内容

第１章　Java とは

●ソースファイル

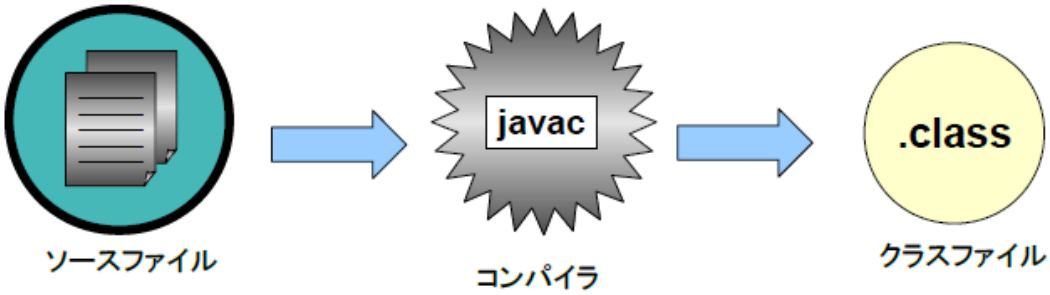
- 拡張子は「.java」
- エディターを使って記述する
-通常のテキストエディターでも編集可能

● コンパイル

- javac というコマンド（コンパイラ）を使う
- コンパイラが実行される（コンパイル）とソースファイルの内容をJavaVMで実行することのできる「クラスファイル」が作成される

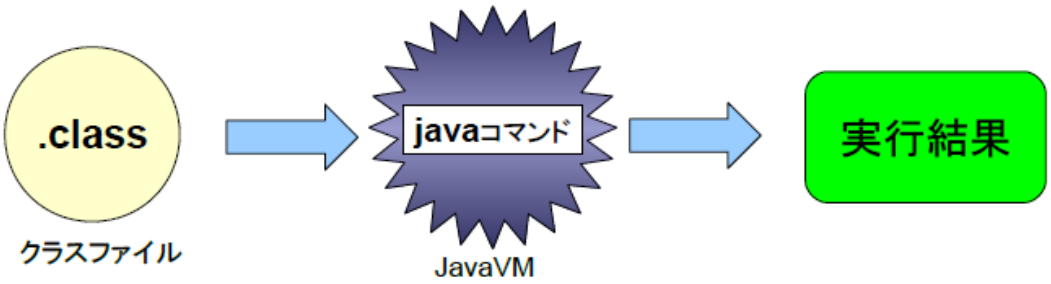
-拡張子は「.class」

-　クラスファイルはOSネイティブなバイナリではないので、JavaVM上でしか実行できない



● 実行

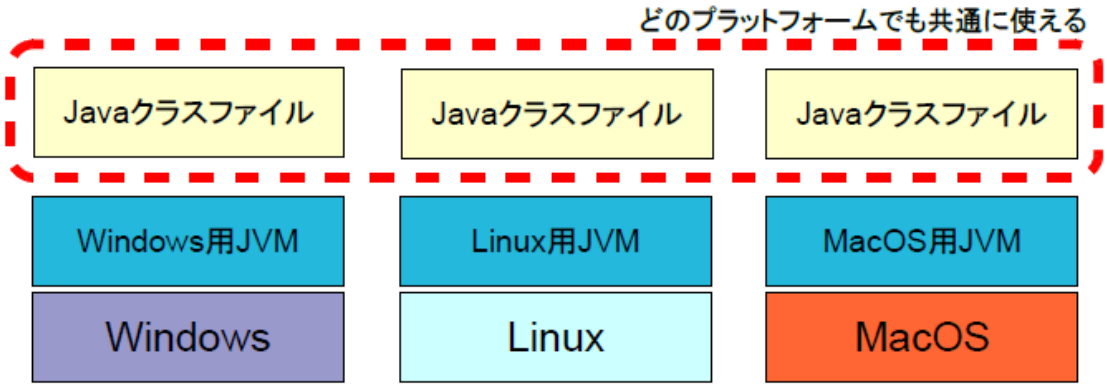
- java というコマンド（インタプリタ）を使う
-　クラスファイルを指定すると、そのファイルをJavaVM上で実行してくれる



Javaのバイナリ形式をそのまま解釈して実行できる仮想的な機械（Java Virtual Machine）を、プラットフォームごとに提供

各OSでは専用のJavaVMを使うことにより共通な実行結果を得ることができる

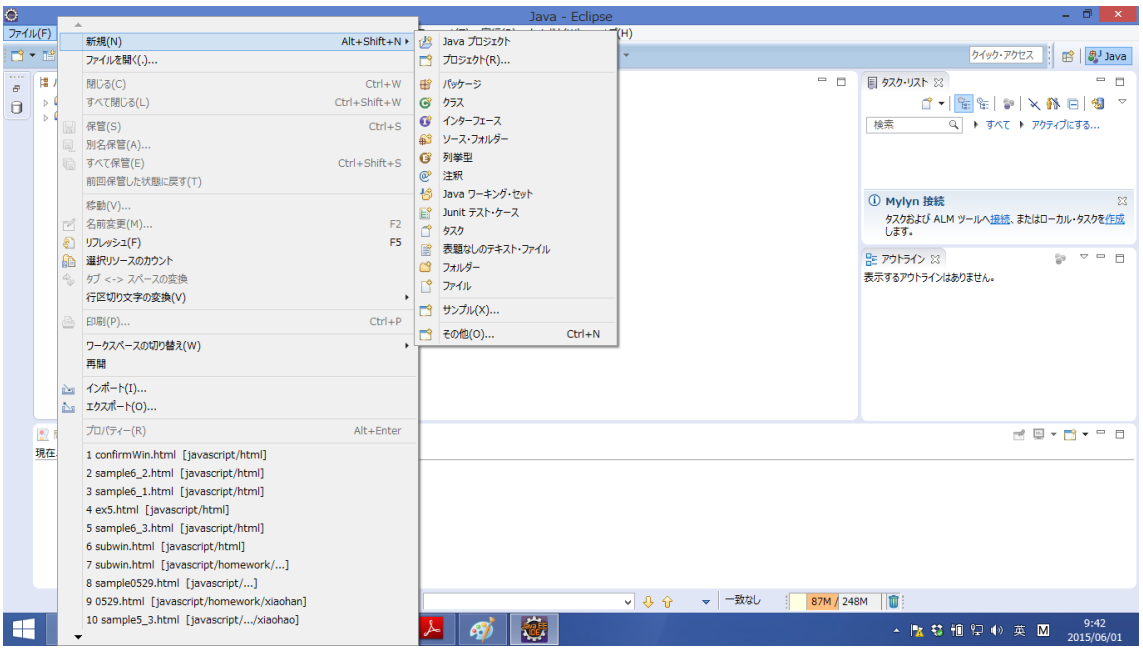
Windows ,Linux ,MacOS。



第２章　Eclipse で Java プログラミング

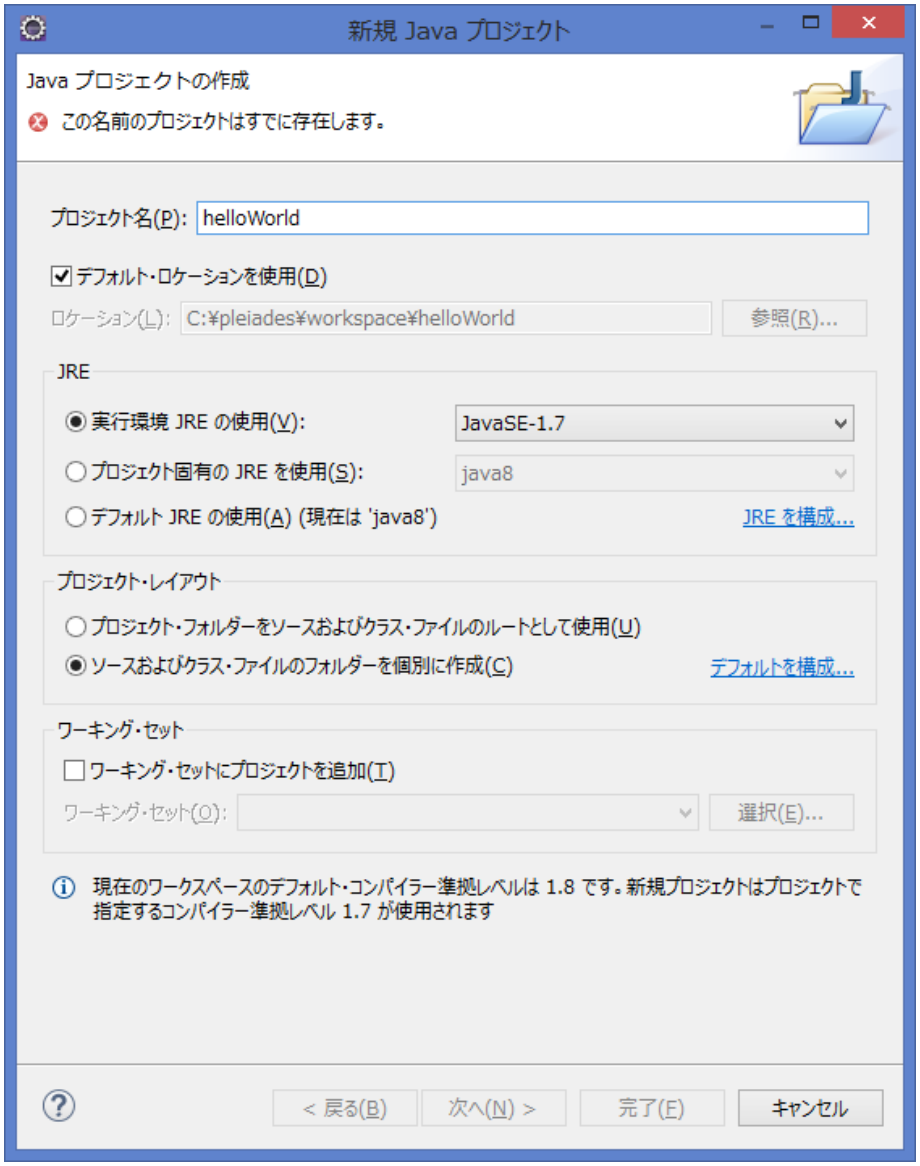
最も簡単な Java のプログラムをコンパイルするところまで作業を行ってみて、実際に動作するかを確認してみましょう。

