3　データを保存する～変数～

3.1　変数とは

さて、いよいよ本章から本格的に Java について勉強していくことになります。その最初の題材として扱われるのが、変数です。最初に来ていることからわかるとおり、変数なしでは Java のプログラミングは成り立ちません。

変数とは何か？一言で言えば、「記憶しておくもの」です。数値計算の結果を代入しておいて，あとでその値を別の計算に使うときなどに利用します.

例えば，パケット代が 1 パケット 1 円(高いな！)で，電話代が 1 分で 30 円の携帯電話を契約したとしましょう．基本料は 2680 円です.

この携帯電話を使って，一ヶ月でインターネットを 1000 パケット，電話を 1 時間半したとします.

このとき，この月の料金は果たしていくらでしょうか？

その計算は次のように行います.

```
System.out.println(“利用料金は”);
System.out.println(1*1000+30*90+2680);
```

これで，一ヶ月の料金が 4680 円だと分かりました.

携帯料金を調べたいだけならこれで終了です．しかし，月に 6380 円はちょっと高いからもう少し形態の利用を控えようかな～と思ったとしましょう.

このとき，パケット代と電話料金どちらを節約するべきか調べるために，パケット代と電話料金を別に表示してみたいと思います.

```
public class Sample {


	public static void main(String[] args) {
		System.out.println(″パケット代は″);
		System.out.println(1*1000);
		System.out.println(″電話料金は″);
		System.out.println(30*90);


		System.out.println(″利用料金は″);
		System.out.println(1*1000+30*90+2680);


	}
```

```
}
```

これで，パケット代 1000 円，電話料金 2700 円ということまで分かりました．しかし，このプログラムを見ると，

```
1*1000
30*90
```

が二回も出てきていてちょっと無駄ですね．

もし次の月に電話の利用時間を 60 分に減らしたとすると，

```
System.out.println("パケット代は");
System.out.println(1*1000);
System.out.println("電話料金は");
System.out.println(30*60);


System.out.println("利用料金は");
System.out.println(1*1000+30*60+2680);
```

二カ所も変化させなければ行けません．どっちかを 60 に直し忘れるとちょっと面倒なことになりそうです．

こんなときこそ変数の出番です．

```
int packetPrice = 1*1000;
int callPrice = 30*90;
int basicPrice = 980;


System.out.println("パケット代は");
System.out.println(packetPrice);
System.out.println("電話料金は");
System.out.println(callPrice);
int price = packetPrice+callPrice+basicPrice;
System.out.println(price);
```

これによって，同じ計算を二度も三度もやらずに済むようになります．

また，callPrice の値を入れ間違える可能性が半分に減りましたよね．二行目の callPrice 計算の所で 90 を 60 にするだけで良いのですから，直し忘れる心配はありません．

このように，同じ計算を何度もするときや，ある値をずっと取っておきたい場合などに変数が使われることになります．

プログラミングの基本である変数ですが、一般的には「値を入れておく入れ物」という

説明をされることが多いようです。しかし、「入れ物」だと後々理解が難しくなる可能性があります。というか、初心者の方がそこで詰まることがよくあります。私も詰まりました。そこで、入れ物よりももう少し将来的に理解が深まるように、「変数とは値を入れたロッカーの場所をメモした物」と理解してください。

3.2　変数の型

さて、Java の場合変数に格納できる値はあらかじめ決められています。つまり、整数を入れる変数には整数しか入れられませんし、実数を入れる変数には実数しか入れられません。この辺は c などのプログラム言語を使ってことのある方には当然のように思われるかもしれません。

皆さんも何かをしまって，その場所をメモするときは大抵何を入れたか忘れないようにしますよね？このロッカーには制服を入れて、こっちにはカバンを入れて・・・などなど。制服を入れるべきロッカーにカバンを入れてしまうと、後で制服を着ようと思ったときに、間違えてカバンを着てしまうというとんでもない事態になってしまう可能性があります。カバンを着て制服を持っている・・・シュールな光景です。

そんな大ボケをかますことがないように、Java では、あらかじめロッカーに「何を入れるロッカーか」ということを指定します。

Java の場合、変数には基本的に以下のものを入れることができます。

- 基本型(プリミティブ型)
- 配列
- クラス(オブジェクト型)

この中で配列とクラスについては後ほど詳しく説明するとして、ここでは基本型についてのみ説明したいと思います。

変数を作るときは、基本型のうちどの値を格納する変数なのかを指定しなければいけません。 なお、Java で用意されている基本型と、そこに格納できる値は以下のとおりです。

これを見ると、char 型と boolean 型以外はすべて数字しか扱えないように見えますよね。実はそのとおりなんです。基本的に、変数は数字しか扱うことができません。

型名	値
short	-32768 ～ 32767
int	-2147483648 ～ 2147483647
long	-9223372036854775808～ 9223372036854775807
float	±3.40382347E+38
double	±1.79769313486231570E+308

boolean	true/false
char	文字
byte	-128～127

3.2.1 整数型 short,int,long

これから説明する三つの変数型は、すべて整数を表します。なんだ、同じ整数を表すんだったら一つでいいじゃないか、と思われるかもしれません。いや、ま、まったくそのとおりなんですが、三つもあるということはやっぱり違いはあるということです。

実はこの三つ、それぞれ表すことができる範囲が異なるんです。

型名	範囲
short	-32768 ～ 32767
int	-2147483648 ～ 2147483647
long	-9223372036854775808～ 9223372036854775807

これを超える範囲の値は入れることができないので注意が必要です。

え？だったら最初からずっと long を使えばいいって？まあ､そうなんですけどね｡実は、変数を入れるロッカーには限りがあります。普通に使っている分にはあんまり関係ないんですけどね。で、当然大きい値を入れるには大きいロッカーが必要なんです。つまり、long 型のロッカーを用意すると、そのロッカーはかなり大きくなってしまいます。でも、実際にそのロッカーに入れるものが小さいとちょっと損した気分になりませんか？言うなれば、バッグ一つ入れるのにスキー用のロッカーを借りてしまうようなものです。それはもったいないので、小さい値しか入れないときは小さいロッカー、int や short を使うほうがいいわけです。

