Java入門

1.プログラムの作成と実行

Javaプログラムの記述

- Javaのプログラムは、Javaプログラミング言語のルールに従って、一連の命令をテキスト形式で記述する。
- これらの命令を「ソースコード」と呼び、記述したファイルは「ソースファイル」と呼ばれる。ソースファイルはテキストエディタを使用して作成・編集ができる。
- ソースファイル名や、ソースファイルに記述する文字は、一般的に、半角英数字を使用する。英字の大文字・小文字は区別される。
- ファイル名や、プログラムの名前など、名前(識別子)を付けるときの"一般的"なルールは次のようになっている。
  - Unicode文字を使用できる(一般的な開発慣行として、半角英数字が使われる)。
  - 数字を先頭にすることはできない。
  - 空白やハイフンは使用できない。
  - Javaの予約語も使用できない。

Javaプログラムの記述

- クラスと、その中にあるmainメソッドの形式は次のようになる。

```
public class クラス名{
    public static void main(String[] args){
       実行する命令文
    }
}
```

- ソースコードを保存するときのソースファイルの名前は、(原則として)クラスの名前を使い、拡張子として「.java」をつける。
- 例えばSampleという名前のクラスを作成したときには、ソースファイルの名前はSample.javaとする。

文字や数値の表示

- 文字や数値を画面に表示するには「System.out.println」という命令を使う。丸括弧内には表示したい文字列や数値を指定する。

```
System.out.println(表示する文字列や数値);
```

- 各命令行(ステートメント)の終わりにはセミコロン「;」をつける。
- Javaで文字の並びのデータである文字列を表現するには、文字の列を二重引用符(ダブルクォートまたはダブルクォーテーションという)「"」で囲んで作成する(C/C++と同じ)。

```
"Hello"
```

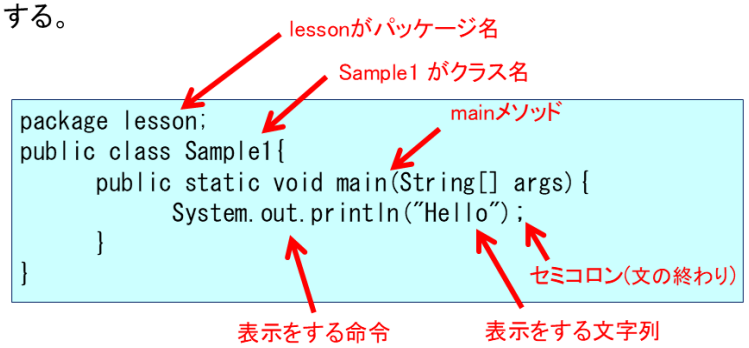
パッケージ

- Javaではクラス単位でプログラムを作成するが、そのクラスにグループ名をつけておくことができる。このグループをパッケージと呼んでいる。
- クラスをパッケージに入れるにはクラス宣言の前にパッケージ宣言を記述する。パッケージ宣言は次のようにする。

```
package パッケージ名;
public class クラス名{
    public static void main(String[] args){
        ・・・処理・・・
    }
}
```


最初のプログラム

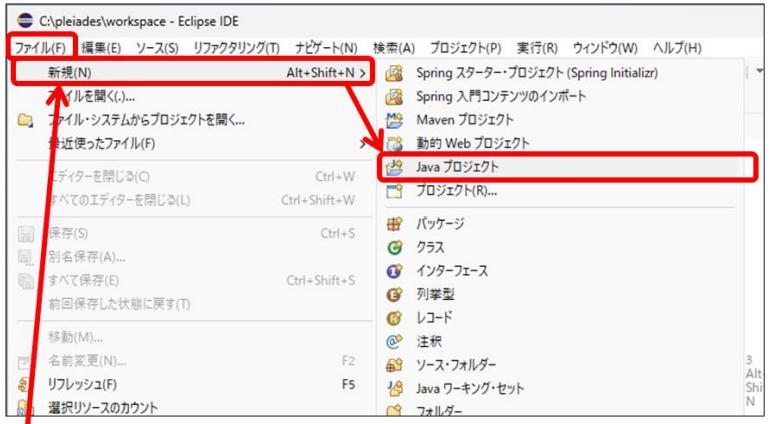
- 最初の例として文字列を表示するプログラムを作成する。
- このプログラムは、パッケージ名をlesson、クラス名をSample1とし、mainメソッドの中には文字列を表示する命令 System.out.println を記述して、文字列「Hello」を指定することにする。



プロジェクトの作成

- Eclipseでプログラムを作成するには、まずプロジェクトを作成する。
- プロジェクトは開発の単位で、複数のファイルやドキュメントなどを管理する。
- プロジェクトを作成してから、プロジェクト内にクラスを作成する。

