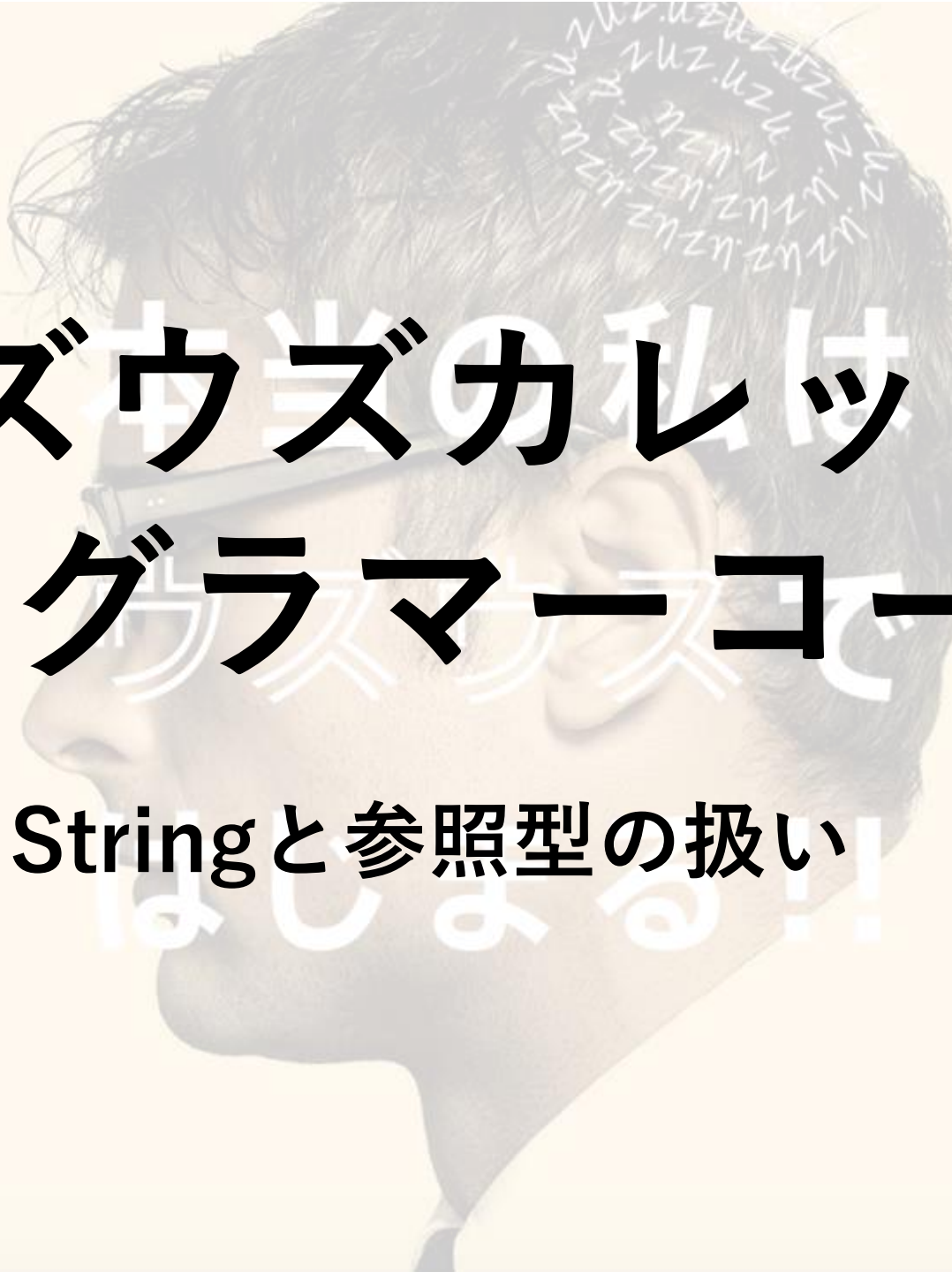


ウズウズカレッジ
プログラマーコース

Stringと参照型の扱い

▼Javaで扱われているプリミティブ型

種類	データ型	サイズ	補足
整数	byte	1バイト	1バイト（8bit）で表現できる符号付整数（-128～127）
	short	2バイト	2バイト（16bit）で表現できる符号付整数（-32768～32767）
	int	4バイト	4バイト（32bit）で表現できる符号付整数（-2147483648～2147483647）
	long	8バイト	8バイト（64bit）で表現できる符号付整数（約-922京～約922京）
小数点数	float	4バイト	4バイト（32bit）で表現できる浮動小数点数
	double	8バイト	8バイト（64bit）で表現できる浮動小数点数
文字	char	2バイト	半角・全角を問わず任意の一文字（Unicode文字）
論理値	boolean	1バイト	trueかfalse