今回は「Sample」という名前でプロジェクトを作成します。メニューの［ファイル］→［新規］→［プロジェクト］を指定します。



［新規プロジェクト］という名前のダイアログが開きます。これはプロジェクトを作成するためのウィザードです。まず、左のウィンドウから［Java］を選択し、次に右のウィンド

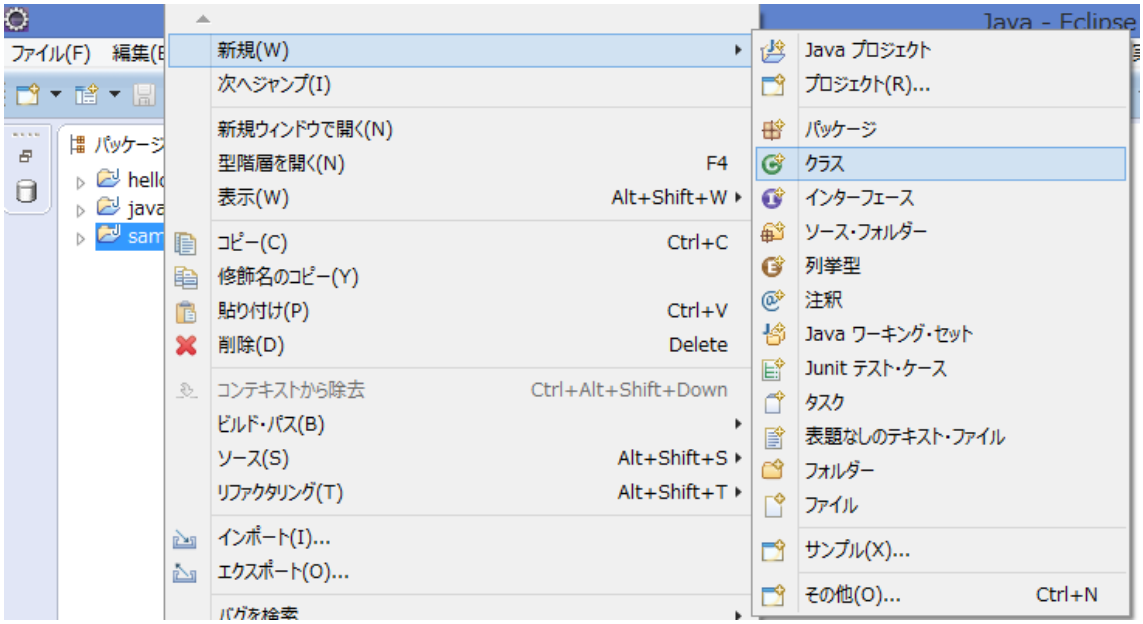
ウから[Java プロジェクト]を選択します。そして、[次へ]ボタンをクリックします。



［新規 Java プロジェクト］というダイアログが開きます。［プロジェクト名］に「Sample」と入力しましょう。最後に[完了]ボタンをクリックします。

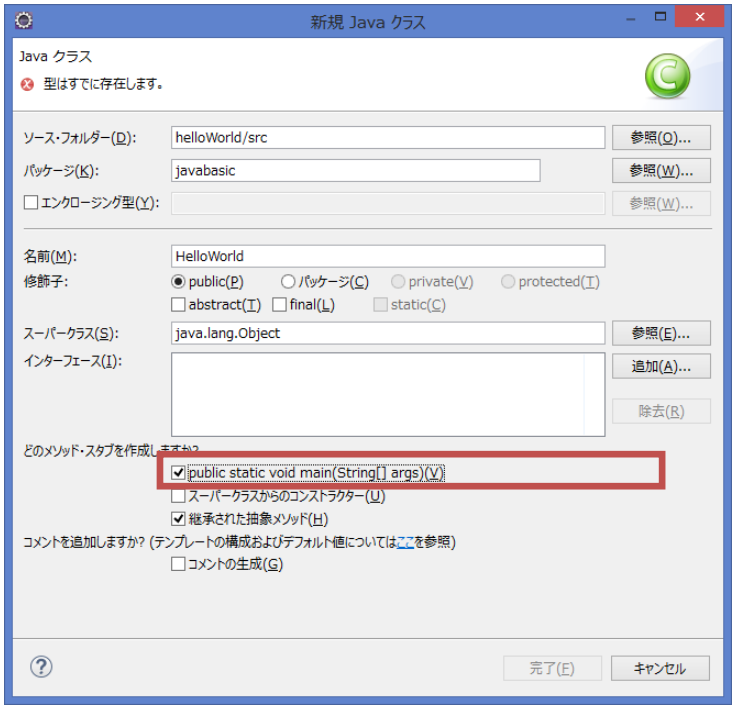
プログラムの作成

［パッケージ・エクスプローラー］の［Sample］をマウス右ボタンでクリックしてポップアップメニューを表示します。そのメニューで［新規］→［クラス］を指定します。

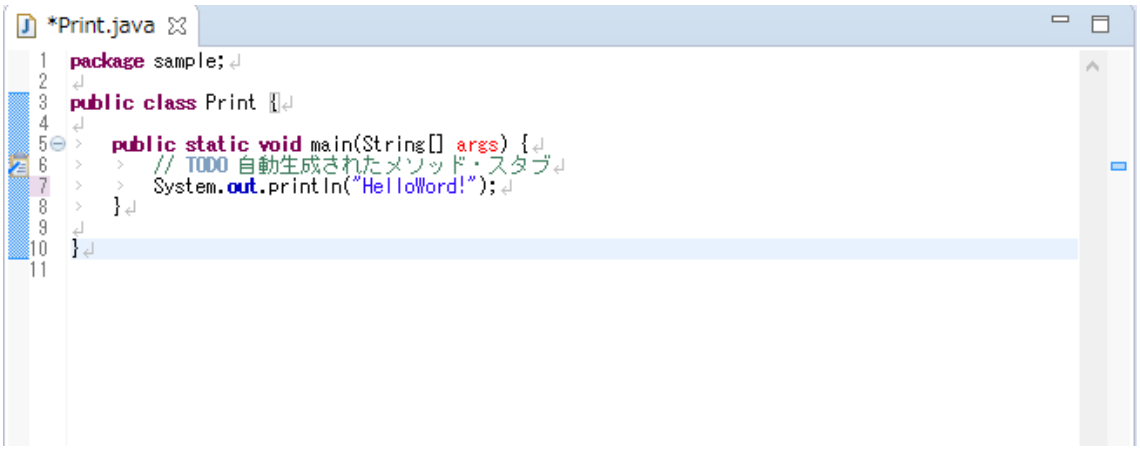


すると、［新規 Java クラス］ダイアログが表示されるので、［名前］に HelloWorld と入力し、［どのメソッド・スタブを作成しますか？］のところにある［public static void main(String[] args)］をチェックして、［終了］ボタンをクリックします。

すると、Print クラスのソースファイルである HelloWorld.java ファイルが自動的に出来上がり、エディタに以下のソースコードが表示されます。

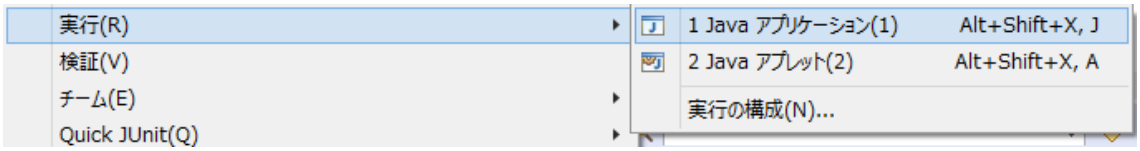


エディタにソースコードが自動生成される



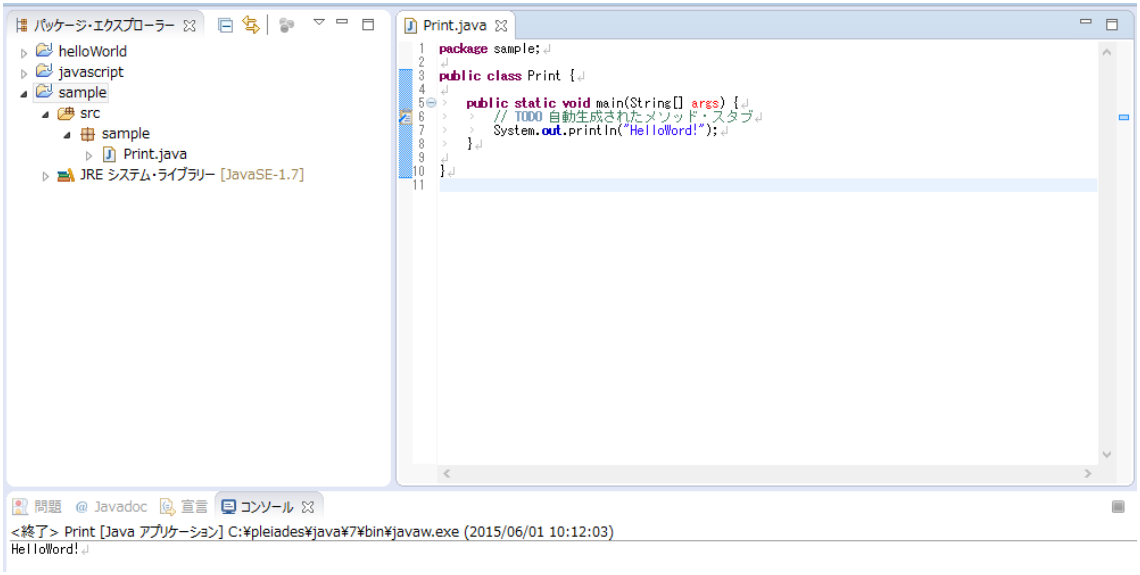
編集を終えたら、保存します。

プログラムを実行するためには、Print.java をエディタに表示したまま、Eclipse のメニューから［実行］→［次を実行］→［Java アプリケーション］を指定します。