とくに、整数の場合は int が一番良く使われます。普通は int を使って、ものすごい大きい値を入れなきゃいけない時だけ long を使うようにすると効率的でしょう｡え？short？すいません、筆者もあんまり使ったことありません・・・

最後に、それぞれの変数型の宣言方法を書いて終わりにしましょう。

```
short s = 128;                  //short 型の場合
int i = 65535;                  //int 型の場合
long l = 1414213561549; //long 型の場合
```

3.2.2 実数型 float,double

int.long 等は整数しか使えません。しかし、プログラミングをしていると、どうしても小数点以下の数字を使いたくなるときもあります。単純な話、0.5 だって整数型では表現できないんですよ。これは困る。というわけで、用意されているのが、実数型です。

整数型にはその数値範囲によって 3 つの型が用意されていますが、実数型には、二つが用意されています。それが、float と double です。

整数型が範囲によって使い分けするように、実数型も使い分けをします。ただし、実数型の場合は範囲によって、というよりは精度によって使い分けをすることになります。まあ、本当は色々小難しい話がありますが、とりあえず、double の方が float よりも精度良く表現できると思ってください。

整数型で short があまり使われないように、実数型でも float はあまり使われません。基本的に実数型は double を使うと思いましょう。

型名	範囲
float	±3.40382347E+38
double	±1.79769313486231570E+308

では、最後に float.double 型の変数を作る方法をお送りします。

```
double d = 3.1415926535;          //double 型の場合
float f = 0.5;                    //float 型の場合
```

3.2.3 論理型 boolean

整数、実数ときて、次にお送りするのは論理型です。ちょっと聞きなれない言葉ですが、実はこれ、Yes/No の二値をもつ変数型なんです。「なんだそれ？」とお思いの方は多いかと思いますが、プログラミングをしていると、Yes/No 判定は異様に多く出てきます。後の章に出てくる「条件分岐」や「繰り返し処理」で Yes/No 判定は必ず使われることになります。いや、むしろ Yes/No 判定しか使われないくらい。

特によく使うのが,

「もしこうだったらこうする」

という判定の時です．プログラムを書いていると，こういう条件分岐というのが嫌と言うほど出てきます．現実だってそうですよね．「もしお金があったらこの前出た新機種を買うのに・・・」「もしうちに電波が入っていたらトクダフォン株式会社の携帯電話に MNP したのに・・・」みたいな．

そんな大活躍予定の論理型は boolean という型になります。で、boolen 型にいれられる値は、数値じゃありません。JAVA には boolean 型専用とも言うべき値が存在します。それが、**true** と **false** です。この二つの値は、

値	意味
true	正(あってるよ)
false	偽(間違ってるよ)

をそれぞれ表現しています。

わざわざ専用の値を用意している時点で、boolean 型の優遇振りが分かるというものではありませんか。こんな恵まれている boolean 型、使わないわけには行きませんね。

型名	値
boolean	true/false

ちょっと小難しいお話をすると、c/c++を使ったことのある方は、bool 型という変数型を覚えているかもしれません。JAVA の boolean 型は、c/c++の bool 型とは根本的に違います。c/c++の bool 型の中身は実際には int 型で、0 か 0 でないかで判定をしていました。そのため、int 型で代用することも出来ました。それに対して、JAVA の boolean 型は true/false しか値を許しません。int 型で代用することも出来ません。

ま、とりあえず、boolean 型は正しいかどうかを格納するロッカーであると覚えてきましょう。

では、最後に boolean 型変数の作り方です。

さて、さっきから数字やら Yes/No の変数型ばっかりで飽きてきたかと思います。そこで、

```
boolean t = true;          //boolean 型(初期値が true の場合)
boolean f = false;   //boolean 型(初期値が false の場合)
```

3.2.4 文字型 char

ここでちょっと趣向を変えて文字を入れる変数型をご紹介しましょう。それが、char 型です。

といっても、実は char 型も元は数字なんですけどね。あ、いや、怒らないで！コンピュータは、その性質上どうしても数字を扱うほうが得意なんですよ。そこで、仕方がなく文字も数字として表すんです。たとえば、65 は大文字の A を表しますし、43 は+記号を表します。こんな感じで数字と文字を組み合わせていくんですが、実際数字と文字を人間が覚えるのは大変なので、char 型には文字を直接入れることが出来ます。

以下の例を見てください。

```
char a = 'a';
char b = 'B';
```

このように、’と’(シングルクォーテーションと呼びます。)の間に一文字書いたものを=で結べば、変数の中に文字を入れることが出来ます。おお、これなら一々「えっと、C は何番目だったかな・・・ああ、そうそう 67 か」なんて調べる手間が省けますね。

とまあ、char 型の説明を書いたんですが，大抵の場合、文字を書くときは文章を書くときですよね。残念なことに char 型は文字一つしか表現できません。ということは、一文書くのに、大量の char 型変数が必要ということになります。いや、やろうと思えばできないことはないんですが、管理が大変そうですよね。思い出してください。変数というのは、ロッカーのようなものなんです。そのロッカー一つに一つの文字しか入っていなかったら、文章中の全ての文字が入ったロッカーを管理しなきゃいけないわけですよ。何千文字とい

う文章を書きたければ、何千ものロッカーを覚えておかなきゃいけないし、それぞれのロッカーに名前をつけなきゃいけない。こんな大変なことはありません。

というわけで、Java には char 型よりも便利な String 型という変数型が用意されています.