プロジェクトの作成



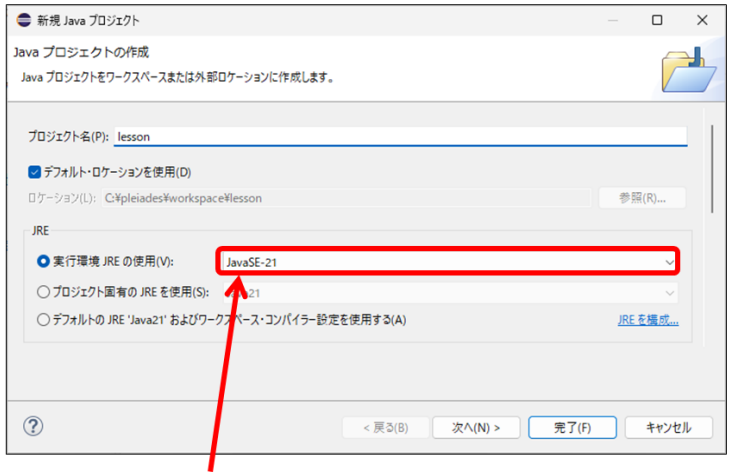
「ファイル → 新規 → Javaプロジェクト」を選択

プロジェクトの作成

プロジェクト名は、ここでは「Java」としておく

9

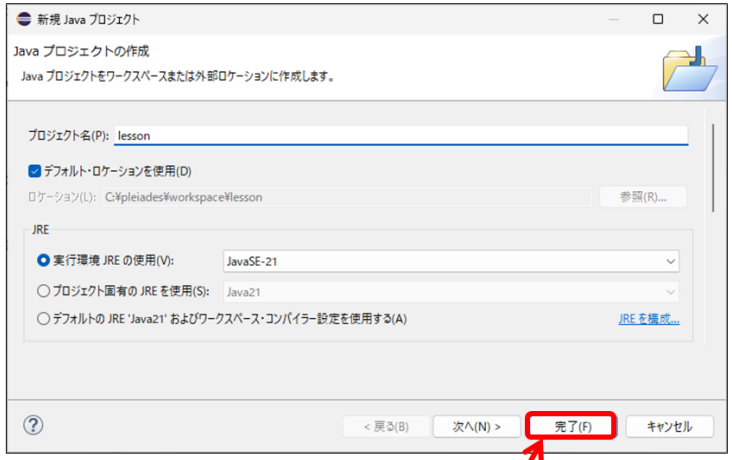
プロジェクトの作成



「JavaSE-21」となっていることを確認

10

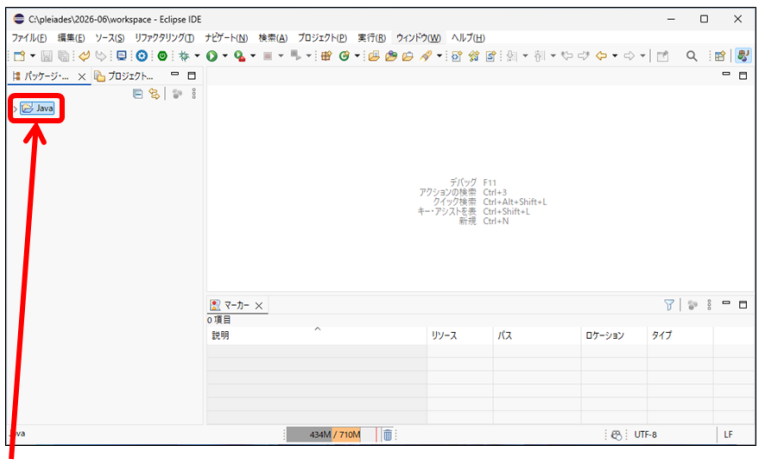
プロジェクトの作成



「完了」をクリック

11

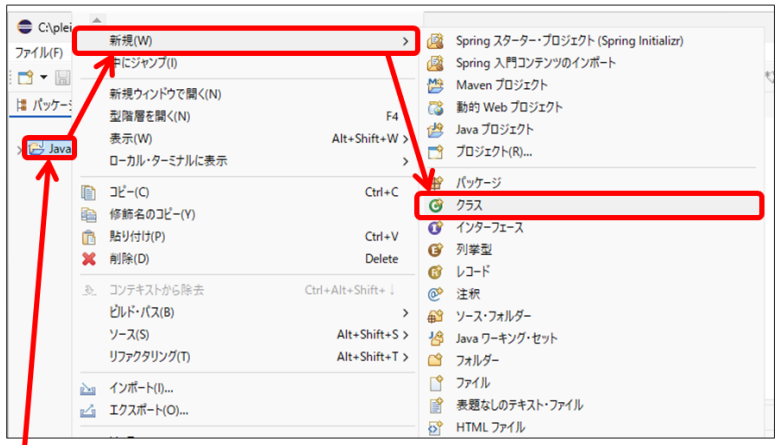
プロジェクトの作成



パッケージ・エクスプローラにプロジェクト名Javaが表示される

12

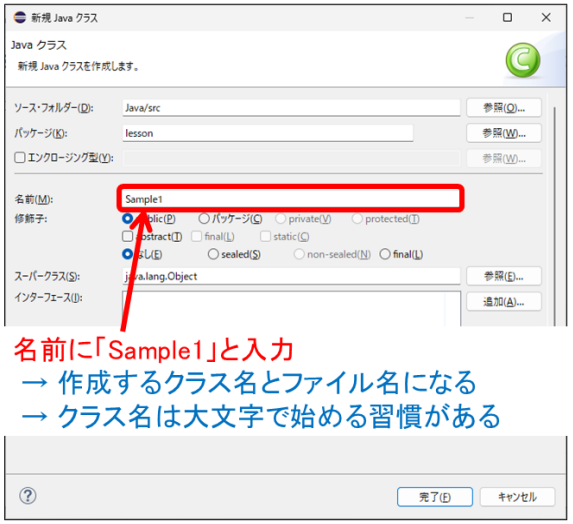
クラスの作成



プロジェクト名「Java」を右クリックして「新規 → クラス」を選択

13

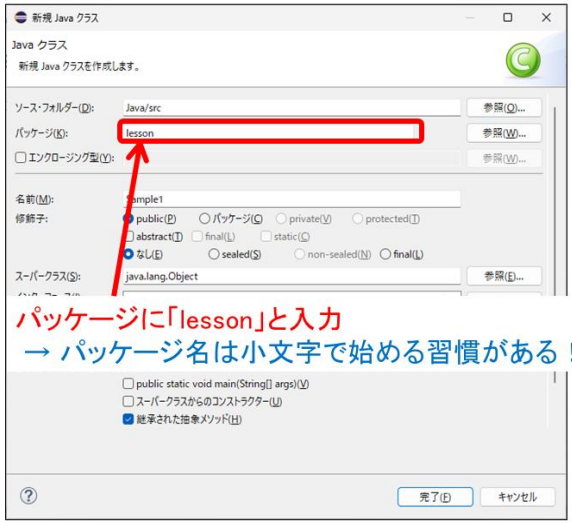
クラスの作成



名前に「Sample1」と入力
→ 作成するクラス名とファイル名になる
→ クラス名は大文字で始める習慣がある

15

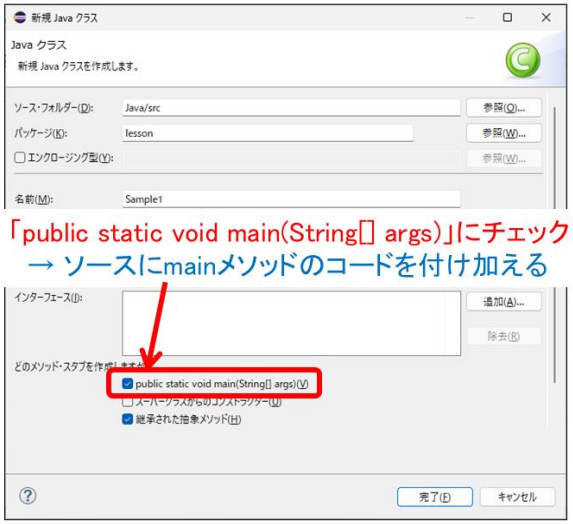
クラスの作成



パッケージに「lesson」と入力
→ パッケージ名は小文字で始める習慣がある！

14

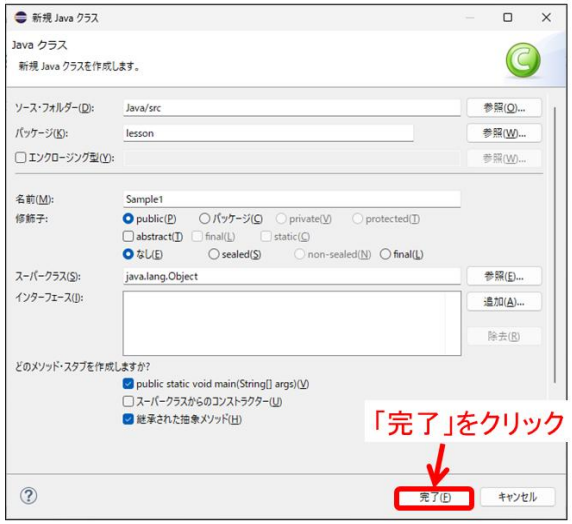
クラスの作成



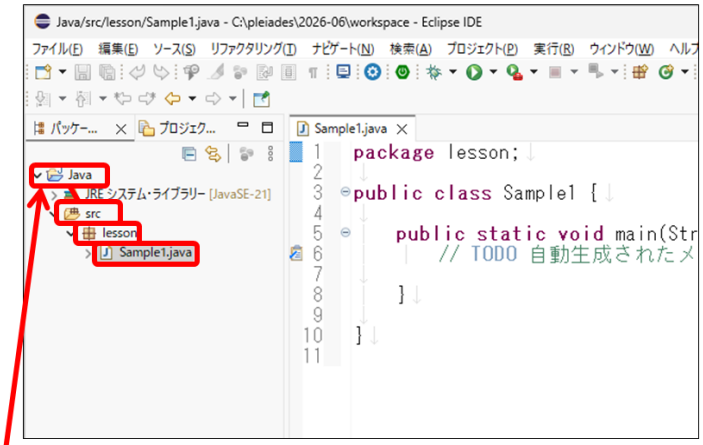
「public static void main(String[] args)」にチェック
→ ソースにmainメソッドのコードを付け加える