ファイルを保存していないので、保存するファイルを確認するダイアログが表示されます。

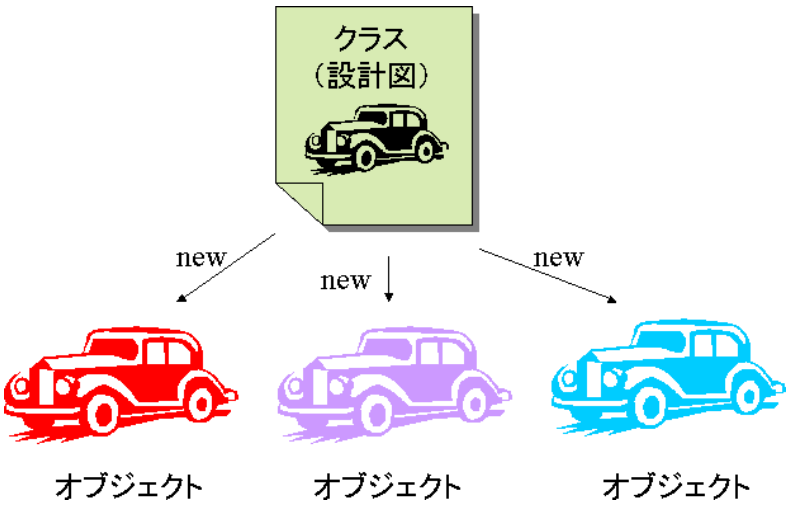
ここでは、そのまま[OK]ボタンをクリックしてください。そうすると、Eclipse の画面の下の部分に[コンソール]が表示されて、そこへ実行結果が出力されます。“HelloWorld!”が出力されていれば成功です。



第２章　クラスとパッケージ

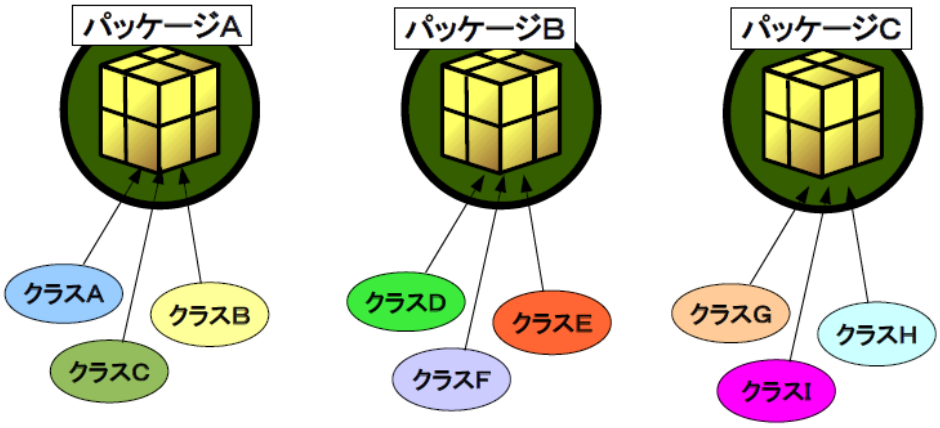
クラスとは

クラスとは，このように同じ性質をもつモノの設計図に相当するものです。あるクラスの実体(モノのこと)のことをオブジェクト(あるいはインスタンス)と呼びます．一つのクラス（設計図）から複数のオブジェクト（モノ）を作ることができます．



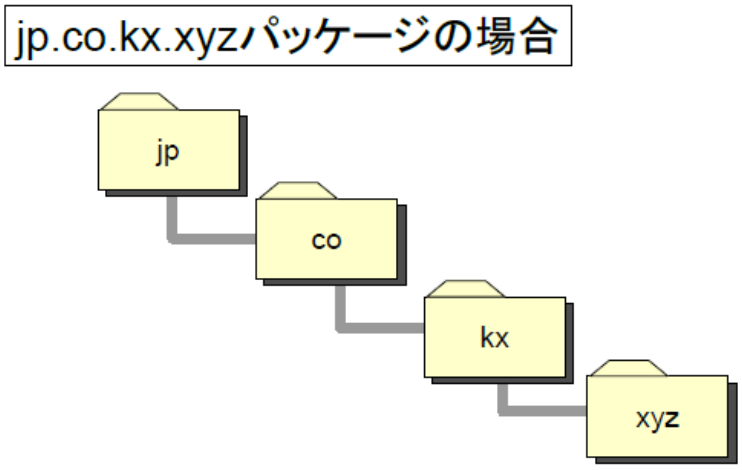
パッケージ

- クラスは「パッケージ」という単位で分けて管理することができる
- 意味や役割などのまとまりごとにパッケージで分類できる



パッケージの格納場所

- パッケージ名は同名のフォルダ内に格納される
- 「.」で区切るとフォルダ階層が作られる



パッケージを定義する

- あるクラスをあるパッケージに含めるには、クラスの先頭に「package宣言」を記述する

● 「パッケージ」だけを別個に作ることはない

● 1クラスにpackage宣言はひとつだけ

凡例

```
package パッケージ名;
class クラス名 {
…
}
```


適当なパッケージ名のクラスを新たに作成してみましょう。
Eclipse上で、パッケージに属するクラスはどのように表現されるでしょうか。
また、パッケージ構成を反映したフォルダが作られるかどうか確認してみましょう。

第３章 Java の変数について

3.1 変数の宣言

変数の初期化

データ型　変数名；

変数名　＝ 初期値；

```
int bookPrice;
```

変数の宣言と初期化は同時に行うことが多いです。

データ型　数名　＝ 初期値；

```
int bookPrice  = 0;
```

Java の基本型(一部)

種別	型	大きさ	値の範囲
整数	int	4 バイト	-2147483648～2147483647
倍長整数	long	8 バイト	-9223372036854775808 ～ 9223372036854775807
実数	float	4 バイト	1.4e-45(最小) ～ 3.4028235e38(最大) (それぞれ 1.4×10 の-45 乗, 3.4028235×10 の 38 乗という意味)
倍長実数	double	8 バイト	4.9e-324(最小) ～ 1.7976931348623157e308(最大)
ブーリアン	boolean		true または false
文字	char	2 バイト	文字データ

// 変数の例

Hensuu.java

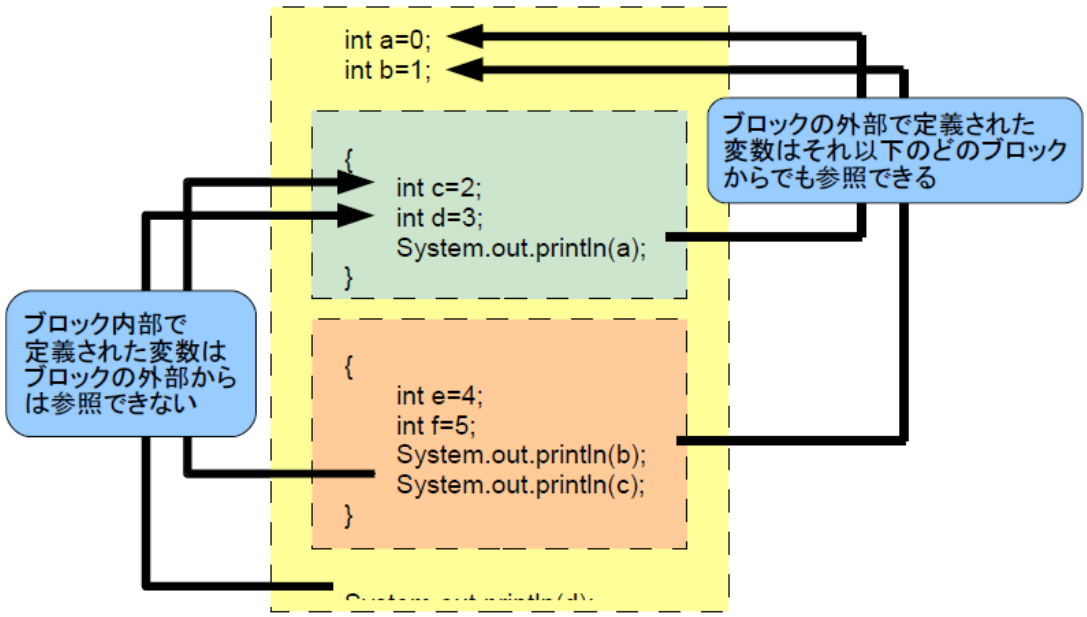
```
package javabasic;

public class Hensuu {

    public static void main(String[] args) {
        int a;
        int b;
        a = 10;
        b = 20;
        int result = a + b;
        System.out.println("result=" + result);
    }

}
```

3.2 変数の有効範囲

Javaコードにはたびたび｛｝が登場するが・・・
● ｛ ・・・ ｝を「ブロック」と呼ぶ
● 変数の有効範囲（スコープ）は、ブロックによって決まる
- 変数のスコープは、ブロックの内側である
- ブロックの内側に定義した変数は、外側では無効

