができます。
例えばlong型でもint型でも10という数値を扱うことが可能です。
本来であればオーバーフローしてしまうlong型からint型への代入も、
キャスト演算子を使用してlong型データを一時的にint型として扱うことで代入を成功させることが可能です。
キャスト演算子は以下のように記述します。

**( 変換したい型 )変換したい変数名**

□キャスト演算子は文字⇔整数のような種類違う型変換には使用できません。

**キャストの使用例**

```
long x = 10 ;
int y = (int)x ;
```

**キャスト**
**一時的にint型として扱う**

## ≪文字⇔整数の型変換≫

□文字型⇔整数型など、データの種類が違う型の変換を行いたい場合はJavaが提供する特殊なメソッドを用います。

**文字列→整数：Integer.parseInt( int型として扱いたい文字列 )**

**整数→文字列：String.valueOf( String型として扱いたい整数 )**

**文字列→整数**

```
String x = “10” ;
int y = Integer.parseInt( x ) ;
```

**整数→文字列**

```
int x = 10 ;
String y = String.valueOf( x ) ;
```

＜演習：Ex1_07_1＞

以下、 どのようなデータが画面に表示されるでしょう？
プログラムを実行せずに予想してみましょう。

（ ）の中心から
考えていこうね！

```
int a = 100 ;
double b = 1.5 ;
String c = "7" ;
double d = 1.2 ;

System.out.println( Integer.parseInt( ( int )( a + b ) + c ) + d  );
```

# ＜演習：Ex1_07_2＞

```
/*-< 演習：Ex1_07_2 >-----------------------------------
アンダーバーに適切な処理を埋め、デバッグを行うことで
ソースコードが正常に通るよう修正してください。
-------------------------------------------------------*/
class Ex1_07_2 {
    public static void main (String[] args) {

        byte  calc1_1 = 1 ;                          //(8行目)変更しないでください
        short calc1_2 = 2 ;                          //(9行目)変更しないでください
        long  calc1_3 = 300000000000 ;               //(10行目)処理が正常に通るようにデバッグしてください
        _____ answer1 = calc1_1 + calc1_2 + calc1_3 ;//(11行目)アンダーバーに適切な処理を埋めてください
        System.out.println( answer1 );               //(12行目)「300000000003」が表示されているか確認してください

        byte   calc2_1 = 4 ;                         //(15行目)変更しないでください
        double calc2_2 = 5 ;                         //(16行目)変更しないでください
        int    calc2_3 = 6 ;                         //(17行目)変更しないでください
        _____  answer2 = calc2_1 + calc2_2 + calc2_3 ;//(18行目)アンダーバーに適切な処理を埋めてください
        System.out.println( answer2 );               //(19行目)「15.0」が表示されているか確認してください

        short  calc3_1 = 7 ;                         //(22行目)変更しないでください
        String calc3_2 = "8" ;                       //(23行目)変更しないでください
        int    calc3_3 = 9 ;                         //(24行目)変更しないでください
        int    answer3 = calc3_1 + calc3_2 + calc3_3 ;//(25行目)処理が正常に通るようにデバッグしてください
        System.out.println( answer3 );               //(26行目)「24」が表示されているか確認してください

        int    calc4_1 = 10 ;                        //(22行目)変更しないでください
        long   calc4_2 = 11 ;                        //(23行目)変更しないでください
        int    answer4 = calc4_1 + calc4_2 ;         //(25行目)処理が正常に通るようにデバッグしてください
        System.out.println( answer4 );               //(26行目)「21」が表示されているか確認してください

    }
}
```