16

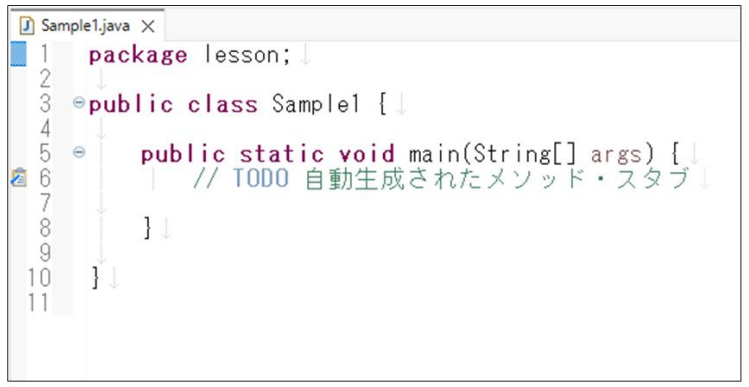
クラスの作成



クラスの作成

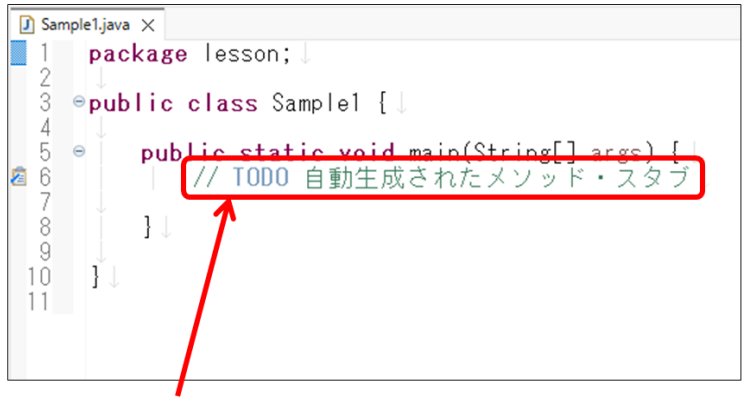


クラスの作成

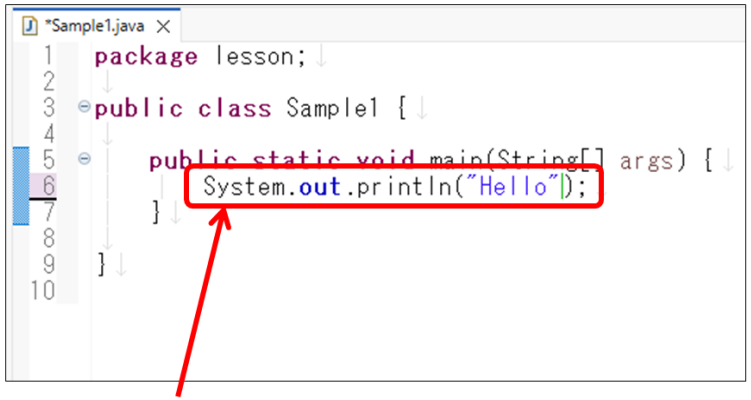


右側にはソース画面が表示され、雛形が作成されている

クラスの作成

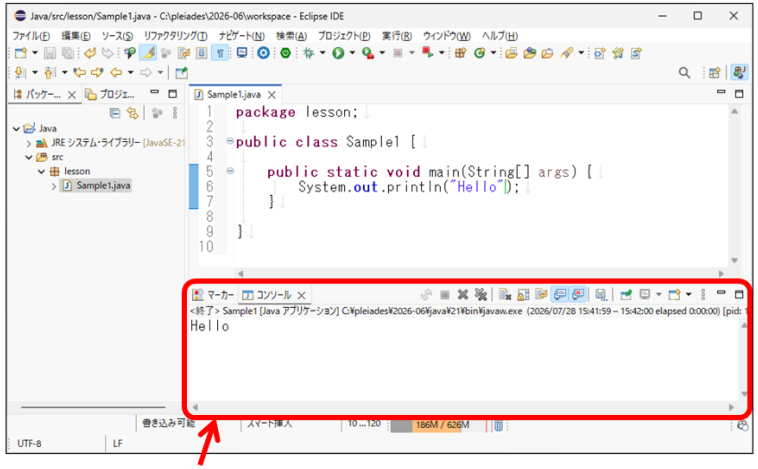


クラスの作成



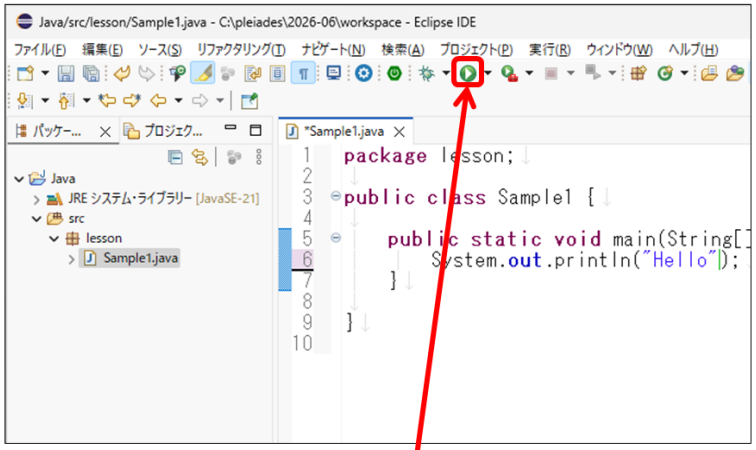
「System.out.println("Hello");」と入力する

実行



コンソールに実行結果「Hello」が表示される

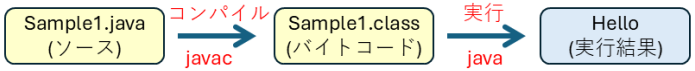
実行



実行ボタン「 」をクリック

コンパイルと実行

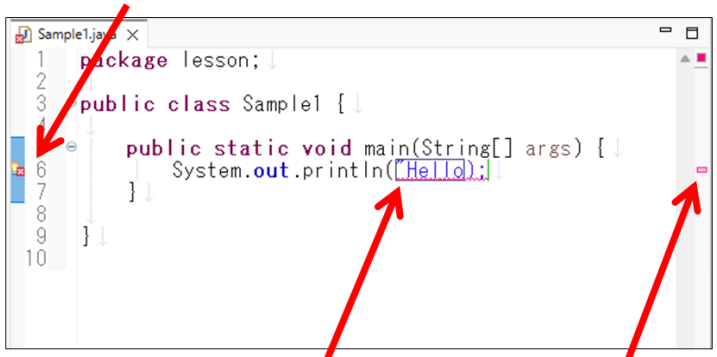
- Javaのソースプログラムが完成しても、そのままでは実行できない。実行するには「コンパイル」という作業をしてソースプログラムをJavaの実行環境(JVM:Java仮想マシーン)が理解できる形式(バイトコード)に変換しなければならない。



- Eclipseでは実行ボタンを押すだけで、内部でコンパイルを行ない、その後で変換したプログラムを実行するようになっている。
- もし、プログラムが間違っている場合には、ソースコード上にエラーを表す波線やマーク等が表示される。そのときはソースコードを見直し、間違いを修正する。コンパイルエラーがあるソースコードは、正常にコンパイルして実行できない。

コンパイルと実行

間違い判断される行に赤い×のマークがつく

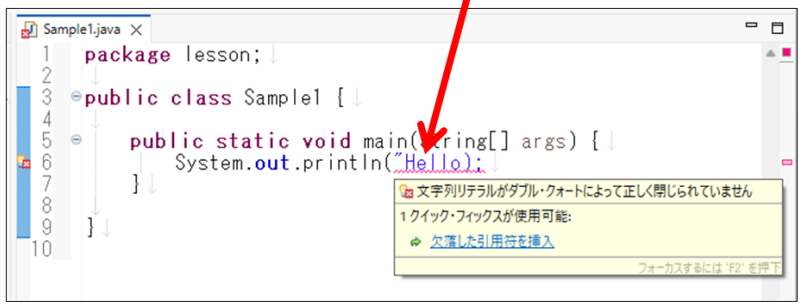


間違いと判断された場所に赤い波線

間違いと判断された位置

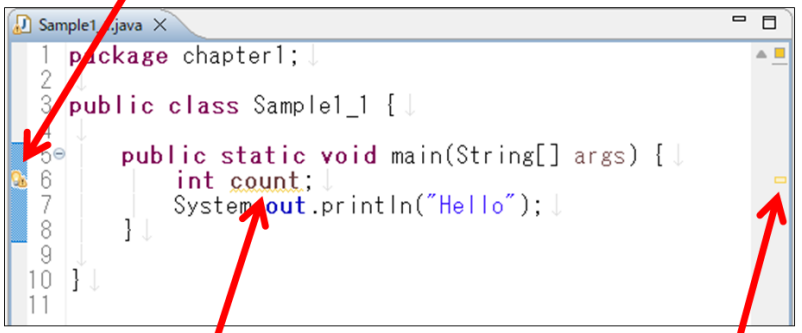
コンパイルと実行

エラーマークをポイントするとその内容を表示



コンパイルと実行

警告のある行に が付く
→警告内容を確認し、できるだけ解消する！



警告の部分に黄色い波線が付く

警告のある位置

問題

① 「こんにちは」と表示するプログラムを作成せよ。ここでパッケージ名はlesson、クラス名はC1Q1と付けることにする。