ScopeExample.java

```
package javabasic;
public class ScopeExample {
        public static void main(String args[]) {
                int a = 0;
                int b = 1;
                {
                        int c = 2;
                        int d = 3;
                        System.out.println("a: " + a);
                }
                {
                        int e = 4;
                        int f = 5;
                        System.out.println("b: " + b);
                        System.out.println("c: " + c);
                }

        }
}
```



3.3 変数のキャスト（型変換）

変数には型があることを勉強しました。型が異なる変数同士の代入などできません。例えば、以下はエラーになります。


```
int integer = 3.1415926;
```

キャストは以下のように行います。

(キャストしたい目的のデータ型)変数名；

```
double db1 = 3.1415926;
int integer = (int)db1;
```

第４章　条件分岐(if 文)

```
処理 1;
if (条件) {
    処理 2;
    処理 3;
} else {
    処理 4;
    処理 5;
}
処理 6;
```

演算子	意味	例
式 1 == 式 2	等しい	a == 0
式 1 != 式 2	等しくない	a != 1
式 1 < 式 2	小さい	a < 10
式 1 > 式 2	大きい	10 > b
式 1 <= 式 2	小さいか等しい	a <= b
式 1 >= 式 2	大きいか等しい	a >= 3.14
!真偽式	否定(	!(a == 1)
真偽式 1 && 真偽式 2	かつ	(1 <= a) && (a <= 2)
真偽式 1 || 真偽式 2	または	(a <= 1) || (b == 2)

// 0 か正か負かを判定

```
package javabasic;
public class ZeroPlusMinus1 {
        public static void main(String[] args) {
                int number = 10;
                if (number == 0) {
```

```
                        System.out.println("0です");
                } else {
                        if (number > 0) {
                                System.out.println("正です");
                        } else {
                                System.out.println("負です");
                        }

                }
        }
}
```

if（条件式1）文1 else **if（条件式2）文2**

if（条件式1）文1 else **if（条件式2）文2 else 文3**

// 0か正か負かを判定（別バージョン）

```
package javabasic;
public class ZeroPlusMinus1 {
        public static void main(String[] args) {
                int number = 10;
                if (number == 0) {
                        System.out.println("0です");
                } else {
                        if (number > 0) {
                                System.out.println("正です");
                        } else {
                                System.out.println("負です");
                        }

                }
        }
}
```

練習：JSample1.java

第５章　場合分けが多いとき（switch文）

```
switch (式) {
  case 値1:
    文1
    break;
  case 値2:
    文2
    break;
  default:
    文3
}
```


CaseSamle.java

```
package javabasic;
public class CaseSample {
	public static void main(String[] args) {
		int num = 3;
		switch (num){
		  case 1:
		    System.out.println("非常に不満");
		    break;
		  case 2:
		    System.out.println("少し不満");
		    break;
		  case 3:
		    System.out.println("どちらとも言えない");
		    break;
		  case 4:
		    System.out.println("少し満足");
		    break;
		  case 5:
		    System.out.println("大変満足");
		    break;
		}
	}
```

第６章　繰り返し(for)

```
for (int i = 0 ; i < 10 ; i++) {
	// i はこのブロックの中だけで有効
}
```

```
sum += 1;
sum += 2;
sum += 3;
sum += 4;
sum += 5;
sum += 6;
sum += 7;
sum += 8;
sum += 9;
sum += 10;
```

// この時点で変数 i は消滅している

JSample2.java

```
package javabasic;
public class JSample2 {
        public static void main(String[] args) {
            int sum = 0;
            for (int i = 1 ; i <= 1000 ; i++){
              sum += i;
            }
            System.out.println("合計は" + sum + "です");
        }
}
```

JSample3.java

```
package javabasic;
public class JSample3 {
    public static void main(String[] args) {
    int i;
    for (i = 3; i <= 20; i += 3){
      System.out.println(i);
    }
    System.out.println("変数の値が" + i + "で条件式がfalseとなりました");
  }
}
```

複数の初期化式と変化式

for 文では初期化式、条件式、変化式の 3 つの式を使用しますが、初期化式と変化式には複数の式を記述することができます。複数の式を記述する場合は式と式の間をカンマ(,)で区切って記述します。

for（初期化式 1，初期化式 2，..；条件式；変化式 1，変化式 2，..）{

**　実行する文；**

}

※条件式は関係演算子と論理演算子で複雑な条件式を記述できます。

```
package javabasic;
public class JSample4 {
        public static void main(String[] args) {
                int i, j;
                for (i = 1, j = 5; i < 6; i++, j--) {
                        System.out.println(i + " + " + j + " = " + (i + j));
```

```
                }
        }
}
```

for 文の中に for 文を記述

for 文で実行されるブロックの中には様々な文を記述することができますので他の for 文を記述することができます。次の例を見て下さい。

```
for (int i = 0; i < 2; i++){
  for (int j = 0; j < 2; j++){
    System.out.println("i = " + i + ",j = " + j);
  }
}
```

i*j

練習：JSample6.java

第７章　繰り返し(while 文)

繰り返し回数が分からない場合は、 ある一定の条件を指定し、その条件を**満たしている間**繰り返すという方法をとります。

処理 1;
while (条件) {
**   処理 2;**
**   処理 3;**
}
処理 4;

JSample7.java

```
int i = 0;
    while (i < 5){
      System.out.println("i = " + i);
      i++;
    }
```

```
while (i <= 9){
while (j <= 9){
```

練習：JSample8.java

繰り返し(do..while)文

do{

処理 1;

処理 2

}while (条件式);

JSample9.java

```
do{
	System.out.println("i = " + i);
	i++;
}while (i < 2);
```

練習：JSample10.java

ラベル付き break 文

break 文が実行されると break 文を含む一番内側のブロックを抜けますが、ラベル付きの break 文にすることで任意のブロックを抜けることができます。ラベル付き break 文の書式は次の通りです。

break ラベル;

JSample11.java

```
package javabasic;
public class JSample11 {
	public static void main(String[] args) {
		outside: for (int i = 1; i < 5; i++) {
			for (int j = 1; j < 5; j++) {
				if (i * j > 10) {
					break outside;
				}
				System.out.println(i + "*" + j + "=" + (i * j));
			}
		}
	}
}
```


第８章　配列

型名 配列変数名[];

配列変数名 = new 型名[要素数];

型名 配列変数名[] = new 型名[要素数];

```
package javabasic;

public class JSample13 {

	public static void main(String[] args) {

	    char[] moji;

	    moji = new char[2];

	    moji[0] = 'A';

	    moji[1] = '漢';

	    for (int i = 0; i < 2; i++){

	      System.out.println(moji[i]);

	    }

	}

}
```

参照型の変数

```
int n[] = new int[3];
int m[];
```

```
m = n;
```

JSample14.java

```
package javabasic;
public class JSample14 {
	public static void main(String[] args) {
	    int n[];
	    int m[];
	    n = new int[2];
	    System.out.println(n);
	    n[0] = 10;
	    m = n;
	    System.out.println(m);
	    System.out.println("n[0] = " + n[0]);
	    System.out.println("m[0] = " + m[0]);
	}
}
```

配列の長さ

JSample16.java

```
package javabasic;

public class JSample16 {

	public static void main(String[] args) {

	    int n[] = {18, 29, 36, 12};

	    for (int i = 0; i < n.length; i++){

	      System.out.println(n[i]);

	    }

	}

}
```

多次元配列

```
型名 配列変数名[][];
型名 配列変数名[][] = new 型名[要素数][ 要素数];
```

```
package javabasic;

public class JSample17 {

	public static void main(String[] args) {

	    int seiseki[][] = new int[2][3];

	    seiseki[0][0] = 80;

	    seiseki[0][1] = 92;

	    seiseki[0][2] = 45;

	    seiseki[1][0] = 75;

	    seiseki[1][1] = 89;

	    seiseki[1][2] = 54;

	    String kyoka[] = {"国語", "数学"};

	    for (int i = 0; i < 2; i++){

	      System.out.println(kyoka[i] + "の成績");

	      for (int j = 0; j < 3; j++){

	        System.out.println(seiseki[i][j]);

	      }

	    }

	}

}
```

多次元配列の初期化

```
型名 配列変数名 1[] = {値 1, 値 2, ..};
```

型名 配列変数名 2[] = {値 1, 値 2, ..};

型名 配列変数名[][] = {{値 1_1, 値 1_2, ..}, {値 2_1, 値 2_2, ..}};

```
int num[][] = {
  {10, 8, 5},
  {9, 16, 4},
  {3, 7, 5}
};
```

JSample18.java

```
package javabasic;

public class JSample18 {

        public static void main(String args[]){

            int seiseki[][] = {

              {68, 82, 92},

              {76, 33, 83},

              {92, 45, 38}

            };

            String kurasu[] = {"Aクラス", "Bクラス", "Cクラス"};

            for (int i = 0; i < 3; i++){

              System.out.println(kurasu[i] + "の成績");

              for (int j = 0; j < 3; j++){

                System.out.println(seiseki[i][j]);

              }

            }

          }

}
```

多次元配列の長さ

配列変数名[インデックス].length

JSample19.java

```
package javabasic;

public class JSample19 {

        public static void main(String args[]) {

                int num[][] = { { 1, 3, 5 }, { 2, 4, 6 } };
```

```
                System.out.print("多次元配列の長さは");
                System.out.println(num.length);
                System.out.print("インデックス0の要素の長さは");
                System.out.println(num[0].length);
                System.out.print("インデックス1の要素の長さは");
                System.out.println(num[1].length);
        }
}
```

第９章　String クラスを使った文字列処理

```
String str;
str = "文字列";
```

文字列の処理

```
String str;
str = "abc";
str = str + "def";
```

JSample21.java

```
package javabasic;

public class JSample21 {

        public static void main(String[] args) {
            String str = "こんにちは。";
            str = str + "加藤さん。";
            System.out.println(str);
        }
}
```