3.2.5 文字列 String

この **String** 型は、char 型と違い複数の文字を同時に入れることができるロッカーを使うことができる変数です。

イメージが湧きやすいように、先に例を示してしまいましょう。

```
String terminator = “I’ll be back”
```

String 型の場合、”と”(ダブルクォーテーションと呼びます)に囲まれた文字列を=で結ぶことで値を変数に入れることが出来ます。

というわけで、あの有名なシュワちゃんのセリフだって簡単に再現できちゃいました。これで「pneumonoultramicroscopicsilicovolcanoconiosis（脚注：世界一長い単語)」と書きたいときでも安心です。文字を扱いたい場合は、基本的に String 型を使うと覚えておきましょう。

そうそう、char 型では日本語が使えませんでしたが、String 型では日本語だってラクラク使えます。

```
String kakusan = “この紋所が目に入らぬか！”
```

これで悪代官たちも平身低頭です。黄門様は高笑いです。

変数型	値
char	文字一文字
String	文字列

さて、こんな便利な String 型があるのに、どうして char 型なんてものがあるんでしょうか？実は、String 型には重大な秘密があります。今までに紹介した int 型や char 型は**プリミティブ型**と呼ばれる型で、JAVA にもともと備わっている変数型で、これらの型は一つの型に付きロッカー一つを使うという決まりがあります。一方で、String 型は実は JAVA にもともと備わっている型ではなく、**参照型**と呼ばれる特殊な型なんです。仲間はずれなんですね。

まあ、でもここではあんまり難しいことは考えずに int 型や double 型と同様に仲良くしてあげてください。プリミティブ型と参照型の違いについては後ほどじっくりと説明しますから。

3.2.6 バイト型 byte

さて、長かった変数形説明の最後は byte 型です。なぜ byte 型が最後に来たかといえば, byte 型を使うようになるのは､JAVA について相当詳しくなってきてからになると予想されるからです。あるいは、JAVA だけじゃなく、もっとコンピュータについて詳しくならないと、イマイチ byte 型は使い道がありません。というわけで、「byte 型？ああ、そんなものもあったなあ」と後で思い出せる程度に頭の片隅にそっと追いやっておいてください。

と、これで説明を終わらせるのもなんなので、一応説明しておきます。

「byte 型とは、0～7 までの数値を格納できる整数型と一緒」です。そんなものがなんの役に立つんでしょうか？実は、コンピュータ内のデータはすべてバイトと呼ばれる単位で管理されています。このバイトというのが、0～7 までの数値なんです。これまでに説明した int 型も double 型も、元を正せばすべてこのバイトによって表現されているものを、人間がわかりやすいようにいくつかを一まとめにしているだけなんですね。これはびっくり。と同時に、byte 型の重要さは分かっていただけたかと思います。

コンピュータ内部と深く関係するようなプログラミングをするようなとき、byte 型は重要になることがありますので、そのときになったら思い出してあげてください。

3.2.7 変数まとめ

以上で、変数型の説明は終わりです。String を除いたすべての変数型がプリミティブ型と呼ばれる JAVA が本来持つ変数型です。逆にいうと、これ以外にはプリミティブ型の変数は存在しません。

たった 8 個しかないので、是非全部覚えて・・・おきたいところですが、別に全部覚える必要はありません。特に、皆さんのように JAVA を勉強したての方にとっては、今後も新しい情報が次から次へと押し寄せてくるのですから､8 個も覚えるのは面倒くさいでしょう。というわけで、良く使う変数型を厳選して載せておきましょう。

実は、基本的に Java では以下にあげる 5 つの型だけ覚えておけば十分です。Java の基本ライブラリである JavaSDK で以下の 5 つ以外はほとんど使っていない、というくらい、他の型はあんまり使われることがありません。というわけで、以下の 5 つは今後本書でも頻出しますので、ぜひ覚えて置いてください。

変数型	意味	値
int	整数	-2147483648 ～ 2147483647
long	大きい整数	-9223372036854775808～ 9223372036854775807
double	実数	$\pm 1.79769313486231570 \times 10^{308}$
boolean	論理値	true/false
String	文字列	文字列

3.3　計算式

Java では変数を使って色々計算することができます。すでに使っている=も代入演算子という式の一種になります。

では、Java で行える計算にどのようなものがあるのか、見ていきましょう。

3.3.1　基本演算

まずは、計算の基本四則演算です。四則演算とはご存知、計算式のスター、足し算引き算掛け算割り算のことです。その計算方法を具体的に見てみましょう。

足し算，引き算，掛け算，割り算，そして割り算の余りの計算を行います.

```
int x = 10+20;
int y = 100-80;
int z = y*50;
int a = z/50;
int b = a%7;

System.out.println("10+20="+x);
System.out.println("100-80="+y);
System.out.println("y*50="+z);
System.out.println("z/50="+a);
System.out.println("a%7="+b);
```

これを実行した結果は以下のとおりです。

```
10+20=30
100-80=20
y*50=1000
z/50=20
a%7=6
```

これを見ればわかるとおり、Java の四則演算は、+^*/を使います。また、余りの計算には%を使います。

意味はそれぞれ以下のとおり。

種類	演算子	内容	例	結果
算術演算子	+	足し算	1+1	2
	-	引き算	10-5	5
	*	掛け算	5*5	25
	/	割り算	15/5	3

	%	余りの計算	10%3	1

通常と違うのは、×と÷がそれぞれ＊と/になっているだけですね。それに、余りの計算%が追加されていることです。

それ以外は普通の計算とまったくいっしょで、違いはコンピュータが勝手に計算をしてくれる点だけです。

では、応用編として以下のような式があったとき、どう計算するか考えてみましょう。

```
int answer1 = ((-50*5)+14-6)/2+15;
double answer2 = 1.5*(-2)+8.123/4.2;
```

