Soft Forum P.C. Club
プログラミング講習　2
2.1 データ型　数値
Sample002 は変数を宣言し表示するプログラムですが、変数のデータ型が空白(_)となっています。変数の宣言に応じたデータ型を考えプログラムを完成させてください。（ヒント：各変数は最適なデータ型の頭文字となっています。）

```
public class Sample002{
public static void main(String[] args){
_____ s=200;
___ i=-50000;
____ l=12300345000l; //最後は小文字の L
______ d=2.3;
_____ f=-2.3f;
System.out.println("変数 s の値は"+s+"です");
System.out.println("変数 i の値は"+i+"です");
System.out.println("変数 l の値は"+l+"です");
System.out.println("変数 d の値は"+d+"です");
System.out.println("変数 f の値は"+f+"です");
}
}
```

表 2.1 Java 言語のデータ型一覧

	データ型	使用バイト数	範囲
整数型	byte	1 byte	-128 ~ 127
	short	2 byte	-32768 ~ 32767
	int	4 byte	-2147483648 ~ 2147483647
	long	8 byte	-9223372036854775808 ~ 9223372036854775807
浮動小数点型	float	4 byte	±3.40282347E+38 ~ ±1.40229846E-45
	double	8 byte	±1.79769313486231570E+308 ~ ±4.9406564841246544E-324
文字	char	2 byte	'¥u0000' ~ '¥uffff'
論理値	boolean	1 byte	true / false

使用するバイト数が多いデータ型ほど扱えるデータの範囲は広くなります。
Sample002 の各値は表 2.1 の範囲を参照すると、それぞれ short、int、long、double、float が最適であることがわかります。なお例えば変数 s を int で宣言するのも間違いではありません。
ただし必要以上に広い範囲を持つデータ型を定義すると、メモリを無駄に使ってしますので実際にプログラムを作成するときは変数に格納するデータの範囲を考慮し適切なデータ型を選びましょう。

また、変数 l と f の値の最後がそれぞれ【l】と【f】になっていますが、long 型の数値の最後には【l】を、float 型の数値の最後には【f】をつける決まりになっています。

Println メソッドで、複数のデータをまとめて出力する場合は【+】でつないで記述します。
Sample002 では【""】で囲まれた文字列と変数とを+でつなぐことでまとめて表示するようにしています。【""】で囲まれた部分の文字はそのまま表示されますが、囲まれていない部分では変数に代入された値が表示されます。
ただし【+】の使い方には少し注意が必要な場合があります。

```
public class Sample003{
public static void main(String[] args){
int i=-2;
System.out.print("計算結果=");
System.out.println(i+1); //(1)
System.out.println("計算結果="+i+1); //(2)
System.out.println("計算結果="+(i+1)); //(3)
}
}
```

【+】は複数のデータの出力の連結に使用しますが、引数が数値のみの場合(1)は加算演算子の【+】として扱われ、計算結果が出力されます。数値計算の【+】と文字連結の【+】を同時に使う場合(2 or 3)は、数値計算部を【()】で囲みます(3)。
また、それぞれのデータ型で扱える数値の範囲が異なるため、扱える範囲の大きいデータ型の変数を、範囲の小さいデータ型の変数に代入しようすると、データの精度が落ちる可能性があるためエラーになります。次の Sample004 をみてみましょう。

```
public class Sample004{
public static void main(String[] args){
```

```
int i=-2;
short j=i+3;
int k=i+3.2;
}
}
```

Sample004 を実行しようとするとエラーが出て実行できません。
これは、int 型の変数 i をより範囲の小さい short 型の変数 j に、実数である 3.2 と変数 i の和を int 型の変数 k にそれぞれ代入しようといているためです。
以上のことから、変数の代入に関しては、基本的に同じデータ型かより大きい範囲のデータ型へのみ許されることを覚えてください。

変数を宣言すると、変数に特定の値を記憶させることができます。このことを値を代入するといいます。代入するには Sample005 のように【=】という記号を使って記述します。

```
public class Sample005{
public static void main(String[] args){
int i=2;
System.out.println("変数 i に"+i+"を代入しました");
i=50000;
System.out.println("変数 i の値を"+i+"に変更しました");
}
}
```