文字列と文字列の比較

文字列と文字列を比較する場合、「==」演算子を使うと同じ文字列でも等しいと判定されたり等しくないと判定されたりします。どういった時にどう判定されるのかは Java の実装にも依存してしまいますのでオブジェクトの比較ではなく同じ文字列が格納さえているかどうか調べる時には「==」演算子は使わないで下さい。

equals メソッド

オブジェクトが同一なのかを調べるのではなく、オブジェクトに格納されている文字列が単に同じかどうか調べるには String クラスで用意されている equals メソッドを使います。実際には次のように記述します。

JSample22.java

```
package javabasic;
public class JSample22 {
	public static void main(String[] args) {
		String str1 = new String("abc");
		String str2 = new String("abc");

		String str3 = "ab";
		System.out.println("str1 = " + str1);
		System.out.println("str2 = " + str2);


		if (str1 == str2){
		  System.out.println("等しい");
		}else{
		  System.out.println("等しくない");
		}
		str3 = str3 + "c";
		System.out.println("str1 = " + str1);
		System.out.println("str3 = " + str3);
		if (str1 == str3){
		  System.out.println("等しい");
		}else{
		  System.out.println("等しくない");
		}
		System.out.println("equalsメソッドで比較");
		if (str1.equals(str3)){
		  System.out.println("等しい");
		}else{
		  System.out.println("等しくない");
		}
	}
}
```

Stringクラスのメソッド

指定の位置の文字を取得
文字列.charAt(インデックス)
JSample23.java

```
package javabasic;

public class JSample23 {

	public static void main(String[] args) {

		String str = "腕時計";

		   char c1 = str.charAt(0);

		   char c2 = str.charAt(1);

		   char c3 = str.charAt(2);

		   System.out.println(c1);

		   System.out.println(c2);

		   System.out.println(c3);
```



```
        }
}
```

文字列の長さを取得

文字列.length()

```
package javabasic;
public class JSample24 {
        public static void main(String[] args) {
                String str = "腕時計";
                System.out.println("文字列の長さは" + str.length());
                for (int i = 0; i < str.length(); i++){
                  char c = str.charAt(i);
                  System.out.println(c);
                }
        }
}
```

文字列の一部を取得(substring)

元の文字列から一部分を取り出して新しい文字列を作成します。

```
substring
public String substring(int beginIndex, int endIndex)
```

```
package javabasic;
public class JSample26 {
        public static void main(String[] args) {
                String str1 = new String("Hello World!");
                String new_str1 = str1.substring(2, 5);
                System.out.println(str1 + "のsubstring(2,5)は" + new_str1 + "です");
                String str2 = new String("こんにちは");
                String new_str2 = str2.substring(2, 4);
                System.out.println(str2 + "のsubstring(2,4)は" + new_str2 + "です");
        }
}
```

toUpperCase メソッド ：

文字列. toUpperCase()

toLowerCase メソッド ：

文字列. toLowerCase ()

JSample27.java

```
package javabasic;
public class JSample27 {
        public static void main(String[] args) {
```

```
                String str1 = new String("Good Morning");
                String upper_str1 = str1.toUpperCase();
                String lower_str1 = str1.toLowerCase();
                System.out.println("元の文字列 : " + str1);
                System.out.println("大文字へ変換 : " + upper_str1);
                System.out.println("小文字へ変換 : " + lower_str1);
                String str2 = new String("現在 A.M. 3時20分です");
                String upper_str2 = str2.toUpperCase();
                String lower_str2 = str2.toLowerCase();
                System.out.println("元の文字列 : " + str2);
                System.out.println("大文字へ変換 : " + upper_str2);
                System.out.println("小文字へ変換 : " + lower_str2);
        }
}
```

先頭又は最後の空白を取り除く(trim)

文字列に先頭又は最後に空白文字がくっ付いている場合、それらを全て取り除きます。

文字列.trim();

```
package javabasic;
public class JSample28 {
        public static void main(String[] args) {
            String str1 = new String("  Good Morning    ");
            System.out.println("元の文字列「" + str1 + "」");
            System.out.println("文字数は" + str1.length() + "です");
            String new_str1 = str1.trim();
            System.out.println("空白を取り除いた文字列「" + new_str1 + "」");
            System.out.println("文字数は" + new_str1.length() + "です");
        }
}
```

文字列を分割する(split)

配列 = 文字列.split(分割文字列)

任意の文字で、指定文字列を分割して配列に格納します。

```
String str = "りんご,レモン,すいか,ぶどう";
```

```
String[] fruit = str.split(",", 0);
```

```
package javabasic;
public class JSample29 {
	public static void main(String[] args) {
		String str = "りんご,レモン,すいか,ぶどう";
		String[] fruit = str.split(",");

		for (int i = 0; i < fruit.length; i++) {
			System.out.println(i + "番目の要素 = :" + fruit[i]);
		}
	}

}
```

最初に該当した文字列を置換する(replaceFirst)

文字列. replaceFirst()

```
String str = "今日の天気は晴れでした。明日の天気は晴れです。";
str = str.replaceFirst("晴れ", "雨");
```

JSample30.java

```
package javabasic;
public class JSample30 {
	public static void main(String[] args) {
		String str = "今日の天気は晴れでした。明日の天気は晴れです。";
	    System.out.println("変換前:");
	    System.out.println(str);
	    str = str.replaceFirst("晴れ", "雨");
	    System.out.println("変換後:");
	    System.out.println(str);
	}
}
```

該当する全ての文字列を置換する(replaceAll)

文字列. replaceAll()

```
String str = "今日の天気は晴れでした。明日の天気は晴れです。";
str = str.replaceAll("晴れ", "雨");
```

```
package javabasic;

public class JSample30 {

	public static void main(String[] args) {

	    String str = "今日の天気は晴れでした。明日の天気は晴れです。";

	    System.out.println("変換前:");

	    System.out.println(str);

	    str = str.replaceAll("晴れ", "雨");

	    System.out.println("変換後:");

	    System.out.println(str);

	}

}
```

第１０章　メソッドの使い方

メソッドは複数の処理をまとめたものです。プログラムの中からメソッドを呼び出すと、そのメソッドで記述された処理が実行されます。

メソッドの定義

```
[修飾子] 戻り値のデータ型 メソッド名(引数 1, 引数 2, ....){

}
```

何も値を返さない場合は void 型を指定することになっていますので、取り合えず「void」を記述して下さい。

メソッドの呼び出し
変数 = メソッド名(値 1, 値 2, ...);

JSample32.java

```
package javabasic;
public class JSample32 {
	public static void main(String[] args) {
		hello();
		bye();
		hellobye();
	}
	private static void hello() {
		System.out.println("こんにちは。");
	}
	private static void bye() {
		System.out.println("さようなら。");
	}
	private static void hellobye() {
		hello();
		bye();
	}
```

```
}
```

引数を使ってメソッドに値を渡す

メソッドを呼び出す時に呼び出し元から値をメソッドに渡すことができます。まずどのような場合に使うのかを確認してみます。

```
package javabasic;
public class JSample33 {
        public static void main(String[] args) {
            int eigo = 78;
            int suugaku = 90;
            int kokugo = 68;
            check("英語", eigo);
            check("数学", suugaku);
            check("国語", kokugo);
          }
          private static void check(String kyoka, int seiseki){
            System.out.print(kyoka + "の試験結果は");
            if (seiseki > 80){
              System.out.println("合格です");
            }else{
              System.out.println("不合格です");
            }
        }
}
```

JSample33_2.java

引数に配列やクラスオブジェクトを渡す

メソッドに渡す値が基本データ型であった場合には、値がコピーされて値そのものが引数に代入されます。考え方としては基本データ型の値を他の変数に代入する場合と同じです。

```
package javabasic;
public class JSample34 {
        public static void main(String[] args) {
            int num = 8;
            int array[] = {10, 4};
            System.out.println("num = " + num);
            System.out.println("array[0] = " + array[0]);
            henkou(num, array);
            System.out.println("num = " + num);
            System.out.println("array[0] = " + array[0]);
          }
          private static void henkou(int num, int array[]){
            num = 5;
            array[0] = 12;
          }
}
```

戻り値を使ってメソッドから値を返す


```
package javabasic;
public class JSample35 {
	public static void main(String[] args) {
		int kekka;
		kekka = bai(9);
		System.out.println(kekka);
		kekka = bai(5);
		System.out.println(kekka);
	}
	private static int bai(int n) {
		return n * 2;
	}
}
```

JSample36.java

```
package javabasic;
public class JSample36 {
	public static void main(String[] args) {
	    int num;
	    String kekka;
	    num = 9;
	    kekka = hantei(num);
	    System.out.println(num + "は" + kekka);
	    num = 6;
	    kekka = hantei(num);
	    System.out.println(num + "は" + kekka);
	  }
	  private static String hantei(int n){
	    if (n % 2 == 0){
	      return "偶数";
	    }else{
	      return "奇数";
	    }
	  }
	}
```

return文

メソッドから呼び出し元に値を返す場合には、return文を使って戻り値を指定しますが、return文は戻り値を返すためだけに使用するわけではありません。return文がメソッドの中で実行されるとそれ以降の処理を実行せず呼び出し元へ処理が移すことができます。

JSample37.java

```
package javabasic;
public class JSample37 {
	public static void main(String[] args) {
		int data[];
		test(15, 4);
		test(7, 0);
	}
	private static void test(int n1, int n2) {
		if (n2 == 0) {
			System.out.println("0で割ることはできません");
			return;
```

```
                }
                System.out.println(n1 + " / " + n2 + " = " + (n1 / n2));
                return;
        }
}
```