なんとなく見ればどう計算されるのかわかったのではないでしょうか？この計算結果と皆さんが計算した結果が正しいかどうかチェックしてみてください。

ちなみに、答えは**-106 -1.0659523809523812となるはずです。**

3.3.2　代入演算子

代入演算子とは、ある変数に値を入れるための変数です。実はもうこれまでにも何度も利用している=がもっとも代表的な代表演算子なのです。もうすでにみなさん分かっているとおり、=というのは等しい当為意味ではなく、「この変数の値はこれですよ」ということを示しています。

つまり、

```
x = 10;
```

は x と 10 が等しいのではなく、x に 10 という値を指しますよ(代入する)という意味なのです。

もちろん、

```
y = x+10
```

と書けば、今度は x に 10 を足した値を y に代入するという意味になります。さきほど x に 10 を代入しましたから、この場合 y は 20 を指すわけですね。

これだけですので、使い方に迷うことはないでしょう。

ところで，代入というとまるで x が 20 そのものになってしまった様な気がしてしまいますが，それは違います．あくまでも，x は 20 という数字を指しているだけなのです．

```
y=x+10
```

も，「x に 10 を足したものが y」のではなくて，x が指している値（それが 10）に 10 を加えたもの(20)を y は指しているよ，20 が格納されているロッカーの場所をメモしてあるんだよ，ということを忘れないでください．

代入演算子には=以外にも以下のような演算子があります。それは、複合的な代入演算子というもので、四則演算などの計算をした結果を代入するという演算子です。

複合的な演算子の例を見てみましょうか。

```
y = 10;
y += 5;
System.out.println("y="+y);
```

出力結果

```
y=15
```

というわけで、

```
y = y+5;
```

と書くところを

```
y += 5;
```

と書くことができる、これが複合的な代入演算子です。まあ、別になくても何とかなる演算子なのですが、これがあればちょっとプログラムを書く手間が省ける、という利点があります。

というわけで、Java で用意されている基本的な代入演算子を下に上げておきましょう。ちなみに、例で使っている a には全部 6 が入っていると思ってください。

種類	演算子	内容	例	結果
代入演算子	=	代入	a=0	a は 0
	+=	足して代入	a+=2	a は 8
	-=	引いて代入	a-=2	a は 4
	*=	掛けて代入	a*=2	a は 12
	/=	割って代入	a/=3	a は 2
	%=	余りの代入	a%=5	a は 1

なお、本当はもうちょっと代入演算子はあるのですが、正直使う機会はあまりないので本書では紹介しないことにします。

3.3.3 特殊な演算子

Java で使うことができる演算子はまだまだあります。四則演算ができればもういいんじゃないかという気がしますが、これから先はもっと便利に Java を使うために有効な演算子を紹介してきます。

まず、最初に紹介するのが**インクリメント演算子**と**デクリメント演算子**です。これらの演算子の役目は「1 を足すこと」「1 をひくこと」です。なんだこれ、と思うかもしれませ

んが、見てみましょう。

```
x = 10;
y = 20;
x++;
y--;
System.out.println("x="+x);
System.out.println("y="+y);
```

出力結果

```
x=11
y=19
```

というわけで、1 増えたり 1 減ったりしたことがわかったと思います。これで何がうれしいのかといわれると、**書くのが簡単**これ以外にはありません。

ちなみに、++演算子、--演算しともに変数のあとに書く場合と前に書く場合があります。ちょっとそれを見てみましょうか。

```
x = 5;
y = 5;
System.out.println(x++);
System.out.println(++y);
```

出力結果

```
5
6
```

あれ、x++の方の値が変化していません。パソコンが壊れているのではないかと疑ってしまいそうになりますが、これで正しいのです。

++演算子は、変数の先に書くと先に足した結果を返しますが、変数の後ろに書くと計算順位が下がって、他の計算が終わってから計算されることになるのです。

この例の場合だと、x++の方は先に x の値を画面に出力してから 1 足しているので、足される前の 5 が出力されます。一方、++y の方は先に 1 を足してから y の値を画面に出力しています。そのため、5 から 1 足された 6 が出力されているのです。

最初のうちはこの違いはなかなかわかりづらいので、変数のあとにつけるように統一しておいた方がわかりやすいでしょう。そして、他の計算が終わってから 1 増えるんだな、ということを忘れてはいければ大丈夫です。

というわけで，1 増やす書き方には以下の 3 種類がある事を覚えておきましょう．

```
x = x +1;
x += 1;
x++;
```

1 増やす，1 減らすという作業は何度も繰り返し行われることが多いので，できるだけプ

ログラムを書くときに手間をかけずに済むように，このような書き方が許されています．まあ，最初の内は分かりづらければ

```
x = x +1;
```

と書けばよいと思いますよ．

さて、インクリメント演算子とデクリメント演算子を紹介しましたが、それ以外にも**シフト演算子**というものがあります。・・・が、シフト演算子は初心者のうちはあまり使うことはないでしょう。そこで、あくまでも入門書な本書は余計な情報で混乱させないためにもシフト演算子には**そんなものもあるんだね**くらいの遠くから生暖かく見つめる気分に浸っていてください。

というわけで、最後に特殊な演算子について表にまとめて終わりにしたいと思います。

種類	演算子	内容	例	結果
インクリメント演算子	++	1 増やす(後置)	a++	a は 7
	++	1 増やす(前置)	++a	a は 7
デクリメント演算子	--	1 減らす(後置)	a--	a は 5
	--	1 減らす(前置)	--a	a は 5

3.4　条件式

条件式は、次章で紹介する if 文、for 文、wfile 文、do-while 文といったプログラム制御で利用する非常に重要な式です。

条件式とは、答えが true/false になる式です。通常の足し算などは答えが 1 とか 2 とかいう数字で表現されますが、条件式の場合は「あってるよ」「間違ってるよ」としか答えが返ってきません。