Stringはプリミティブ型ではない！

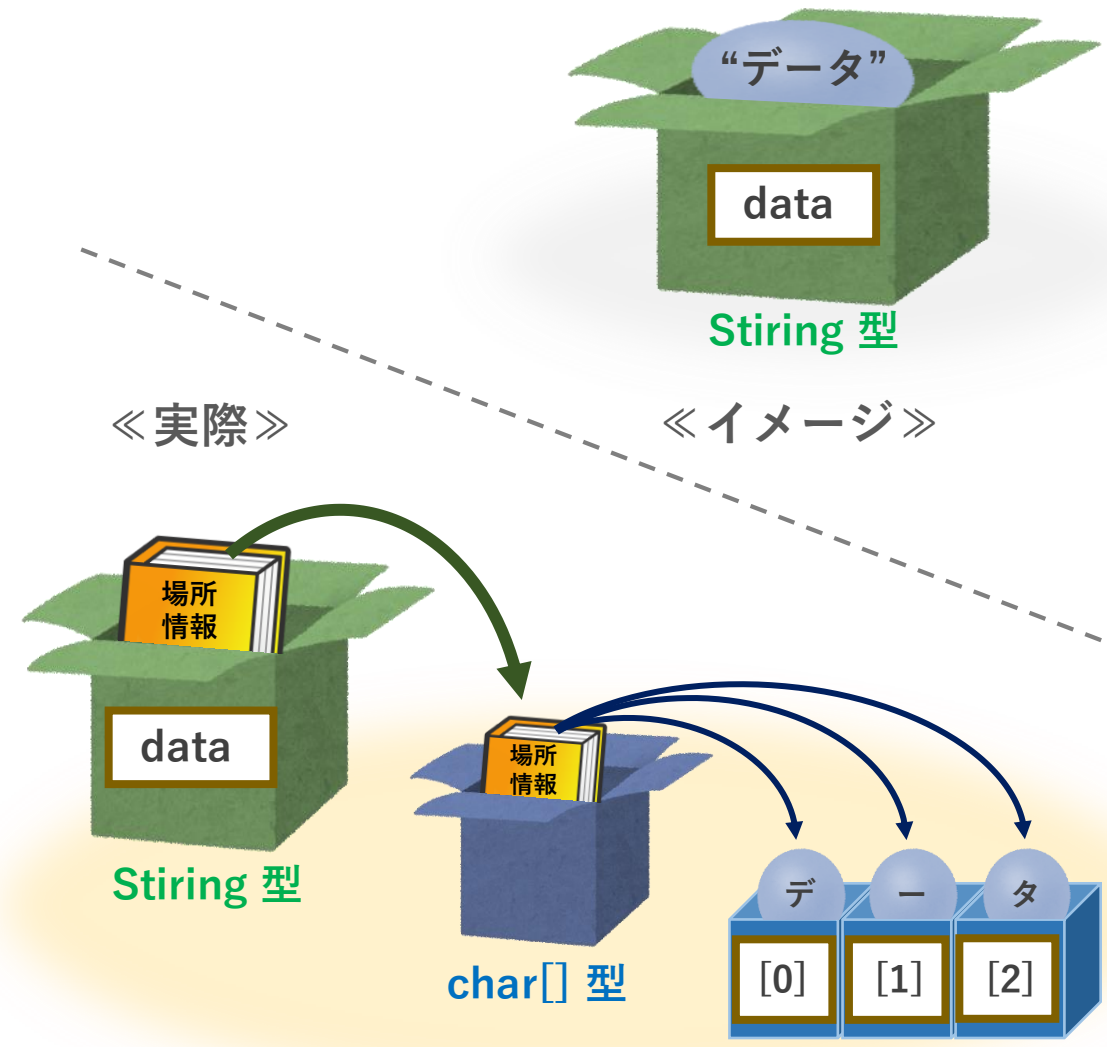
≪String型の正体≫

□プリミティブ型とは値が直接入っている箱のようなものですが、参照型との大きな違いとして**箱の大きさ（変数を作るために必要なメモリ領域）が決まっている**ということが挙げられます。
具体的には左表のとおりです。

□String型は文字列を格納しておける変数としてこれまでプリミティブかのように扱ってきましたが、実はプリミティブ型ではなく**参照型にカテゴライズされます。**

文字データをメモリに保存するためには1文字につき2バイト必要であり、例えば“プログラミング”という7文字の文字列であれば2×7=14バイト、”moco”であれば2×4=8バイトのメモリ領域が必要になります。
つまりString型は**どれだけメモリ領域を取ることになるのか読めない変数**であり、このような変数はすべて参照型となります。

String data =“データ” ;