メソッドのオーバーロード

メソッドを引数を付けて呼び出す時、引数に記述する値のデータ型はメソッドで決められたものしか指定できません。その為、同じような機能を持つメソッドであっても引数のデータ型が異なれば別々のメソッドを用意する必要があります。

```
public static void main(String args[]){
  int n = plus(10, 7);
  System.out.println(n);
  double d = plus(3.14, 7.4);
  System.out.println(d);
}
private static int plus(int n1, int n2){
  return n1 + n2;
}
private static double plus(double d1, double d2){
  return d1 + d2;
}
```

オーバーロードができる場合

同じメソッド名で複数のメソッドを定義するには、引数の数が異なっているか、引数のデータ型が異なっている必要があります。

○ 引数の数が異なっている場合:

```
private static void test(int n1){
  // ...
}
private static void test(int n1, int n2){
  // ...
}
```

○ 引数の種類が異なっている場合:

```
private static void test(int n1){
  // ...
}
private static void test(double d1){
  // ...
}
```

○ 引数のデータ型と数は同じでも順番が異なっている場合:

```
private static void test(int n1, double d1){
  // ...
}
private static void test(double d1, int n1){
  // ...
}
```

このいずれかの条件さえ満たしていれば、同じメソッド名であっても 2 つまたそれ以上のメソッドを同じクラス内に定義することができます。

オーバーロードができない場合

次にオーバーロードができない場合も確認しておきます。

メソッド定義の時の引数の変数名だけが異なるようなものはオーバーロードできません。また戻り値の有る無しや戻り値のデータ型異なっているだけのものもオーバーロードすることはできません。

× 引数の変数名だけが異なっている場合:

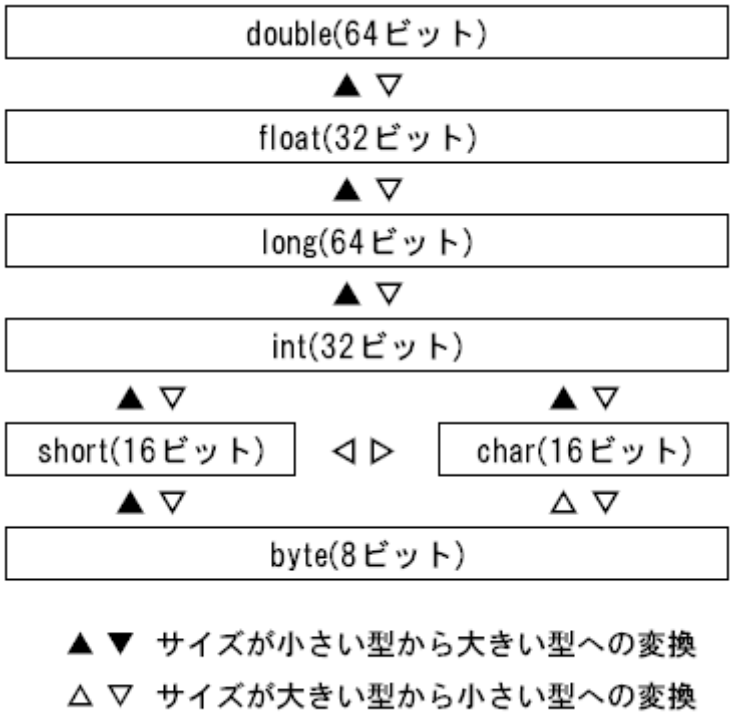
```
private static void test(int a){
  // ...
}

private static void test(int b){
  // ...
}
```

× 戻り値だけが異なっている場合:

```
private static void test(int n){
  // ...
}
private static int test(int n){
  // return n;
}
```

このように変数名だけが異なっている場合や戻り値だけが異なっているメソッドを複数記述した場合は、コンパイルエラーとなり「(メソッド名) は (クラス名) で定義されています。」のようなエラーメッセージが表示されます。

第１１章　型変換の基本ルール



サイズが小さい型から大きい型への変換

short 型から int 型への変換や、int 型から long 型への変換などサイズが小さい型から大きい型への変換の場合、より大きな数値が格納できるところへ変換するわけですから特に意識することなく変換することができます。

```
short s = 10;
int i = s;
long l = i;
```

サイズが大きい型から小さい型への変換

int 型から short 型への変換や、long 型から int 型への変換などサイズが大きい型から小さい型への変換の場合、キャスト演算子を使って明示的に型が変更されることを示す必要があります。

```
int i = 10;
short s = (short)i;
```


では簡単な例で試しておきます。

```
JSample3_1.java
class JSample3_1{
   public static void main(String args[]){
      int i = 7;

      float f1 = (float)i / 3;
      float f2 = (float)(i / 3);

      System.out.println("f1 = " + f1);
      System.out.println("f2 = " + f2);
   }
}
```

第１２章　オブジェクトの作成と値の設定

クラスとは

```
class クラス名{
....
}
```

イメージ的には下記のような感じとなります。

```
設計図 車{
   車を動かす{
      ....
```



```
  }

  車を止める{
      ....
  }

  ライトを付ける{
     ....
  }
}
```

クラスと言うのはこのような感じのものです。

クラスからオブジェクトを作る

クラスを元に実際に作った物をオブジェクトと呼んでいます。クラスから物体を作成するには"new"を使います。

```
クラス名  変数名  = new  クラス名(引数);
```

例えば"Televison"と言うクラスがあった場合に、そのクラスからオブジェクトを作成するには下記のような記述となります。(引数が無い場合)。

```
Televison tv = new Television();
```

又は次のようにも書く事ができます。

```
Televison tv;
tv = new Television();
```

クラスを利用してみる

```
class Television{
  int channelNo;
  String place;

  void setChannel(int newChannelNo){
    channelNo = newChannelNo;
  }

  void setPlace(String newPlace){
    place = newPlace;
  }

  void dispChannel(){
    System.out.println(place + "にあるテレビの現在のチャンネルは " +  channelNo + "
です");
  }
```



```
}
```

ctest1. java

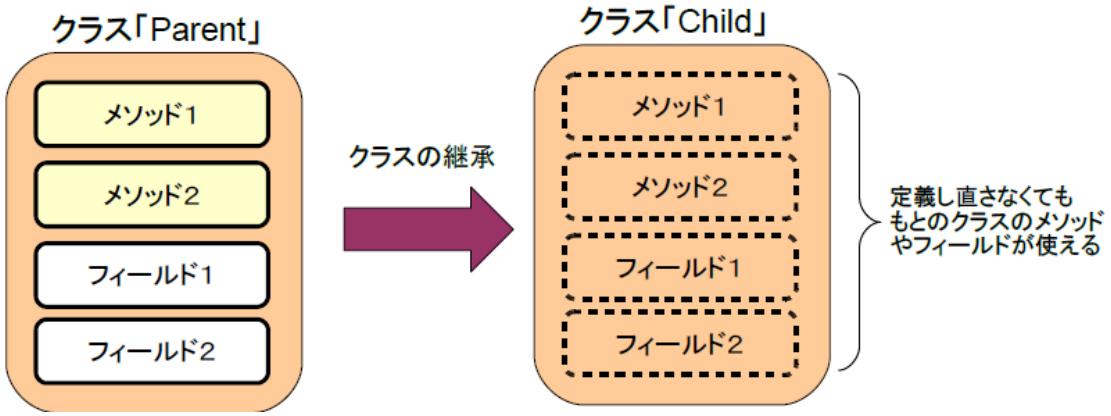
```
class ctest1{

	public static void main(String args[]){
		Television tv1 = new Television();
		Television tv2 = new Television();

		tv1.setPlace("居間");
		tv2.setPlace("寝室");

		tv1.setChannel(1);
		tv2.setChannel(8);

		tv1.dispChannel();
		tv2.dispChannel();
	}
}

class Television{
	int channelNo;
	String place;

	void setChannel(int newChannelNo){
		channelNo = newChannelNo;
	}

	void setPlace(String newPlace){
		place = newPlace;
	}

	void dispChannel(){
		System.out.println(place + "にあるテレビの現在のチャンネルは " +
channelNo+ " です");
	}
}
```

クラスの継承

Java における継承

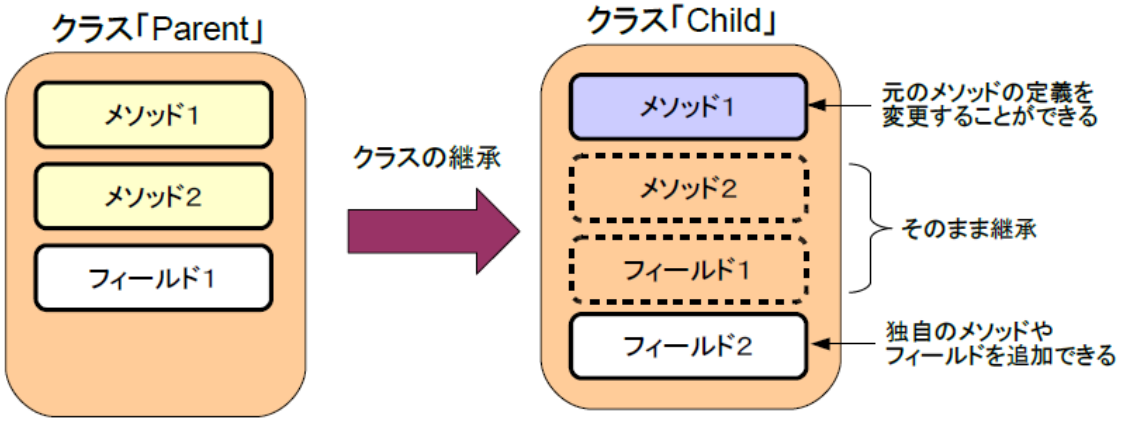
- あるクラスの定義内容を受け継ぐこと

● メソッド・フィールドを定義し直さなくてもそのまま使える



● Java における継承

- 継承したクラスで独自のメソッドやフィールドを追加することもできる
（＝拡張）
- 継承したクラスで元のメソッド・フィールドの定義を変更することもできる
（＝再定義・オーバーライド）