で、どういう場合に使うのかといえば、たとえば

「**通話料が 2000 円超えていたら**もう電話しない」

「**相手のメールアドレスが登録されていたら**、登録名を画面に表示する」

という、「もしも～なら」という判断をしたいときに利用します。

具体的な使い方は次章を読めばいろいろな例が出ていますので、簡単にわかると思います．まずは，ここではその条件式の書き方を学んでいきましょう。

3.4.1　等号

まず最初に覚えるべき条件式は、同一性の判断です。ある二つの変数（あるいは値）を

比べて、同じ物かどうか判断する方法です。まあ、すでに何回か出てきているので改めて説明するまでもないかもしれませんが・・・

同じ物かどうか判定する場合には、==という記号を使います。使い方の例をちょっと見てみましょう。

```
int a = 2;
System.out.println(a==2);
```

出力結果

```
true
```

この例だと、a が 2 なので、結果は正しいことを示す「true」となります。

なお、このように、条件式の結果は必ず true/false で返ってきますので、その結果はすべて boolean 型に代入することができます。

```
boolean result;
result = (a==2);
```

これによって、result の中身は true になるわけです。

コラム

ところで、なんで同一かどうかの判断に==を使うのでしょうか？=の方が意味的にはあっていますよね？

5 = 5

なんて書きたい！

でも、思い出してください。本章で代入演算子を勉強しましたよね。そのときに、=は代入するための演算子だと定義されていたことを思い出してください。そのために=を同一性の比較に使うわけにはいかなくなってしまったのです。

なんとも不条理な話ですが、代入を=、同一性の比較を==というのは、Java が参考にした c というプログラム言語の時代から使われているルールなのです。Java ではルールを変更しても良かったのですが、もし変えてしまうと c 言語をやってきた人たちが Java を使うときに混乱してしまうことになります。

というわけで、混乱を避けるために c 言語にあわせて=は代入演算子として使われることになりました。

ただし、c 言語より後から出てきた Java は c 言語よりはもうちょっと賢い工夫をしています。

実は c 言語では=と書いた場合と==と書いた場合でどちらでもコンパイルがとおってしまいました。つまり、同一性比較をしようとして=と書いても、コンパイルはできたのです。しかしながら、Java では同一性を比較しようとしているときに=と書いてあったら、「書き

間違えているよ！」と教えてくれる機能が追加されています。

昔の流れを継承しているとはいえ、よりわかりやすくしている Java の方が c 言語よりは少しだけ開発者にやさしいのかもしれません。

コラムここまで

3.4.2　不等号

次にご紹介するのは、比較です。比較というのは、数字の大きさを比較するものです。数字の大きさを比較する記号は全部で 4 種類あります。

```
a > 5
a >= 5
a < 5
a <= 5
```

どれも見れば意味がわかるようなものばかりですし、中にはすでに一回見たものもありますね。

a>5 は a が 5 より大きいかどうかを判定します。a が 6 や 100 や 5.4 などの場合、true になり、a が 4 や-10、あるいは 4.999 の場合も false になります。そして、注意するべき点として、a が 5 の場合も false になります。注意してください。

a>=5 の場合は、逆に a=5 の場合は true になります。まあ、見れば分かると思いますが。その他は、a>5 と一緒です。

a<5 は、a>5 と逆で、a が 5 より小さい場合にのみ true となります。a が 4 でもいいですし、4.5 でも-100 でも true です。逆に a が 5、6、150 やあるいは 5.1 の場合も false となります。

最後の **a<=5** はもう分かってしまったと思いますが、a が 5 以下の場合に true となります。5 より少しでも大きければ false です。

まあ、中学の数学で習うような式そのままですので、混乱することはないでしょう。

3.4.3　否定

さて、次は否定の表現です。否定の表現の仕方には二種類あります。

```
a != 10
!(a > 10)
```

JAVA において否定を表現する記号は!です。!がついていれば true/false が自動的に逆になります。

a !=10 は、a==10 の否定という意味になります。つまり、**a が 10 ではない場合のみ true** になります。

一方、**!(a>10)**はちょっと特殊な書き方ですが、**()内の条件式を反転**します。この場合ですと、a>10 の結果が ture ならば!(a>10)の結果は false になります。つまり、a が 12 であれば false なわけです。逆に a>10 が false ならば!(a>10)は true です。a が 5 ならば true

になるわけですね。これは，先に（）内を先に計算すると考えれば簡単に分かるでしょう．（）内が true ならば，その後否定しているので false になります．逆に（）内が false なら，その後否定するので true になるのです．

3.4.4　まとめ

最後に、条件式をまとめてみます。例もあわせて乗せておきますので、分からなくなったらここに戻ってきて確認してみて下さい。

条件式	演算子	例	例の意味
同一性判定	==	a == b	a と b が等しければ true
大小判定	>	a > b	a が b より大きければ true
大小判定	>=	a >= b	a が b 以上ならば true
大小判定	<	a < b	a が b より小さければ true
大小判定	<=	a <= b	a が b 以下ならば true
否定	!=	a != b	a と b が等しくなければ true
否定	!	!(a > b)	a が b より大きくなければ true

これら比較演算子を使ったプログラムの例を示しましょうか．

```
//現在までの通話料金
int callPrice = 550;
//無料通話分
int noChargeLimit = 2900;

System.out.println("通話料金:"+callPrice);
System.out.println("無料通話:"+noChargeLimit);

if(callPrice == noChargeLimit){
        System.out.println("無料通話分ちょうどです");
}
if(callPrice > noChargeLimit){
        System.out.println("無料通話分超過しています");
}
if(callPrice < noChargeLimit){
        System.out.println("無料通話分が余ってます");
}
```

色々比較して，比較結果によって表示が異なるようになっています．

```
通話料金:550
```

```
無料通話:2900
無料通話分が余ってます
```

今回は無料通話分がまだ余っていますが，通話料金 550 円を色々変えて確かめてみてください．2900 円を超えたら表示が変わるはずですよ．

以上で条件式の説明を終了です。次章で嫌というほど条件式は出てきますので、楽しみにしていてください。え？楽しみじゃない？せっかく勉強したんですから、それを使える場があることは楽しみにしておきましょうよ。