こんにちは

Java入門

2.基本文法

基本文法

- Javaは、クラスを基本単位としてプログラムを構成する、静的型付けのオブジェクト指向言語となる。変数を使用する際は、原則としてintやStringなどのデータ型を宣言する。文の末尾にはセミコロンを付け、処理のまとまり(ブロック)は波括弧で表す。条件分岐にはifやswitch、繰り返しにはforやwhileを使用する。
- Javaの文法はC/C++に似ているため、基本的な制御構文はほぼ共通している。ただし、JavaにはC/C++のようなポインタ演算がなく、メモリ管理はガベージコレクションによって自動的に行われる。また、多重継承はクラスでは利用できず、インターフェースによって同様の仕組みを実現する。
- PythonはJavaやC/C++と異なり、動的型付け言語であり、変数の型宣言や文末のセミコロンは通常必要ない。ブロックは波括弧ではなくインデントで表す。JavaはPythonより記述量が多い一方、コンパイル時に型の誤りを発見しやすく、大規模なシステム開発に適している。

Javaと他の言語の比較

- 基本的な違い

やりたいこと	Java	C / C++	Python
文の終わり	; セミコロン	; セミコロン	改行
ブロック	{ }	{ }	インデント
変数宣言	int x = 10;	int x = 10;	x = 10
文字列	String	char* / std::string	str
出力	System.out.println(x)	printf / cout << x	print(x)
エントリ	public static void main	int main	任意の位置
メモリ管理	GC (自動)	手動 (new/delete)	GC (自動)

Javaと他の言語の比較

- 基本的なデータ型の比較

種類	Java	C/C++	Python
整数型	int (約21億を扱える)	int (約21億を扱える)	int (何桁でも扱える)
浮動小数点型 (整数以外の数値)	double (10^{308}を扱える)	double (10^{308}を扱える)	float (10^{308}を扱える)
文字列型	String	char配列, char* std::string	str
真偽値型	boolean (true/false)	bool (true/false)	bool (True/False)