● **継承**

- extendsキーワードによって継承を表現
- 子クラスでは、特に必要なければ親クラスのフィールド・メソッドを
定義し直さなくて良い（何も書かなくて良い）
- クラス定義には親クラスはひとつしか指定できない（単一継承）

```
class 子クラス名 extends 親クラス名 {
}
```



```
package javabasic;

public class ColTV extends TV {

	String col;

	void setCol(String newCol){
		col = newCol;
	    }
	void dispChannel(){
		System.out.println(place + "にある" + col + "テレビの現在のチャンネルは " + channelNo+ " です");
		}
}
```

ctest2. java

```
package javabasic;

public class ctest2 {

	public static void main(String[] args) {

		ColTV tv3 = new ColTV();
		ColTV tv4 = new ColTV();

		tv3.setPlace("居間");
		tv4.setPlace("寝室");

		tv3.setChannel(1);
		tv4.setChannel(8);

		tv3.setCol("白い");
		tv4.setCol("黒い");

		tv3.dispChannel();
		tv4.dispChannel();

	}

}
```

修飾子について

修飾子とはメソッドを使用できる範囲を指定するものです。

- public と指定すると、そのメソッドはどこからでも使用できます。
- protected と指定すると、そのメソッドはそのクラスを継承したサブクラスからしか使用できません。
  クラスの継承については後の章で説明します。

- 修飾子を省略すると、そのメソッドは同じパッケージのクラスからしか使用できません。
  パッケージについては後の章で説明します。
- private と指定すると、そのメソッドはそのクラスの中からしか使用できません。

メンバ変数はローカル変数と違い、クラス内であればどこからでも参照することができます。また、メソッドのように修飾子をつけることで、変数を参照できる範囲を指定することもできます。
修飾子の意味はメソッドのときと同じです。

- public と指定すると、そのメンバ変数はどこからでも参照できます。
- protected と指定すると、そのメンバ変数はそのクラスを継承したサブクラスからしか参照できません。
- 修飾子を省略すると、そのメンバ変数は同じパッケージのクラスからしか参照できません。
- private と指定すると、そのメンバ変数はそのクラスの中からしか参照できません。

第１３章　例外処理

ここでは例外とは何かについて簡単に確認しておきます。Java でプログラムを作成した場合、コンパイルの時点でエラーが発生する場合と実行中にエラーが発生する場合があります。

try 文
例外処理を行うには try 文を使用します。基本的な書式は次の通りです。

```
try{
   例外が発生しているかどうか調べる文 1;
   例外が発生しているかどうか調べる文 2;
   ...
}
catch (例外クラス 1 変数名 1){
   例外クラス 1 の例外が発生した時に行う文;
   ...
}
catch (例外クラス 2 変数名 2){
   例外クラス 2 の例外が発生した時に行う文;
   ...
```



```
}
```

まずプログラムの中で例外が発生するかどうかを調べる対象の文を try の後の「{」から「}」までのブロック内に記述します。そして例外が発生した時に行いたい処理を catch の後の「{」から「}」までのブロック内に記述します。

```
int n[] = {18, 29, 36};
System.out.println("開始します");
try{
   for (int i = 0; i < 4; i++){
      System.out.println(n[i]);
   }
}
catch(ArrayIndexOutOfBoundsException e){
   System.out.println("配列の範囲を超えています");
}
System.out.println("終了しました");
```

必ず実行する処理の記述(try..catch..finally)

```
try{
   例外が発生しているかどうか調べる文 1;
   例外が発生しているかどうか調べる文 2;
   …
}
catch (例外クラス 1 変数名 1){
   例外クラス 1 の例外が発生した時に行う文;
   …
}
catch (例外クラス 2 変数名 2){
   例外クラス 2 の例外が発生した時に行う文;
   …
}
finally {
   例外が発生するしないに関わらず実行する文;
   …
}
```

finallyの後の「{」から「}」までのブロックに記述された処理は、try文の中で例外が発生してもしなくても必ず実行されます。その為、必ず行っておきたい処理がある場合にはfinallyブロックを用意してブロック内に記述するようにして下さい。

```
class JSample40{
  public static void main(String args[]){
    disp(1);
    disp(2);
    disp(3);
  }
```



```
  private static void disp(int no){
    int n[] = {18, 29, 36};

    try{
      System.out.println(n[no]);
    }
    catch(ArrayIndexOutOfBoundsException e){
      System.out.println("配列の範囲を超えています");
      return;
    }
    finally{
     System.out.println("要素の出力を終了します。");
    }
  }
}
```


第１４章　ファイルを管理する

File クラス

```
File オブジェクト名 = new File(ファイル名);
```

対象とするファイル名をディレクトリの位置＋ファイル名の形で指定します。例えば下記のようになります。

```
File file = new File("c:¥¥tmp¥¥test.txt");
```

下記のサンプルでは「C ドライブ」のルートディレクトリ(つまり C:¥のことです)に含まれるファイルやディレクトリ一覧を取得しています。(list メソッドの使い方は次のページ以降で詳しく見てみます)。

```
import パッケージ名.クラス名;
```

```
import java.io.File;
```

```
import java.io.File;
class fileTest1{

    public static void main(String args[]){
        File cdirectory = new File("c:¥¥");

        String filelist[] = cdirectory.list();
        for (int i = 0 ; i < filelist.length ; i++){
            System.out.println(filelist[i]);
        }
    }
}
```

ファイルかディレクトリかの判別

```
import java.io.File;
class fileTest2{
    public static void main(String args[]){
        File cdirectory = new File("c:¥¥");
        File filelist[] = cdirectory.listFiles();
        for (int i = 0 ; i < filelist.length ; i++){
            if (filelist[i].isFile()){
                System.out.println("[F]" + filelist[i].getName());
            }else if (filelist[i].isDirectory()){
                System.out.println("[D]" + filelist[i].getName());
            }else{
                System.out.println("[?]" + filelist[i].getName());
            }
        }
```

```
    }
}
```

ファイルのパスの取得

Fileクラスで用意されている"getAbsolutePath"メソッドは絶対パスによるファイルの位置と名前を取得できます。

getAbsolutePath

```
public String getAbsolutePath()
```

では簡単なサンプルで試してみます。「C:¥tmp」ディレクトリに「test.txt」という名前のファイルを用意します。そして Java のサンプルプログラムも同じディレクトリに設置し、相対パスで指定した「test.txt」から File クラスのオブジェクトを作成し、絶対パスでの位置と名前を取得してみましょう。

```
import java.io.File;
class fileTest3{
    public static void main(String args[]){
        File file = new File("test.txt");
        String path = file.getAbsolutePath();
        System.out.println("File : " + path);
    }
}
```

ファイルを作成する

createNewFile

```
public boolean createNewFile() throws IOException
```

fileTest4.java

```
import java.io.File;
import java.io.IOException;
class fileTest4{
    public static void main(String args[]){
        File newfile = new File("c:¥¥tmp¥¥newfile.txt");
        try{
            if (newfile.createNewFile()){
                System.out.println("ファイルの作成に成功しました");
            }else{
                System.out.println("ファイルの作成に失敗しました");
            }
```



```
        }catch(IOException e){
            System.out.println(e);
        }
    }
}
```

ディレクトリを作成する

```
mkdir
public boolean mkdir()
```

次に作成した File クラスのオブジェクトに対して"mkdir"メソッドを実行します。

```
File newdir = new File("c:¥¥tmp¥¥sub");
newdir.mkdir();
```

これでディレクトリが作成されました。

fileTest5.java

```
import java.io.File;
class fileTest5{
    public static void main(String args[]){
        File newfile = new File("c:¥¥tmp¥¥sub");
        if (newfile.mkdir()){
            System.out.println("ディレクトリの作成に成功しました");
        }else{
            System.out.println("ディレクトリの作成に失敗しました");
        }
    }
}
```

ファイルの確認と削除

fileTest6.java

```
import java.io.File;

class fileTest6{
    public static void main(String args[]){
        File file = new File("c:¥¥tmp¥¥newfile.txt");

        if (file.exists()){
            if (file.delete()){
                System.out.println("ファイルを削除しました");
            }else{
                System.out.println("ファイルの削除に失敗しました");
            }
        }else{
            System.out.println("ファイルが見つかりません");
        }
```

```
    }
}
```

読み込みの許可と書き込みの許可

canRead メソッド:

```
canRead
public boolean canRead()
```

canWrite メソッド:

```
canWrite
public boolean canWrite()
```

```
setReadOnly
public boolean setReadOnly()
```

fileTest7.java

```
import java.io.File;

class fileTest7{
    public static void main(String args[]){
        File file = new File("c:¥¥tmp¥¥test.txt");

        checkPermission(file);

        if (file.setReadOnly()){
            System.out.println("ファイルを読み取り専用にしました");
        }else{
            System.out.println("読み取り専用に変更が失敗しました");
        }

        checkPermission(file);
    }

    private static void checkPermission(File file){
        if (file.canRead()){
            System.out.println("ファイルは読み込み可能です");
        }

        if (file.canWrite()){
            System.out.println("ファイルは書き込み可能です");
        }
    }
}
```


第１５章　テキストファイルの入出力

ファイルからテキストを読み込むためには色々なクラスが用意されています。まずは基本となる"FileReader"クラスを使ってみます。

```
File file = new File(file_name);
FileReader filereader = new FileReader(file);
```