3.5　型変換

Java の変数は、基本的に決められた値しか入れることができません。つまり、整数型である int 型に double 型を代入することはできませんし、boolean 型と int 型を変換することはできません。

しかし、異なる型同士の代入を行いたい場合というのも存在します。そこで、そのような場合のルールを説明していきましょう。

3.5.1　明示的な型変換

明示的な型変換とは「型を変えるよ」ということを Java にプログラム上で教えてあげることを言います。

たとえば、以下のような例を考えてみましょう。

```
long longValue = 1000;
int x = longValue;
```

この場合、われわれとしては、x にも 100 が入ってくれることを期待します。しかしながら、このコードをコンパイルしようとすると、

```
型の不一致: long から int には変換できません。
```

というエラーメッセージが出てきてコンパイルする事ができません。

これは、long 型は 9223372036854775807 までの整数を格納できますが、int 型は、2147483647 までの値しか持つことができないためです。もし long 型の値を int 型の変数に入れようとしたときに、int が持てる値の範囲を超えていたら困ってしまいます。もちろんプログラムをしている我々は、そんな大きい値じゃないということを知っていて、無理やり int 型変数に入れることができると知っていても、Java にはわからないわけです。もしかしたら、書いた人が間違えて書いてしまっただけかもしれません。もしそうだとすると、無理やり long の値を int にしてしまったらプログラムに重大な問題を起こしてしまう可能性があります。

というわけで、こういう場合は、以下のように「この long 型の値を無理やり int 型にしてもいいよ！」ということを明示的に知らせてあげるわけです。

```
long longValue = 1000;
int x = (int) longValue;
```

これによって、プログラムは「ああ、ここは int にしていいんだな、プログラムを書いた人は、もし longValue が int に入りきらない値だったとしても、無理やり int にしてしまうことを理解しているんだな」とわかるわけです。

というわけで、明示的な型変換は以下のように行います。

(型名)変数名;

ちなみに、このような型変換のことを「キャスト」といい、(型名)と書く部分を「キャスト演算子」といい、演算子のひとつとして扱います。

コラム

ところで、int の範囲を超えた long 型の値を int に入れてみたらどうなるでしょうか？

この場合格納される値は必ず-1 になります。

一方、double 型などの実数を無理やり int 型に入れると、小数点以下が切り捨てられて 0 に近い値になります。ですので、5.52 を int 型に型変換して入れようとすると、5 になりますし，-10.5 を int 型に変換すると-10 になります。

小数点以下を切り捨てたい時などには便利だったりしますけどね.

コラム終了

3.5.2 暗黙の型変換

Java には暗黙の型変換というのがあります。なんか「暗黙の」とか言うとちょっとかっこいいですね。スティーブンセガールとか出てくる映画っぽいです。

さて、この暗黙の型変換とは、代入するときに特に意識しなくても型を自動的に変換してくれることを言います。

先ほど long→int の型変換は明示的に書かないとコンパイルがとおらないという話をしましたが、int→long の型変換は明示的に書かなくてもコンパイルがとおってしまいます。

```
int intValue = 1000;
long x = intValue;
```

なぜならば、long の方が int より大きい値を格納できるため、この代入は必ず成功するということがわかっているからです。必ずうまくいくことがわかっているのに、わざわざ書くこともないですよね。

というわけで、小さい値が格納できる型から大きい値が格納できる型への変換は何も書

かなくても自動的に行うことができるのです。

値の格納できる範囲が大きい型を「大きい型」、値を格納できる範囲が小さい型を「小さい型」と呼ぶことにします。

型の大きさは以下の表にまとめたとおりです。上にあるものほど大きいので、上にある型の変数へ下の変数から代入する場合は、暗黙の型変換が行われます。ちなみに、上から下へ変換する場合は、明示的な型変換が必要です。

型	範囲
double	±1.79769313486231570E+308
float	±3.40382347E+38
long	-9223372036854775808～ 9223372036854775807
int	-2147483648 ～ 2147483647
short	-32768 ～ 32767

3.5.3　計算結果の型変換

JAVA の計算式には、「計算結果は大きい型で返す」というルールがります。int 型と double 型で計算したら、double 型で答えが返ってくるわけです。

```
int x = 100;
double y = 2;
System.out.println(x*y);
```

出力結果

```
200.0
```

というわけで、計算結果自体は 100×2 で 200 という整数ですが、先ほどのルールが適用されて double 型で答えが返ってくることになりました。これは、計算結果の制度を落とさないために行われている処理です。もし、ここで y=2 はなく、y=0.001 だった場合、答えが 0.1 となり int 型では表現できなくなってしまいますからね。

ちなみに、もし答えを int 型に格納したい場合は、注意してください。型変換を行わないとコンパイルエラーが出てしまいますよ。

```
int x = 100;
double y = 2;
int z = (int)(x*y); //(int)を忘れずに・・・
```

3.6　変数名

変数には必ず名前を付けることになります。これまでの例だと， x とか y とか名前を付けていましたよね。しかし、実際に使うときは x や y だと何を意味しているのかよくわからなくありませんか？

実は、変数の名前は割と柔軟に付けることができます。つまり、X とか Y じゃなくていいわけです。

変数につける名前には以下のような決まりがあります。

- 英字、数字、アンダーバーなどを使う
- 長さに制限はない
- あらかじめ決められたいくつかの単語(予約語)は使うことができない(return, int, class など)
- 数字ではじめることは出ない
- 大文字と小文字は区別される

これらの決まりさえ守っていればどんな変数名をつけてもかまいません。たとえ変数名に好きな子の名前を付けたって Java はニヤニヤ笑ったりしませんので安心です。ほかの人に見られると恥ずかしいけど。