- Javaでは変数は、使用前に型を指定して「宣言」を行う必要がある。

```
例. int x;          //宣言のみ
    int y = 10;     //値を代入して宣言
```

文字列の連結とコメント

●Java(そしてC++,Python)では「＋」演算子を使って文字列を連結できる。

```
"Hello" + " world"  → "Hello world"
```

●Javaでは「＋」演算子を使って、文字列と文字列以外の値を文字列として連結することができる。

```
"答えは" + 123  → "答えは123"
```

●Javaのコメントは次の2つの形式がある。

➢/* */ 形式
/* がコメントの始まりで、*/ で終了となる

➢// 形式
//がある行内で、//から後ろがコメントとなる

文字列の連結

Sample2.java

```
/*
 * これは複数行コメントの例です。
 * 基本文法をまとめたサンプルです。
 */
package lesson;
public class Sample2 {
    public static void main(String[] args) { //エントリー
        int age = 25;              // 整数型
        String name = "山田太郎";  // 文字列型
        System.out.println("Hello" + " world");
        System.out.println("私の名前は" + name + "です。");
        System.out.println("年齢は" + age + "歳です。");
    }
}
```

実行結果

```
Hello world
私の名前は山田太郎です。
年齢は25歳です。
```


算術演算子

●Javaで四則の計算を行うには次の演算子を使う。

演算子	記入例	説明
+	x + y	xとyを加算
−	x − y	xからyを減算
*	x * y	xをyで乗算
/	x / y	xをyで除算した商
%	x % y	xをyで除算した余り

注.Javaでは整数どうしの割り算は、小数点以下が切り捨てられて、結果も整数となる。

比較演算子

●数値の大小や等価を判定する比較演算子として次が用意されている。これらは、条件が成り立てばtrue、そうでなければfalseを返す。

演算子	記入例	説明
>	x > y	xがyよりも大きい場合にtrue
>=	x >= y	xがyよりも大きいか等しい場合にtrue
<	x < y	xがyよりも小さい場合にtrueを返す
<=	x <= y	xがyよりも小さいか等しい場合にtrue
==	x == y	xとyが等しい場合にtrue(文字列はNG)
!=	x != y	xとyが等しくない場合にtrue

●Java文字列の等価は演算子ではなく equals() メソッド使う。比較する2つの文字列の内容が等しい場合にはtrueを返す。

```
文字列1.equals(文字列2)
```

インクリメント演算子

●「++」はインクリメント演算子と呼ばれ、数値を扱う変数に適用でき値を1増加させる。++は変数の前に記述する前置と、変数の後に記述する後置がある(前置・後置は、代入など他の演算と一緒に使用した場合に動作が異なる場合がある)。

演算子	記入例	説明
++(前置)	++x	++x は、xを1増加させた後に、xを評価する
++(後置)	x++	x++ は、xを評価した後に、xを1増加させる

例. int x = 10;
x++; // xの値を+1する(++xでも可)
System.out.println(x); →「11」と表示される。

●Pythonにはインクリメント演算子がないので「x += 1」とする。

if 文

●if文の基本ロジックは、C/C++やPythonと同じだが、書き方と条件式で許容される型(評価判定)に違いがある。

●Javaでは、条件式には必ず boolean(true/false)を返す式を渡す必要がある。数値や文字列などは指定できない。

Java、C/C++

```
if (score >= 80) {
    ・・・
}
else if (score >= 60) {
    ・・・
}
else {
    ・・・
}
```

Python

```
if score >= 80:
    ・・・
elif score >= 60:
    ・・・
else:
    ・・・
```


for 文

●Javaのfor文は C/C++と同様だが、Pythonではイテレータ(要素の反復処理)ベースの構造で大きな違いがある。

Java、C/C++

```
for (int i = 0; i < 5; i++) {
    ・・・
}
```

Python

```
# range(5) で 0 から 4 までのシーケンスを生成
for i in range(5):
    ・・・
```


while 文

●Javaのwhile文は C/C++と同様だが、Pythonでは若干書き方が異なる。また、JavaとC/C++ではdo-while文が使用できるがPythonでは使用できない。

Java、C/C++

```
int count = 0;
while (count < 3) {
    ・・・
    count++;
}
```

Python

```
count = 0
while count < 3:
    ・・・
    count += 1  # ※Pythonに「count++」は存在しない
```

if文とfor文

Sample3.java

```
package lesson;
public class Sample3 {
    public static void main(String[] args) {
        String name = "Java";
        if (name.equals("Java")) {
            System.out.println("Javaと一致");
        }else if(name.equals("C/C++")) {
              System.out.println("C/C++と一致");
        }
        else {
              System.out.println("言語はその他");
        }
        for (int i = 0; i < 3; i++) {
            System.out.println(i);
        }
    }
}
```

実行結果

```
Javaと一致
0
1
2
```


配列

●Javaの配列の基本的な構文は C/C++ と同様だが(C++では、固定長ならstd::arrayも使える)、PythonにはJavaやCのような固定長配列はなく、標準の list を使う。

Java(変数宣言時の[]は型名の直後に書くのが一般的)

```
int[] arr1 = {10, 20, 30};   // 値を代入して作成
int[] arr2 = new int[3];     // 要素数3の配列を作成
arr2[0] = 100;
```

C/C++

```
int arr1[] = {10, 20, 30};   // 値を代入して作成
int arr2[] = new int[3];     // 要素数3の配列を作成
arr2[0] = 100;
```

Python

```
arr1 = [10, 20, 30]   # 値を代入して作成
arr2 = [0] * 3        # Javaの new int[3] に相当
arr2[0] = 100
```


配列の繰り返し(1)

●インデックス指定で繰り返し

Java(要素数は「.length」で取得できる)

```
int[] arr = {10, 20, 30};
for (int i = 0; i < arr.length; i++)
    //arr[i]で参照
```

C/C++

```
int arr[] = {10, 20, 30};
int size = sizeof(arr) / sizeof(arr[0]); //要素数
for (int i = 0; i < size; i++)
    //arr[i]で要素を参照
```

Python(要素数はlen()関数を使って取得する)

```
arr = [10, 20, 30]
for i in range(len(arr)):
    # arr[i]で参照
```


配列の繰り返し(2)

●値の参照で繰り返し

Java(for-each文;拡張for文)

```
int[] arr = {10, 20, 30};
for (int x : arr)
    //xで要素を参照
```

C++(範囲for文)

```
int arr[] = {10, 20, 30};
for (auto x : arr)
    //xで要素を参照
```

Python

```
arr = [10, 20, 30]
for x in arr:
    #xで要素を参照
```

配列と繰り返し

Sample4.java

```
package lesson;
public class Sample4 {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30};
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }

        for (int n : numbers) {
            System.out.println(n);
        }
    }
}
```