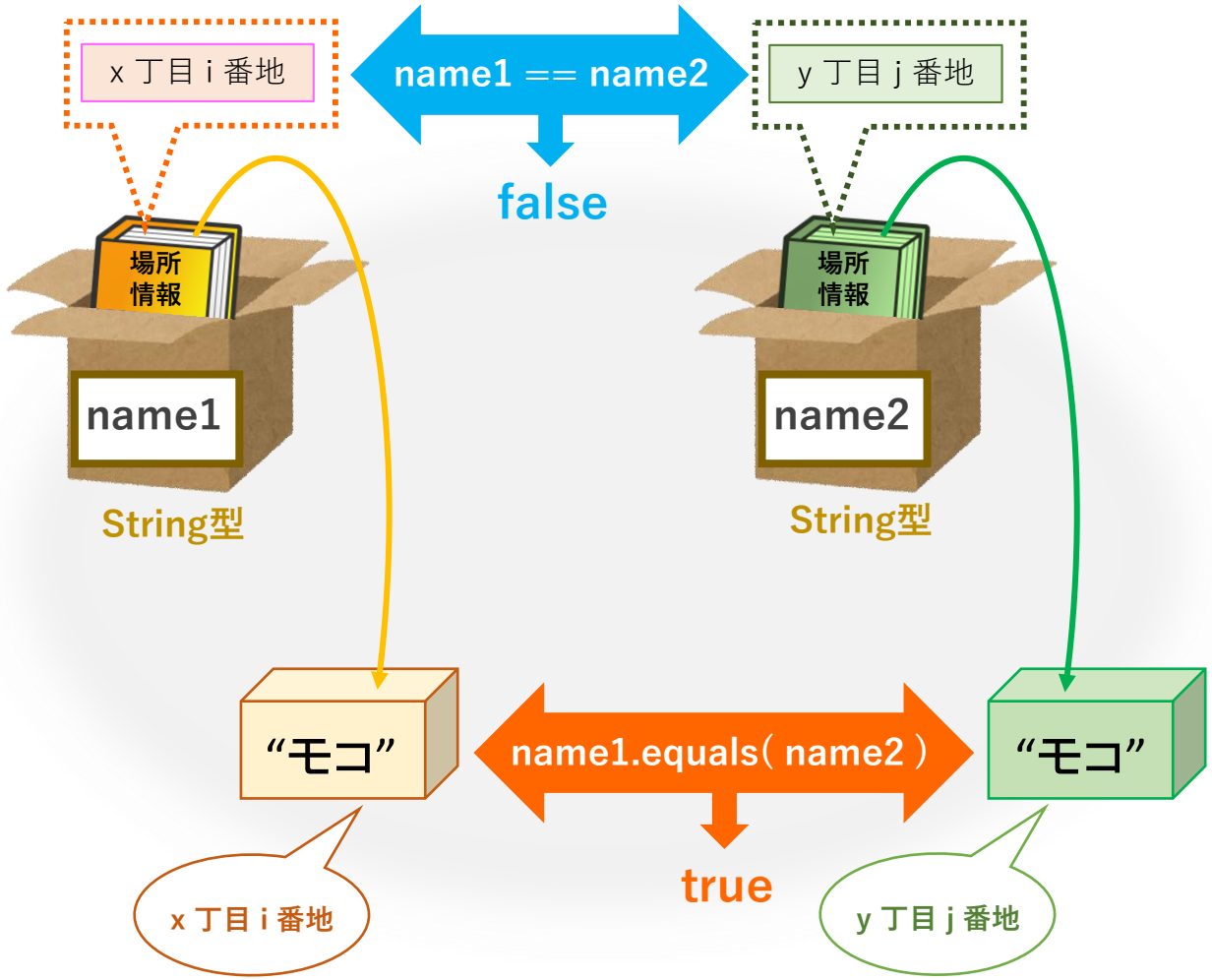
≪疑似プリミティブ型≫

□String型変数はプリミティブ型のように『型名 変数名』で生成することができますが、実はこれはString型が特殊なだけで、本来参照型の変数はこの方法では生成することができません。
実際、String型の正式な生成方法は以下となります。（String型変数dataを生成して「データ」という文字列で初期化する場合）

String data= new String(“データ”);

□文字列を扱うString型はプリミティブかのように使用されるシーンがあまりに多いため、プリミティブ型と同じような感覚で扱えるようJavaの開発者によって特別なプログラムが組まれています。
このような特性から、Stringは**疑似プリミティブ型**と呼ばれます。

□String型は文字列を内部的にchar型配列で管理しています。
String型の変数が参照された際は配列で管理している文字をすべて取得・結合した値を参照元に渡すという処理を内部的に行うことで、まるでプリミティブかのようなふるまいを実現しています。