というわけで、以下のような名前が Java では許されています。

```
a
something
something_new
Windows2000
_thisIsAPen
```

一方、こんな変数名はだめです。

```
47todofuken //数字から始まっている
double //予約語である
(^_^) //使ってはいけない文字()^が使われている
```

ちなみに、同じ名前を別の変数に対して使うことはできません。つまり、同じ名前の変数が一度に二つ存在してはいけないわけです。それはそうですよね。コンピュータは変数を名前によって区別しているので、同じ名前が存在したら当然同じものだと思ってしまうわけですから。

したがって、ひとつの変数に名前を付けて、同じような名前を別の変数に付けたくなると困ったことになってしまいます。そのため、変数に名前を付けるときは慎重に、かつ変数の中身をよくあらわしているものにしましょう。

たとえば、通話料をあらわす変数名を作るときに、

```
int callPrice; //電話料金
```

と名づけるのはお勧めです。

もし、ここで、

```
int price; //電話料金？
```

と名づけてしまって、パケット代の価格を変数にしたくなった場合名前を付けるとき，整合性が取れなくなってしまいます.

```
int packetPrice = 2000; //間違いなくパケット代
int price = 1000; //電話料金？
```

そんなわけで、変数名をつけるときは注意して付けるようにしてください。

ちなみに､変数に名前を付けるときには暗黙の決まりがあります｡別に守らなくても Java プログラミングはできますが、守ったほうが楽にプログラミングできるよ、という決まりです。その決まりについては後々お話したいと思います。

3.7 間違えやすい変数

さて，変数について色々書いてきましたが，変数には初心者が間違えやすいポイントがいくつかあります．それらの間違えやすいポイントについていくつか揚げていって見たいと思います．本書を読んだ皆さんはこの間違えポイントに引っかからないように一流プログラマを目指してください.

3.7.1 0 割の恐怖

計算式を使う場合、いくつか**やってはいけないこと**があります。そのひとつが、0 での割り算です。

これは義務教育の時代にすでに教わっていることですが、割り算の計算で 0 で割ってはいけない、というルールがあります。覚えていますか？

このルールは数学のルールですので、当然 Java の計算でも当てはまります。

次のようなプログラムを書いてみましょう.

```
int result = 100/0;
System.out.println("100/0="+result);
```

これを実行すると・・・

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

というエラーメッセージが出てきて、プログラムが終了してしまいます。したがって、割り算をするときは、常に 0 で割っていないかどうか注意する必要があります。

100 を 0 で割った結果は出すことが出来ません.

もしこのルールを忘れると、とんでもないところでエラーが発生してプログラムが停止

してしまうかも知れませんよ。

「まさか,そんな間抜けなことはしないよ！」

と皆さん思うかも知れません．が，こんなプログラムだと同でしょうか？

```
int result = 100/x;
System.out.println("100/x="+result);
```

一見普通の計算式のように見えます．でも,実は，プログラムを最初から見て言ったら実はこんなことが書いてあるかもしれません．

```
int x = 0;
//色々な処理
int result = 100/x;
System.out.println("100/x="+result);
```

このとき,色々な処理の中で x が 0 以外の数値になっていなければ，結局 100 を 0 で割っていることになってしまい，エラーメッセージが出てきてプログラムが終了してしまいます．

このように変数を使って割り算を計算する場合は，その変数が 0 にならないかどうか常にチェックする必要があります．

一般的には，事象で勉強する if 文を使って，

```
if(x != 0){
        int result = 100/x;
        System.out.println("100/x="+result);
}
else{
        //x が 0 のときの処理
}
```

という書き方をします．こうすることによって x の値が 0 以外のときは割り算を行って，0 のときは特別な処理を行うようにすることができます．

0 割は，プログラムをしている際に気づきにくいバグの一つです．一歩上行くプログラマを目指す皆さんは，是非割り算を行うときは 0 割に注意してください．

あ，ちなみに当然計算に割り算を使う，%演算子を使った計算でも 0 は使えませんからね．まあ，言わなくても分かると思いますが・・・

3.7.2　整数計算問題

Java の変数は整数と実数で分けて考えているというお話をしましたよね．ここで，計算するときに間違えやすいポイントを一つあげておきましょう．

基本的に，Java で計算をするとき，計算結果は**大きい型**に変換されます．つまり，int と long の計算をすれば，計算結果は long になりますし，double と int で計算すれば，計

算結果は double になります．もちろん，int 型と int 型の計算結果は int 型になります．

```
int x = 10;
int y = -55;
double d = 5.5;
double z = 10.0;

System.out.println(x+y); //int 同士の足し算
System.out.println(x+d); //int と double の足し算
System.out.println(x*z); //int と double の掛け算
```

例えば，この結果は，

```
-45 //int 型
15.5 //double 型
100.0 //double 型
```

となり，それぞれ大きい型にあわせて計算結果が出ていることが分かります．

ちなみに，z の計算に注目してください．z の値は整数 100 であるにもかかわらず，x*z は double 型になります．これは，z が中身としては整数でも double 型に入っているため計算結果も double 型となります．つまり，中身が整数か実数かが問題なのではなくて，どの型の変数になるのかが重要なんですね．

しかしながら，ここでちょっとだけ注意する必要があります．それが，整数同士の割り算です．整数同士の割り算は，必ずしも整数ではなくて実数になることがありますよね．こんなとき，ちゃんと double 型にキャストしないと Java 君はうまく計算してくれません．

ここで，例として携帯電話のメモリを現在何パーセント利用しているかを表示するプログラムを作るとしましょう．

携帯電話のメモリは 1024K バイトあるとします．そして，現在その容量のうち，558K バイト利用しているとしましょう．さて，このとき現在のメモリ利用量は何パーセントでしょうか？

```
int totalMemory = 1024;
int usedMemory = 558;
double usedRate = usedMemory/totalMemory*100;

System.out.println("メモリ容量="+totalMemory);
System.out.println("使用量="+usedMemory);
System.out.println("利用割合="+usedRate+"%");
```