実行結果

```
10
20
30
10
20
30
```


メソッド(関数)定義の違い

- Javaはメソッド(関数)をクラス内のメソッドとして定義し、引数と戻り値の型を明記する必要がある。
- Javaaは型指定が必須な点はC/C++同じだが、クラスの外に独立した関数を定義できない。
- Pythonは「def」キーワードを用いて定義し、クラス内外どちらでも記述できる。さらに動的型付けであるため型宣言が基本的には不要であり、波括弧の代わりにインデントを用いて範囲を表現する。
- メソッドを定義するときstaticキーワードを付けるとインタンスを生成せずに利用できるが、staticキーワードがつかないときにはインタンス生成を行ってからコールする。

メソッド(関数)定義の違い

Java(クラスの中で定義)

```
public class Calculator {
    //インスタンス生成なしで使えるようにするにはstaticを付ける
    static int add(int a, int b) {
        return a + b;
    }
}
```

C/C++

```
int add(int a, int b) {
    return a + b;
}
```

Python

```
def add(a, b):
    return a + b
```


キャスト

- Javaでは小さな数値を扱う型から大きな数値を扱える型への変換(例えばintからdouble)は自動で起こるが、その逆は文法間違いとなる。
- 型を強制的に変換するには次のように値の前に()で変換したい型を指定する。この強制的な変換を「キャスト」と言う。

```
(変換したいデータ型)値
```

```
例.double x = 10.0;
  //int y = x; //Javaでは文法間違い！
   int y = (int)x; //doubleをintに代入するにはキャストが必要
```

メソッド定義とキャスト

Sample5.java

```
package lesson;
public class Sample5 {
   public static void main(String[] args){
     String value = label("えんぴつ", 50);
     System.out.println( value );
   }
   static String label(String goods, int price){
      int total = (int)(price * 1.1);
      String s = "商品名:" + goods + " 税込価格:" + total;
      return( s );
   }
}
```

実行結果

```
商品名:えんぴつ 税込価格:55
```


Java入門

3.クラス

クラスとオブジェクト

●クラスはデータの入れ物の設計図と考えることができる。例えば、次のような会員情報を保持することを考える。

```
会員番号：1230
名　　前：山田太郎
```

●この情報を扱うためにMemberというクラスを作ることにする。例えば次のようにする。

```
public class Member{
    int no;       //会員番号を入れる変数
    String name; //名前を入れる変数
}
```

ここでは、no、nameという変数でそれぞれ会員番号、名前を保持できるようにしている。クラス内(かつメソッドの外)の変数は「フィールド」とも呼ばれる。

クラスとオブジェクト

●クラスはプログラムの設計図なので、動かすためにはこの設計図から実際の"モノ"を作る必要がある。

●"モノ"を作るには「new演算子」を使い次のようにする。

```
new クラス名();
```

●Javaではこの作成した"モノ"は「オブジェクト」と呼ばれる。

●オブジェクトは、それを生成するときに使ったクラスの型の変数に入れて利用できる。クラスMemberのオブジェクトを生成して変数xで使えるようにするには次のようにする。

```
Member x = new Member();
```

クラスとオブジェクト

設計図

```
class Member {
    int no;
    String name;
}
```

new Member()

設計図から作られたモノ(オブジェクト)

```
Memberオブジェクト
  int no;
  String name;
```

作成された(モノオブジェクト)には、設計図どおりに、
変数(フィールド) no や name が作りこまれている。

クラスとオブジェクト

- 生成したMemberクラスのオブジェクトには、設計図のとおりその中にno、nameという変数(フィールド)が作り込まれている。
- このオブジェクト内に作り込まれた変数を使うには、生成したオブジェクトを入れておいた変数の後にドット「.」を書きその後に変数(フィールド)を記述する。

```
オブジェクトを入れた変数.フィールド
```

- Memberクラスのオブジェクトを変数xに代入して使えるようにした場合、オブジェクト内に組み込まれた変数に値を代入するには次のようにする。

```
Member x = new Member();
x.no = 1234;
x.name = "山田太郎";
```


クラスとオブジェクト

Member.java

```
package lesson;

//会員情報を入れるMemberクラスを定義
public class Member{
    int no;       //会員番号を入れる変数(フィールド)
    String name; //名前を入れる変数(フィールド)
}
```


クラスとオブジェクト

Sample6.java

```
package lesson;
public class  Sample6 {
    public static void main(String[] args){
        //Memberオブジェクト生成
        Member x = new Member();

        //オブジェクト内のフィールドに値をセット
        x.no = 1230;
        x.name = "山田太郎";

        //オブジェクト内のフィールドの設定値を表示
        System.out.println(x.no);
        System.out.println(x.name);
    }
}
```

実行結果

```
1230
山田太郎
```

クラスとオブジェクト

```
class Member {
    int no;
    String name;
}
```

設計図

```
Member x = new Member()
```

変数x

オブジェクトの参照
が代入されている

```
Memberオブジェクト
  int no;
  String name;
```

設計図から
作られたモノ
(オブジェクト)

コンストラクターの利用

- 今の例では、オブジェクトを生成して、その後でフィールド内に一つ一つ値を代入したが、オブジェクト生成時に値をセットする方法もある。
- これを行うには「コンストラクター」を使う。コンストラクターはオブジェクトを生成するときに(すなわちnewするときに)呼ばれる特別なメソッドで、オブジェクト生成時に実行したい処理があるときに定義しておいて使うことができる。
- コンストラクターは戻り値が書かれていない以外は通常のメソッドと同じ形式である。
- Javaではコンストラクターは、クラス名と同じ名前にする決まりになっている。
- thisはオブジェクト自身を指し示すキーワードとなる。

コンストラクターの利用

- Memberクラスに、会員番号と名前をセットするコンストラクターを追加する。

Member.java(Memberクラスにコンストラクターを追加)

```
package lesson;
//会員情報を入れるMemberクラスを定義
public class Member{
    int no;       //会員番号を入れる変数(フィールド)
    String name; //名前を入れる変数(フィールド)
    //コンストラクター
    Member(int no, String name){
        this.no = no;       //引数のnoをフィールドのnoに代入
        this.name = name; //引数のnameをフィールドのnameに代入
    }
}
```

thisは現在処理しているオブジェクト自身を表す。

コンストラクターの利用

- new Member(1230, "山田太郎") を実行したときの動き

```
class Member {
    int no;
    String name;
}
```

設計図

```
new Member(1230,"山田太郎")
```

```
Memberオブジェクト
  int no = 1230;
  String name = "山田太郎";
```