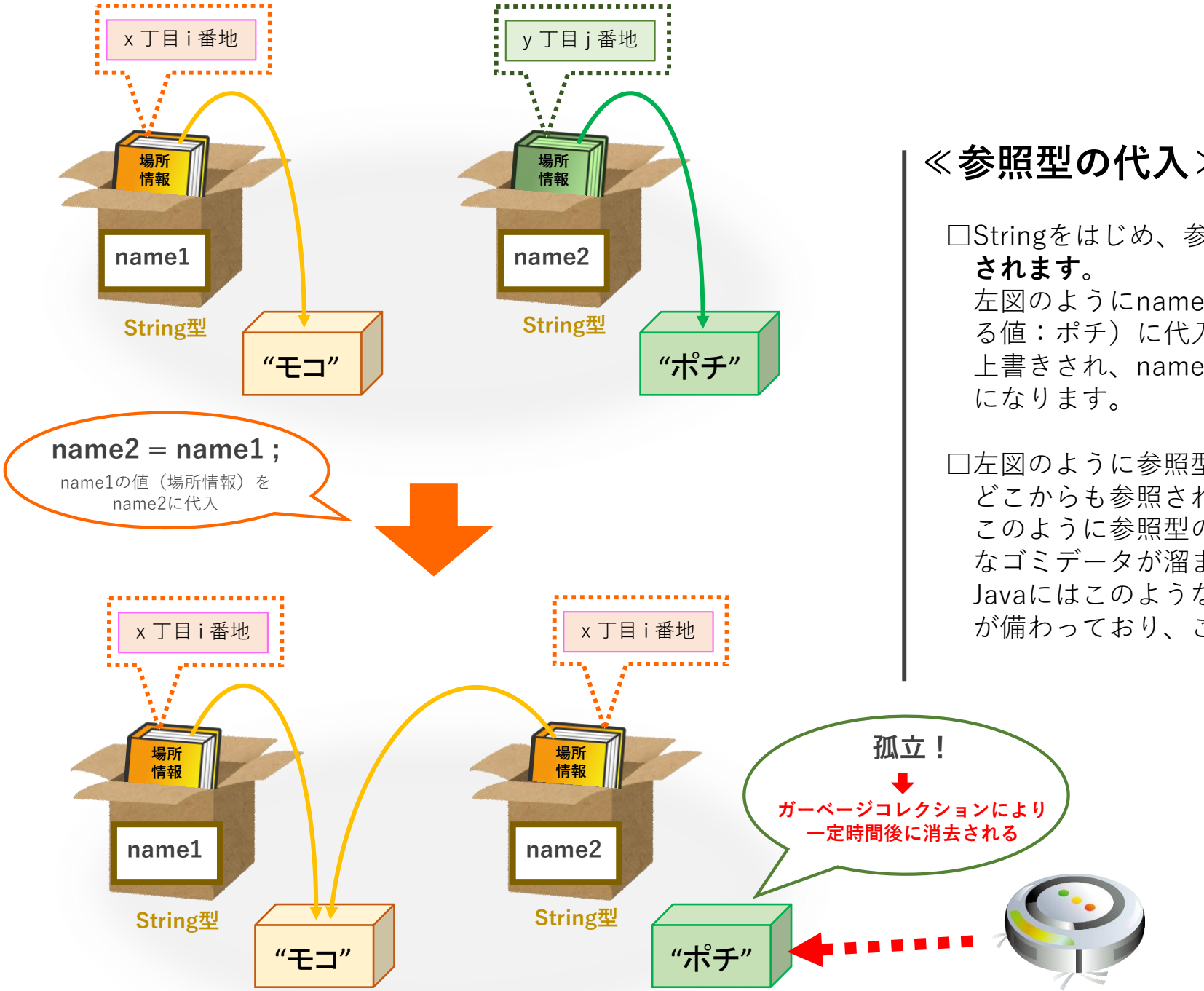


≪Stringの比較≫

□参照型の変数に格納されているデータは具体的な値ではなく他の変数の場所情報です。
この場所情報はコンピュータが内部的に扱っている人間には理解できない情報であるため、参照型の変数を参照しようとすると多くの場合場所情報のデータとして意味不明な文字列が取得されます。

□参照型であるString型同士を『==』で比較しようとすると場所情報の比較となってしまうため、それぞれが保持している文字列データが等しいかどうかという比較はできません。
文字列データの比較をしたい場合、以下のように『equals』を用いる必要があります。
（例ではString型変数 name1 と name2 の値を比較しています）

name1.equals(name2)



≪参照型の代入≫

□Stringをはじめ、参照型を代入する際は**場所情報がそのまま代入されます**。
左図のようにname1（参照している値：モコ）をname2（参照している値：ポチ）に代入すると、name2の場所情報がname1の場所情報で上書きされ、name1もname2も同じ場所（モコ）を参照している状態になります。

□左図のように参照型の代入をすると代入先で元々参照していた変数はどこからも参照されない孤立状態になります。
このように参照型の代入を繰り返していくとメモリ上にどんどん不要なゴミデータが溜まってしまいます。
Javaにはこのようなゴミデータをメモリ上から定期的に消去する機能が備わっており、この機能のことを**ガベージコレクション**と言います。