-
  -
  -
  -
-
  -
  -
  -
-
  -
  -
  -
  -
-
  -
  -
  -
  -
  -
-
  -
  -
  -
  -
  -
-
-
  -
  -
  -
-

http://java.sun.com/
JDK をダウンロードしてインストール

他でインストールしたディレクトリをコピーしても良い。
ただし、bin にパスを通すこと。

```
javac test.java
```

```
java test
```

Java には大きく分けて

・プリミティブ型
・オブジェクト型（参照型）

の２種類の型が存在する。

プリミティブ型

byte	1 バイト整数 (-128 〜 127)
short	2 バイト整数 (-32768 〜 32767)
int	4 バイト整数 (-2147483648 〜 2147483647)
long	8 バイト整数 (-9223372036854775808 〜 9223372036854775807)
float	4 バイト単精度浮動小数点数
double	8 バイト倍精度浮動小数点数
char	2 バイト文字データ (\u0000 〜 \uffff)
boolean	論理値 true または false

オブジェクト型（参照型）

プリミティブ以外の全て。
配列もオブジェクト型。

```
int a;
Object b;
```

基本的には全て値渡し。
オブジェクト型の場合は、アドレスを値渡しするため参照渡しと説明されることもある。

型	渡し方
プリミティブ	値渡し
オブジェクト型	アドレスを値渡し

```
if (a == 1) {
  System.out.println("a is 1");
}
```

```
for (int i = 0; i < 10; i++){
  System.out.println(i);
}
```

ループ中断処理

```
for (int i = 0; i < 10; i++){
  if (i == 5){
    break; // i == 5 の時、ループを抜ける
  }
  System.out.println(i);
}
```

ラベルを使ったループ中断処理

```
end:
for (int i = 0; i < 10; i++){
  for (int j = 0; j < 10; j++){
    if (j == 5){
      break end;
    }
    System.out.println(j);
  }
}
```

ループのスキップ処理

```
for (int i = 0; i < 10; i++){
  if (i == 5){
    continue; // i == 5 の時、以下の処理を行わずにループのはじめに戻る
  }
  System.out.println(i);
}
```

while

```
while (i < 10){
  System.out.println(i);
  i++;
```

```
}
```

do while

```
do {
  System.out.println(i);
  i++;
} while(i < 10);
```

```
switch (i) {
case 1:
  System.out.println("1 です ");
  break;
case 2:
case 3:
  System.out.println("2 か 3 です ");
  break;
default:
  System.out.println(" その他です ");
  break;
}
```

演算子	意味
a + b	a と b を足す
a - b	a から b を引く
a * b	a と b をかける
a / b	a を b で割る
a % b	a を b で割った余り
a++	a の値を 1 増やす（インクリメント）
a--	a の値を 1 減らす（デクリメント）

オブジェクト型の比較は equals メソッドを使うこと。

演算子	意味
a == b	等しい
a != b	等しくない
a > b	a が b より大きい
a >= b	a が b 以上
a < b	a が b より小さい
a <= b	a が b 以下

左辺、右辺両方を判定する

演算子	意味
a & b	論理積（かつ） AND
a | b	論理和（または）OR
a ^ b	排他的論理和 XOR
!a	否定

必要ない場合は右辺は判定されない。

演算子	意味
&&	論理積（かつ）
||	論理和（又は）

演算子	意味
>>n	算術右シフト（符号付 n ビット右シフト）
>>>n	論理シフト（0 埋め n ビット右シフト）
<<n	n ビット左シフト
~	ビット反転

```
int[] a = new int[10];
int[] b = new int[] {1,2,3,4,5};
int[] c = {1,2,3,4,5};

System.out.println(a.length);
System.out.println(b.length);
System.out.println(c.length);
```

```
int[][] a = new int[10][20];
int[][] b = new int[][] { {1,2}, {3,4}, {5,6,} };
int[][] c = { {1,2}, {3,4}, {5,6,} };

System.out.println(a.length + " " + a[0].length);
System.out.println(b.length + " " + b[0].length);
System.out.println(c.length + " " + b[1].length);
```

Vector や ArrayList クラスを使う

```
Vector vec = new Vector();
```

```
vec.add("test1");
vec.add("test2");
vec.add("test3");

System.out.println(vec.get(0));
```

```
Hashtable hash = new Hashtable();

hash.put("key1","hogehoge1");
hash.put("key2","hogehoge2");
hash.put("key3","hogehoge3");

System.out.println(hash.get("key2"));
```

```
int a[] = {1,2,3};
int b[] = {4,5};
int c[][] = new int[2][];
c[0] = a;
c[1] = b;
System.out.println(c.length + " " + c[0].length + " " + c[1].length);
```

配列は Object クラスを継承しているので、以下のようなことも可能

```
int a[] = {1,2,3};
int b[] = {4,5};
Object o[] = new Object[2];
o[0] = a;
o[1] = b;

Object o2 = o;
System.out.println(o2 instanceof Object[]);
System.out.println(o2.getClass().isArray());
```

戻り値、引数なし

```
public void test(){
  // なにか処理
}
```

戻り値、引数あり

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

```
class test {
  int a;
  int b;
}
```

```
interface itest {
  public void test();
  public void test2(int a, int b);
}
```

クラスがクラスを継承する場合は extends
クラスがインターフェイスを実装する場合は implements

```
class test extends Frame implements Runnable {
  public void run() {

  }
}
```

インターフェイスがインターフェイスを継承する場合は

```
interface itest extends Runnable {
}
```

修飾子	--	説明
アクセス修飾子	public	制限はありません。どこからでもアクセス可能です。
--	protected	同一パッケージ内、またはそのサブクラスからのみアクセス可能です。
--	private	同一クラス内でのみアクセス可能です。
修飾子	abstract	抽象メソッドや抽象メソッドをもつクラスに指定します。 インターフェースは元々抽象メソッドしか定義できない為、省略されることが多いです。
--	final	変数を変更したくない場合、このクラスを継承することを禁止する（サブクラスを作成させたくない）場合に指定します。
--	strictfp	「float」や「double」の浮動小数点を用いた演算を厳密 （プラットフォームに依存しない（IEEE754 に従う））に行いたいクラスに指定します。

--	static	クラス固有のものとして、同一クラスからなる全インスタンスに共有させる場合に指定します。
--	transient	シリアライズ（バイトストリーム変換）の時に値を保存したくない（一時的に使用する値・初期化したい値）フィールドに指定します。
--	volatile	コンパイラによる最適化をさせたくないフィールドに指定します。（マルチスレッド等による参照値の相違をなくします。）
--	native	プラットフォーム依存のコードで定義します。（型、メソッド名、引数の型といったインターフェースにみを定義し、中身はＣ言語などの他の言語を用いて実装します。）
--	synchronized	マルチスレッド環境において、同期処理（インスタンス単位で排他制御）させる場合に指定します。