設計図から
作られたモノ

コンストラクターの利用

Sample6.java(変更)

```
package lesson;
public class Sample6{
    public static void main(String[] args){
        //Memberオブジェクト生成
        //オブジェクト生成時にコンストラクターを使って値もセット
        Member x = new Member(1230, "山田太郎");

        //オブジェクト内のフィールドの設定値を表示
        System.out.println(x.no);
        System.out.println(x.name);
    }
}
```

実行結果

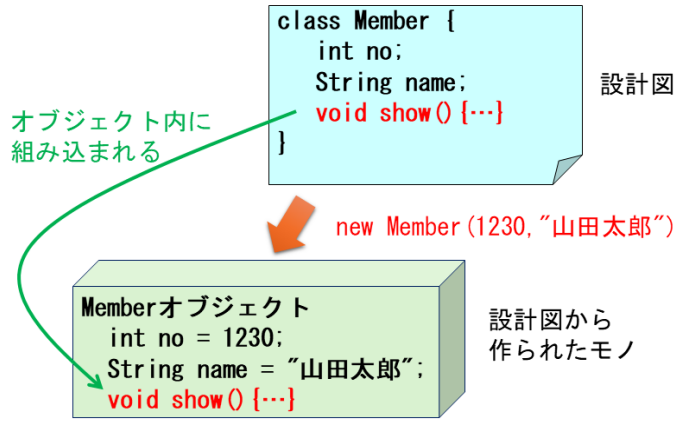
```
1230
山田太郎
```


メソッドの追加

- クラスは、いわば"高機能な入れ物"であり、単に値を保持するだけでなく、さまざまな処理(メソッド)を定義することができる。
- 例えばMemberクラス内に次のようにして会員情報を表示するメソッドshowを追加することもできる。

```
public class Member{
    ・・・(略)・・・
    void show(){
        System.out.println(no);
        System.out.println(name);
    }
}
```


メソッドの追加



メソッドの追加

Member.java(Memberクラスにshowメソッドを追加)

```
package lesson;
class Member{
    int no;       //会員番号を入れる変数
    String name; //名前を入れる変数

    Member(int no, String name){
        this.no = no;
        this.name = name;
    }
    //会員の情報を表示するメソッド（追加）
    void show(){
        System.out.println(no);
        System.out.println(name);
    }
}
```

メソッドの追加

Sample6.java(Sample6クラスを変更)

```
package lesson;
public class Sample6{
    public static void main(String[] args){
        //Memberオブジェクト生成
        //生成時にコンストラクターを使って値もセット
        Member x = new Member(1230, "山田太郎");

        //オブジェクト内のshowを呼び出し情報を表示
        x.show();
    }
}
```

実行結果

```
1230
山田太郎
```


乱数

- ランダムに生成される数を乱数という。一般的にはどの数も同じ確率(あるいは特定の確率分布)に従って生成されることが要求される。
- 乱数はさまざまなプログラムで幅広く利用されている。
- Randomクラスを使って乱数を生成するには次のようにする。
  ① Randomクラスをインポートする(クラス宣言の前に記述)。

```
import java.util.Random;
```

  ② Randomを使う準備をする(変数名は自分でつける)。

```
Random 変数 = new Random();
```

  ③ ②の変数に「.」をつけて、次頁のメソッドを使って乱数値を取得する。

```
変数.乱数値を取得するメソッド
```


乱数

- 乱数値を取得するには次のメソッドを使う(これ以外にもある)。

int nextInt() intの範囲で乱数を返す。
int nextInt(int N) 0以上,N未満のint型の乱数を返す。
int nextInt(int S,int N) S以上,N未満の擬似乱数を返す。(SE17以降)
double nextDouble() 0以上1未満のdouble型の乱数を返す。

注.「N未満」なっているところは、Nを含まない。

乱数

Sample7.java

```
package lesson;
public class Sample7 {
   public static void main(String[] args){
      java.util.Random r = new java.util.Random();
      System.out.println(r.nextInt(100));
   }
}
```

実行結果(毎回異なる)

```
64
```

import宣言

- パッケージ名付きの名前を完全限定名(フルネーム)といい、単なるクラス名を単純名という。
- ソースファイル内でクラス宣言の前、package宣言の後にimport宣言を記述すると、そのファイル内でパッケージ名を省略して単純名でクラスを使うことができるようになる。
- 特定のクラスをインポートするには次のようにする。複数のクラスをインポートするには複数import宣言を記述する。

```
import　パッケージ名.クラス名;
```

- パッケージ内の全てのクラスをインポートするには次のようにする。ここで'*'はワイルドカードと呼ばれる。

```
import　パッケージ名.*;
```


import宣言

Sample7.java(import宣言を使用して書き換え)

```
package lesson;
import java.util.Random;
public class Sample7 {
   public static void main(String[] args){
      Random r = new Random();
      System.out.println(r.nextInt(100));
   }
}
```

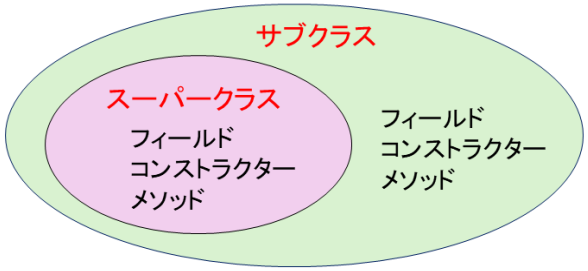
実行結果(毎回異なる)

```
88
```


クラスの継承

- 継承を使うと既存のクラスを拡張して新しいクラスを作ることができる。
- 新しいクラスは、既存のクラスのメソッドとフィールドを継承する。
- 既存のクラスをスーパークラス(または、基本クラス、親クラス)と言い、継承して作成した新しいクラスをサブクラス(または、派生クラス、子クラス)と言う。
- サブクラスはスーパークラスに新しい機能を追加して使うことができる。
- Javaでは継承できるクラスは一つのみに限られる(単一継承)。

クラスの継承



サブクラスは、スーパークラスの機能を受け継いで、より拡張されたクラスを定義する方法である。

クラスの継承

●継承を行うには extends キーワードを使う。これにより、スーパークラスの機能(フィールドやメソッド)を受け継いだサブクラスを作ることができる。

```
class サブクラス名 extends スーパークラス名{
    フィールド
    コンストラクター(引数){
        ・・・・
    }
    メソッド(引数){
        ・・・・
    }
    ・・・
}
```


クラスの継承

Sample8.java(1/2)