なお Sample005 のように一度値を代入した変数に再度新しい値を代入することで、変数の値を書き換えることができます。
2.2 データ型　文字と文字列

```
public class Sample006{
public static void main(String[] args){
char c='a';
String str="abc"
System.out.println("変数 c に"+c+"を代入しました");
System.out.println("変数 str に"+str+"を代入しました");

System.out.println("文字コード¥¥u0061 は"+'¥u0061'+"を表します");
```

```
}
}
```

文字型のデータ型は char です。文字は変数へ代入するときに【' '】で囲みます。また文字は文字コードを使って表現することもできます。この場合も文字コードを【' '】で囲みます。コンピュータの世界では、それぞれの文字とそれを表す文字コードの対応が決められています。文字コードの種類は複数あり、環境やソフトにより異なります。Java で使用する文字コードは Unicode で'¥uXXXX'という形式で表します。XXXX の部分は 4 桁の 16 進数です。

文字列型データ型は String です。文字列は変数へ代入するときに【" "】で囲みます。

改行などの特殊な意味を持つ文字は、【¥n】というように最初に【¥】をつけて表現します。これらをエスケープシーケンスといいます。Sample005 では【¥】を表示させるために【¥¥】というエスケープシーケンスを使用しています。

エスケープシーケンス一覧

エスケープシーケンス	意味
¥'	引用符【' 】
¥"	二重引用符【"】
¥¥	¥記号【¥】
¥b	1 文字戻る
¥f	ページ送り
¥n	改行
¥r	リターン
¥t	水平タブ
¥XXX	8 進数表記(XXX は 3 桁までの 8 進数表記)
¥uXXXX	Unicode（XXXX は 4 桁の 16 進数）

エスケープシーケンスは 2 文字以上で構成されていますが、処理上は 1 文字として扱われています。よって単独で用いる場合は【' '】で囲みます。

Sample007 では文字列に関するいくつかのメソッドが使用されています。

```
public class Sample007{
public static void main(String[] args){
int i,len;
char c;
String str;
str="abc";
```

```
System.out.println(“変数 str に”+str+”を代入しました”);
len=str.length();
System.out.println(“変数 str の長さは”+len+”です”);
c=str.charAt(0);
System.out.println(“変数 str の先頭の文字は”+c+”です”);
str=str.toUpperCase();
System.out.println(“変数 str を大文字にしました：”+str);
str=str.toLowerCase();
System.out.println(“変数 str を小文字に戻しました：”+str);
}
}
```

文字列の長さは、length メソッドを使って求めます。
文字列.length()
Sample007 では int 型の変数 len に length メソッドの戻り値を代入しています。
charAt メソッドは、戻り値として index に指定した位置の文字を返します。文字列の位置は、先頭から 0,1,2,…と数えます。を使って求めます。
文字列.length(index)
Sample007 では index が 0 なのでい 1 文字目の【a】を char 型の変数 c に代入しています。
toLowerCase メソッドは文字列をすべて小文字にし、toUpperCase メソッドは文字列をすべて大文字にします。
文字列. toLowerCase()
文字列. toUpperCase()

キーボードからの入力
キーボードからの入力を受け付ける方法を学ぶことで柔軟なプログラムが記述できるようになります。

```
import java.io.*;
public class Sample008{
public static void main(String[] args) throws IOException{
System.out.println(“文字列を入力してください”);
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str=br.readLine();
System.out.println(str+”が入力されました”);
```

```
System.out.println(“整数を入力してください”);
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str=br.readLine();
int num=Integer.parseInt(str);
System.out.println(num+”が入力されました”);
}
}
```

Sample008 を実行すると、【文字列を入力してください】というメッセージが画面に出力されます。そしてコンピュータはキーボードからの入力を待つ状態になります。ここでキーボードから文字列を入力し Entet キーを押すと、【（入力した文字列）が入力されました】と出力されます。

```
import java.io.*;
public class Sample009{
public static void main(String[] args) throws IOException{
System.out.println(“整数を入力してください”);
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str=br.readLine();
int num=Integer.parseInt(str);
System.out.println(num+”が入力されました”);
}
}
```