この結果を見てみましょう．

```
メモリ容量=1024
```

```
使用量=558
利用割合=0.0%
```

あれ，半分くらい使っていそうなのに，利用割合が 0%のままです．これでは，いつまでもメモリを使える物と勘違いして，余計なデータを携帯電話にためこんでしまいそうです．

なぜこんなバグが発生してしまったのでしょうか？

割った結果を格納する usedRate は double 型ですから，正しい結果を入れてくれそうな物です．しかしながら， int 型と int 型で計算をした場合，その答えは int 型で返ってくるという決まりがあります．つまり、558/1024 を計算しようとした場合、558 も 1024 も int 型なので、答えも int 型で返そうとします。したがって、答えの 0.549…を無理やり整数型で返そうとするので、小数点以下を切り捨てて 0 が答えとなってしまいます．その結果、0 に 100 を掛けた値が double 型の usedRate に格納され, 出力は 0.0 になってしまうのです．double 型に格納するのは 0 に変換された後なので，いまさら double にしようとしても，0 は 0 なのでもう手遅れなわけですね．

これはかなり間違えやすいのでプログラミングをする場合は注意してください。

このような計算をする場合は、キャスト演算子を使って、以下のように書きましょう。

```
int totalMemory = 1024;
int usedMemory = 558;
double usedRate = (double)usedMemory/totalMemory*100;

System.out.println("メモリ容量="+totalMemory);
System.out.println("使用量="+usedMemory);
System.out.println("利用割合="+usedRate+"%");
```

こうすることで，最初の割り算の段階で，double と int の割り算になりますので，結果も double 型で返されるようになります．したがって，userdRate にも double 型の値が格納されて，

```
メモリ容量=1024
使用量=558
利用割合=54.4921875%
```

と，ただしく出力されるようになりました．これでメモリの使いすぎは回避できます．いらない写真データを削除する決心もつきました．

というわけで,この int 同士の割り算によるミスは非常に発生確率が高いです。おそらく Java のプログラミングをしたことがある人ならば一度くらいは誰でも経験しているのではないでしょうか。統計とっていないから分かりませんが.

皆さんは今後の長い Java 人生で一度もこのミスを犯すことなく過ごして行けるよう気をつけてください。

ちなみに，筆者はしょっちゅうこのミスをやらかします．あいたたた．

3.7.3　忘れがちな初期化

変数は宣言と値の代入の二つを行って初めて使うことが出来るようになります．このとき，初めて値を代入することを初期化といいます．

初期化を行わないと，変数名をつけたロッカーをどこかに確保するぞ，ということだけ宣言しておいて，肝心のロッカーの場所が分からないという状態になってしまいます．

```
int x;
System.out.println(x); //x の中身が分からない！
```

初期化を行っていないコードをコンパイルしようとすると，エラーが発生してコンパイル自体に失敗してしまいます．

コンパイルに失敗すると言うことは，バグがある状態でソフトをリリースしてしまう心配などはないので，ある意味安心です．でも，やっぱり最初からバグがないに越したことはありませんよね．というわけで，初期化は忘れずにやるようにしましょう．

ちなみに，初期化を忘れがちな場合に，次に示すような条件分岐が含まれた初期化があります．

ここでは，パケット使用量が 100 パケット以下基本パケット代が 1000 円，パケット使用量が 100 パケットを超えたら基本パケット代が 500 円になるプランを表現している物としましょう．

```
int basePacketPrice; //基本パケット料金
if(packet <= 100){ //パケット使用量が 100 以下なら・・・
        basePacketPrice = 1000;   //基本パケット料金は 1000 円
}
else if(packet > 100){ //パケット使用量が 100 以下なら・・・
        basePacketPrice = 500; //基本パケット料金は 500 円
}
System.out.println("基本パケット使用量:"+basePacketPrice);
```

ここで，初出の if という文がありますが，これは次章で説明するので余り気にせずに，変数 packet の値によって条件分岐すると思ってください．

この場合，変数 packet の値が 100 以上か以下で basePacketPrice はちゃんと初期化されているように見えます．

しかしながら，Java 君はわりとおろかなので「packet の値が 100 以下でも 100 より大きくもないとき初期化されていないぞ！」と思ってしまいます．そんな数字は無いんですが，それを判断することが出来ません．

というわけで，このプログラムをコンパイルしようとすると，エラーが発生してしまいます．このようなコンパイルエラーが発生したら，どこかに見落としがないかどうかチェ

ックして，ちゃんと初期化が行われるようにプログラムを修正しましょう．

ちなみに，どうしても面倒くさい場合は，

```
int basePacketPrice = 0; //基本パケット料金・初期値は 0
if(packet <= 100){ //パケット使用量が 100 以下なら・・・
        basePacketPrice = 1000;   //基本パケット料金は 1000 円
}
else if(packet > 100){ //パケット使用量が 100 以下なら・・・
        basePacketPrice = 500; //基本パケット料金は 500 円
}
System.out.println("基本パケット使用量:"+basePacketPrice);
```

このようにしてしまえば，必ず最初は 0 に初期化されます．最初の内はこのように変数を宣言したら必ず値を代入するようにしておけばコンパイルエラーは防げるでしょう．ただし，0 に初期化したことを忘れてプログラムを書いていってしまう可能性もありますので，初期化の値が重要な変数の場合は最初に適当な値を代入せずに，ちゃんと条件にあった初期化をおこなうようにしましょう．

え？なんだかよくわからないって？

まあ，いずれ分かるときが来ると思いますよ．「なんか変数の値がおかしいなあ，プログラムがうまく動かないなあ」と悩むときがあったら，初期化のことをちょっと思い出してください．