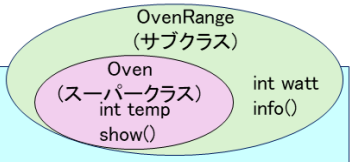
```
package lesson;
public class Sample8 {
   public static void main(String[] args){
      OvenRange or = new OvenRange();
      System.out.println(or.temp); //継承したフィールドを利用
      System.out.println(or.watt); //追加したフィールドを利用
      or.show(); //継承したメソッドを利用
      or.info(); //追加したメソッドを利用
   }
}
```

実行結果

```
200
600
温　度:200
ワット:600
```


クラスの継承

Sample8.java(2/2)




```
class Oven{ //スーパークラス
   int temp = 200;
   void show(){
      System.out.println("温　度:" + temp);
   }
}
class OvenRange extends Oven{ //Ovenを継承したサブクラス
   int watt = 600;
   void info(){
      System.out.println("ワット:" + watt);
   }
}
```


Java入門

4. 日付、例外、ArrayList

日時の扱い

●現在の日付・時刻はjava.time.LocalDateTimeクラスを使って取得できる。

```
//(import java.time.LocalDateTime;が必要)
LocalDateTime now = LocalDateTime.now();
System.out.println(now); //2026-08-12T15:50:36.123456
//秒までの表示にするには次のようにする
//(import java.time.temporal.ChronoUnit;が必要)
System.out.println(now.truncatedTo(ChronoUnit.SECONDS));
```

●日時を特定の書式の文字列に変換するにはString.format()メソッド(他の言語のsprintf()に相当)を使うことができる。ここで「%tF」は日時を「yyyy-MM-dd」の形式に、「%tT」は日時を「HH:mm:ss」の形式に変換して出力する。

```
LocalDateTime now = LocalDateTime.now();
String s = String.format("%tF %tT", now, now);
System.out.println(s); //2026-08-12 15:50:36
```


日時の扱い

Sample9.java

```
package lesson;
import java.time.LocalDateTime;
public class Sample {
   public static void main(String[] args) {
      LocalDateTime now = LocalDateTime.now();
      String s = String.format("%tF %tT", now, now);
      System.out.println(s);
   }
}
```

実行結果

```
2026-08-12 16:47:40
```


例外

●例外は、何か大事なことや予想しなかったこと(たいていはエラー)が生じたときに、制御の流れを変更するための仕組みとなる。

●例外は「配列のインデックスが範囲外」、「ファイルが開けない」などでも発生するし、自分で定義・発生させることもできる。

●例外が発生することを「例外がスローされる」と言う。

●例外の発生する可能性のある処理は、例外の種類により、例外を捕捉(「キャッチ」という)してその対処をする(try～catch)か、上位のメソッドに例外を転送(throws)する必要がある。

●発生した例外がどこにも捕捉されずにJava仮想マシーン(JVM)まで転送されてくると、そのプログラムは実行を停止させられる。

例外の例

Sample10.java

```
package lesson;
public class Sample10 {
   public static void main(String[] args){
      int[] test = new int[5];
      System.out.println("代入します");
      test[10] = 100;
      System.out.println("代入しました");
   }
}
```

実行結果

```
代入します
Exception in thread "main"
   java.lang.ArrayIndexOutOfBoundsException:
      Index 10 out of bounds for length 5
         at lesson.Sample10.main(Sample10.java:6)
```

try-catch文

●例外がスローされたとき、try～catch 文で例外をキャッチして処理を行うことができる。

```
try{ //tryブロック
    例外発生の可能性がある処理
}
catch(例外クラス名 引数){ //catchブロック
    例外が発生したときの処理
}
```

➢tryブロックの中で、catchに指定された例外(あるいはそのサブクラスの例外)がスローされた場合は、tryブロック中の処理を終了してcatchブロックを実行する。catchブロック実行後は、catchブロックの後の処理を続行する。

➢tryブロック中で、例外がスローされなかった場合は、catchブロックは実行されずに、catchブロックの後の処理を続行する。

➢tryブロック中で、catchに指定されていない例外がスローされた場合は、その時点でプログラム(そのスレッド)が終了となる。

try-catch文

Sample10.java(Sample10.javaを変更)

```
package lesson;
public class Sample10 {
   public static void main(String[] args){
     try{
        int[] test = new int[5];
        System.out.println("代入します");
        test[10] = 100;
        System.out.println("代入しました");
     }
     catch(ArrayIndexOutOfBoundsException e){
        System.out.println("配列の要素数を超えています");
        //e.printStackTrace()でエラーメッセージの表示も使われる
     }
     System.out.println("終了");
   }
}
```

実行結果

```
代入します
配列の要素数を超えています
終了
```


ArrayList

●Javaで可変長の配列に相当するものは利用するにはjava.util.ArrayListクラスを使う。

●ArrayListでは要素の追加により内部配列のサイズが足りなくなると自動的に大きなものに作り変えられる。したがって、要素の追加により配列のサイズが足りなくなるという心配はない。

●ArrayListオブジェクトの生成は次のように行う。ここで <型> には格納する値の型を指定する。また、右辺の<型>の型は省略可能である。ここで型には参照型である必要がある(intの場合にはInteger、doubleの場合にはDoubleを指定する)。

```
ArrayList<型> x = new ArrayList<型>();
```

```
ArrayList<型> x = new ArrayList<>();
```


ArrayList

●ArrayListには次のようなメソッドがある(これ以外にもある)。

boolean add(E e) 引数eの値をArrayListに格納する。
boolean remove(Object e) 引数eの値をArrayListから一つ削除する。
boolean contains(Object e) 引数eが格納されているか判定する。
int size() 格納されている要素数を返す。
E get(int index) 引数indexの位置の要素を返す。
E remove(int index) 引数indexの位置の要素を削除する。
E set(int index, E element) 引数indexの位置の要素を引数elementで置き換える。
void clear() すべての要素を削除する。

ArrayListの繰り返し

●ArrayListは配列のように繰り返しを行うこともできる(ここで'変数'にはArrayListのオブジェクトが代入されているものとする)。

```
for(int i = 0 ; i < 変数.size() ; i++ ){
    変数.get(i)でインデックスi番目の要素を参照
}
```

●ArrayListの各要素に対して繰り返しを行うには、for-each文(拡張for文)を使う方法もある。この繰り返しでは、格納されている要素を順に変数に代入し、その都度ブロック内の処理を実行する。

```
for( 型 要素を格納する変数 : ArrayListのオブジェクト){
   要素を格納する変数を使って要素を参照
}
```


ArrayListクラス

Sample11.java

```
package lesson;
import java.util.ArrayList;
public class Sample11 {
   public static void main(String[] args){
      ArrayList<String> colors = new ArrayList<>();
      colors.add("red");
      colors.add("green");
      colors.add("blue");
      for(String c : colors ){
         System.out.println(c);
      }
   }
}
```

実行結果

```
red
green
blue
```