キーボードの数字のキーを押しても、コンピュータは数値ではなく、文字列が入力されたと認識します。よってプログラムの中でキーボードから入力した数値を扱いたい場合は、String 型の文字列から int 型の整数に変換する必要があります。
Sample009 では
int num=Integer.parseInt(str);
で文字列型の変数 str が int 型に変換され、int 型の変数 num に代入されています。
なお入力した文字列を double 型の数値に変換したい場合には
double num=Double.parseDouble(str);
とします。

```
import java.io.*;
```

```
public class Sample010{
public static void main(String[] args) throws IOException{
System.out.println(“整数を 2 つ入力してください”);
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str1=br.readLine();
String str2=br.readLine();
int num1=Integer.parseInt(str1);
int num2=Integer.parseInt(str2);
System.out.println(num1+”と”+num2+”が入力されました”);
}
}
```

Sample010 では readLine()の文を 2 つ記述することでキーボードからの入力を 2 回行っています。最初に入力したものを変数 num1 に、次に入力したものを変数 num2 にそれぞれ数値に変換し代入しています。

コンピュータはいろいろな処理を、計算することによって行います。
Java 言語ではこの計算は演算子とオペランドとからなる式により行います。
ここでは数式 1+2 を例にして考えていきます。
【+】が演算子にあたります。
演算の対象となる【1】と【2】がオペランドにあたります。また【1+2】を計算することが式の評価にあたります。評価あれたあとの【3】を式の値といいます。

```
public class Sample011{
public static void main(String[] args){
System.out.println(“1+2 は”+(1+2)+”です”);

System.out.println(“3-4”は”+(3-4)+”です”);
System.out.println(“5*6 は”+(5*6)+”です”);
System.out.println(“7/8 は”+(7/8)+”です”);
System.out.println(“9%10 は”+(9%10)+”です”);
}
}
```

Sample011 では四則演算とあまりを求める計算を行っています。
Java 言語では掛け算の演算子として【*】を、割り算の演算子として【/】を用います。ま

た【%】は余りを評価する演算子として用いられます。

```
public class Sample012{
public static void main(String[] args){
int num1=2;
int num2=3;
int sum=num1+num2; //(1)
System.out.println(“変数 num1 の値は”+num1+”です”);
System.out.println(“変数 num2 の値は”+num2+”です”);
System.out.println(“num1+num2 の値は”+sum+”です”);
num1=num1+1;
System.out.println(“変数 num1 の値に 1 を足すと”+num1+”です”);
num2+=1;
System.out.println(“変数 num2 の値に 1 を足すと”+num2+”です”);

}
}
```

変数をオペランドとして式を記述することができます。

int sum=num1+num2;

変数 num1 と変数 num2 に記憶されている値同士の足し算を行い、その結果を変数 sum に代入しています。

num1=num1+1

変数 num1 の値に 1 を足し、その値を変数 num1 に代入しています。

右辺と左辺がつりあっていない表記にもみえますが、Java 言語において【=】は等しいという意味の等号ではなく、左辺の変数に右辺の値を代入することを表しています。【=】を代入演算子といいます。

num2+=1

変数 num2 の値に 1 を足し、その値を変数 num2 に代入しています。

【+=】も代入演算子のバリエーションのひとつです。

ここまでは、オペランドがすべて int 型である場合を考えてきました。つづいてオペランドが異なる複数の変数を用いる場合を考えます。

```
public class Sample013{
public static void main(String[] args){
```

```
int inum1=1;
double dnum1=2.34;
dnum1=inum1;
System.out.println("double 型の変数 dnum1 に int 型の変数 inum1 を代入しました¥ndnum1 の値は"+dnum1+"です");
int inum2=5;
double dnum2=6.78;
inum2=(int)dnum2;
System.out.println("int 型の変数 inum2 に double 型の変数 dnum2 を int 型に変換し代入しました¥ninum2 の値は"+inum2+"です");
int inum3=9;
double dnum3=1.23;
double sum=inum3+dnum3
System.out.println("変数 inum3 と変数 dnum3 の和は"+sum+"です");

}
}
```

line5 Java 言語では大きな範囲の型の変数に小さな範囲の型の値を代入することができます。

line9 Java 言語では小さな範囲の変数に大きな範囲の型の値を代入しようとするとエラーが生じます。このような場合は、指定した式の型を【()】内で指定した型に変換するキャスト演算子を用いる必要があります。

キャスト演算子を用いることで、大きな範囲の型を小さな範囲の型に変換することができますが、その型で表せない部分は切り捨てられることになります。

line13 Java 言語では一般的に演算子に 2 つの異なるオペランドを記述した場合は、一方のオペランドを大きな範囲の型に変換してから演算が行われます。よって得られる式の値も大きい範囲のものとなります。