テキストを読む

```
read
public int read() throws IOException
```

ファイルの最後まで読み込む

"read"メソッドを使うとまずファイルの先頭から 1 文字読み込みます。そして自動的に読み込む位置が次の文字に移ります。その為、連続して"read"メソッドを使うとファイルの先頭から順に 1 文字ずつ読み込んでくれます。そしてファイルの最後に達した時に"-1"という値を返してくれます。

ファイルを閉じる

```
close
public void close() throws IOException
```

streamTest1.java

```
import java.io.File;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

class streamTest1{
	public static void main(String args[]){
		try{
			File file = new File("c:¥¥tmp¥¥test.txt");
			FileReader filereader = new FileReader(file);

			int ch;
			while((ch = filereader.read()) != -1){
				System.out.print((char)ch);
			}

			filereader.close();
		}catch(FileNotFoundException e){
			System.out.println(e);
		}catch(IOException e){
			System.out.println(e);
```

```
        }
    }
}
```

読み込みの事前確認

"exists"メソッドでファイルの存在を確認し、"isFile"メソッドで対象の File クラスオブジェクトがファイルであるかどうかを確認後、"canRead"メソッドで読み込み可能かどうかを判別します。

streamTest2.java

```
import java.io.File;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

class streamTest2{

    public static void main(String args[]){
        try{
            File file = new File("c:¥¥tmp¥¥test.txt");

            if (checkBeforeReadfile(file)){
                FileReader filereader = new FileReader(file);

                int ch;
                while((ch = filereader.read()) != -1){
                    System.out.print((char)ch);
                }

                filereader.close();
            }else{
                System.out.println("ファイルが見つからないか開けません");
            }
        }catch(FileNotFoundException e){
            System.out.println(e);
        }catch(IOException e){
            System.out.println(e);
        }
    }

    private static boolean checkBeforeReadfile(File file){
        if (file.exists()){
            if (file.isFile() && file.canRead()){
                return true;
            }
        }

        return false;
    }
}
```


まとめてテキストを読む

FileReader クラスの持つ基本機能でファイルから読み込みは行うのですけど、それに BufferedReader クラスをかぶせて使うことでまとめて読み込む機能を持てるようになります。

```
File file = new File(file_name);
FileReader filereader = new FileReader(file);
BufferedReader br = new BufferedReader(filereader);
```

テキストを 1 行単位で読む

次にファイルの読み込みの仕方です。FileReaer クラスの時と同じく"read"メソッドもありますが、1 行まとめて読むための"readLine"メソッドが別に用意されています

```
readLine
public String readLine() throws IOException
```

ファイルを閉じる

また、BufferedReader クラスを使う場合にも、使い終わったら閉じておく必要があります。BufferedReader クラスでも FileReader クラスと同じように″close″メソッドを使います。

```
close
public void close() throws IOException
```

streamTest3.java

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

class streamTest3{

    public static void main(String args[]){
        try{
            File file = new File("c:¥¥tmp¥¥test.txt");

            if (checkBeforeReadfile(file)){
                BufferedReader br
                    = new BufferedReader(new FileReader(file));

                String str;
                while((str = br.readLine()) != null){
                    System.out.println(str);
                }

                br.close();
            }else{
```



```
                System.out.println("ファイルが見つからないか開けません");
            }
        }catch(FileNotFoundException e){
            System.out.println(e);
        }catch(IOException e){
            System.out.println(e);
        }
    }

    private static boolean checkBeforeReadfile(File file){
        if (file.exists()){
            if (file.isFile() && file.canRead()){
                return true;
            }
        }

        return false;
    }
}
```

テキストをファイルに書き込む

ここからはテキストをファイルに書き込んでいく方法を見てみましょう。まずは基本となる"FileWriter"クラスを使います。

```
File file = new File(file_name);
FileWriter filewriter = new FileWriter(file);
```

ファイルへの書き込み

ではファイルに書き込みを行います。"write"メソッドを使います。"write"メソッドは書き込む値として文字コードを使ったり、文字列を指定したりといくつか用意されているのですが、ここでは文字列を書き込むメソッドを見てみます。

```
write
public void write(String str) throws IOException
```

ファイルを閉じる

```
close
public void close() throws IOException
```

class streamTest4.java

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

class streamTest4{

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



```
        try{
            File file = new File("c:¥¥tmp¥¥test.txt");

            if (checkBeforeWritefile(file)){
                FileWriter filewriter = new FileWriter(file);

                filewriter.write("こんにちは¥r¥n");
                filewriter.write("お元気ですか¥r¥n");

                filewriter.close();
            }else{
                System.out.println("ファイルに書き込めません");
            }
        }catch(IOException e){
            System.out.println(e);
        }
    }

    private static boolean checkBeforeWritefile(File file){
        if (file.exists()){
            if (file.isFile() && file.canWrite()){
                return true;
            }
        }

        return false;
    }
}
```

ファイルに追加で書き込む

既にファイルに書かれているテキストは消さずに、ファイルの最後に追加でテキストを書き込む方法を見ていきます。と言っても変更箇所は 1 箇所だけで FireWriter クラスのオブジェクトを作成する時に、下記のように 2 番目の引数として"true"を指定するだけです。

```
File file = new File(file_name);
FileWriter filewriter = new FileWriter(file, true);
```

streamTest5.java

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

class streamTest5{

    public static void main(String args[]){
        try{
            File file = new File("c:¥¥tmp¥¥test.txt");

            if (checkBeforeWritefile(file)){
                FileWriter filewriter = new FileWriter(file, true);
```



```
                    filewriter.write("はい。元気です¥r¥n");
                    filewriter.write("ではまた¥r¥n");

                    filewriter.close();
                }else{
                    System.out.println("ファイルに書き込めません");
                }
            }catch(IOException e){
                System.out.println(e);
            }
        }

        private static boolean checkBeforeWritefile(File file){
            if (file.exists()){
                if (file.isFile() && file.canWrite()){
                    return true;
                }
            }

            return false;
        }
}
```

第１６章　ArrayList クラス

ArrayList を使うには、まず ArraList クラスのオブジェクトを作成します。ArrayList クラスのコンストラクタの１つは下記のようになっています。

```
ArrayList
public ArrayList()
```

```
ArrayList<型> 変数名 = new ArrayList<型>();
```

対応するクラス	基本型
Boolean	boolean
Character	char
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Doubule	double

値の格納

ArrayListクラスのオブジェクトを作成したら、実際に要素を追加します。ArrayListクラスで用意されている"add"メソッドを使います。

add
public boolean add(E o)

要素の取り出し

次に格納した要素を取り出します。ArrayList クラスで用意されている"get"メソッドを使います。

get
public E get(int index)

collectionTest1.java

```
import java.util.ArrayList;
class collectionTest1{
    public static void main(String args[]){
        ArrayList<String> array = new ArrayList<String>();
        array.add("日本");
        array.add("ブラジル");
        array.add("イングランド");
        array.add("ポルトガル");
        array.add("フランス");
        String country = array.get(2);
        System.out.println(country);
    }
}
```

格納されている要素数

ArrayListクラスのオブジェクトを使用している時に、現在格納されている要素数を確認するにはArrayListクラスで用意されている"size"メソッドを使います。

size
public int size()

collectionTest2.java

```
import java.util.ArrayList;
class collectionTest2{
    public static void main(String args[]){
```



```
        ArrayList<String> array = new ArrayList<String>();
        array.add("日本");
        array.add("ブラジル");
        array.add("イングランド");
        array.add("ポルトガル");
        array.add("フランス");
        System.out.println("登録データ数:" + array.size());
        for (int i = 0 ; i < array.size() ; i++){
            String country = array.get(i);
            System.out.println(country);
        }
    }
}
```

要素の置き換え

ArrayListクラスで用意されている他のメソッドを見ていきます。まずは既に格納されている要素を別の要素で置き換えます。ArrayListクラスで用意されている"set"メソッドを使います。

```
set
public E set(int index, E element)
```

collectionTest3.java

```
import java.util.ArrayList;
class collectionTest3{
    public static void main(String args[]){
        ArrayList<String> array = new ArrayList<String>();
        array.add("日本");
        array.add("ブラジル");
        array.add("イングランド");
        array.add("ポルトガル");
        array.add("フランス");
        for (int i = 0 ; i < array.size() ; i++){
            String country = array.get(i);
            System.out.println(country);
        }
```



```
        System.out.println("¥r¥n3番目の国をイタリアへ置き換えます¥r¥n");
        array.set(2, "イタリア");
        for (int i = 0 ; i < array.size() ; i++){
            String country = array.get(i);
            System.out.println(country);
        }
    }
}
```

第１７章　HashMap クラス

new 演算子を使って HashMap のオブジェクトを作成します。

HashMap<型 1,型 2> 変数名 = new HashMap<型 1,型 2>();

HashMap の場合は型を 2 つ指定します。1 つ目はキーの型、2 つ目は格納する要素の型となります。例えばキーを Integer 型、要素を String 型とした場合は下記のような感じとなります。

HashMap<Integer,String> map = new HashMap<Integer,String>();

キー　　要素

1000　　テレビ

1001　　ビデオ

2000　　エアコン

要素の格納と取り出し

```
HashMap<String,String> map = new HashMap<String,String>();

map.put("りんご", "apple");
map.put("ぶどう", "grapes");

System.out.println(map.get("りんご"));
```

hashmapTest1. java

```
import java.util.HashMap;
class hashmapTest1{

    public static void main(String args[]){
        HashMap<String,String> map = new HashMap<String,String>();
```



```
            map.put("りんご", "apple");
            map.put("ぶどう", "grapes");

            if (map.containsKey("りんご")){
                System.out.print("りんごを英語にすると");
                System.out.println(map.get("りんご"));
            }else{
                System.out.println("指定したキーは存在しません");
            }
    }
}
```

