

情報工学実験 第1回



# Javaの導入と開発環境 リスト構造

---

2019年10月4日



# この実験の目的

- 実践的なプログラム技術の習得



- なぜ？今からまたプログラミング？
  - 卒業研究に向けて課題解決型のプログラム技法に慣れる
  - 作りたいものが作れるスキルを身に着ける
  - プログラミングスキルの幅を広げる



卒業研究を効率的に進められる  
就職試験で話すネタが増える



5年生に  
なって  
役立つ



# なぜJava？

---

## ■ Java言語

- オブジェクト指向のプログラミング言語として研究、開発された
- C言語と基本的には同じ文法
  - なじみやすい
- C言語が苦手とする部分を簡単に実装
  - 文字列処理
  - ネットワーク通信
  - プラットフォーム依存性
  - ライブラリの依存度



# この実験の流れ

- 課題の説明; 10分程度
    - 課題解決のための**基本的な仕組みを説明**
  - プログラムの開発; 70分程度
- 
- 課題解決へのヒント; 10分程度
  - 発表可能時刻: 15:10～(ヒント終了後)
  - 強制的な発表の開始時刻: 15:40～
    - 毎回発表順を決めます
    - 1人2分程度(発表できない場合は後回し)
    - 発表により授業点が加算されます



# 課題

---



# 今日の課題1: 動作テスト

---

- 実験用サイトへのアクセス
  - LMS(学習支援システム)を使います
  - 4年情報工学実験のページを確認してください
- コンパイラのテスト
  - Sample1.javaを開発環境で確認
  - 何をするプログラムか考える
  - プログラムは意図したとおりに動かないので、ソースコードをよく見て考えて修正
- デバッガのテスト
  - 開発環境で変数の変化を確認



# 今日の課題2: プログラミング

- 2つの得点リストを結合し、昇順に並び替えよ

数学	100	56	43	55	78	92	85	64	67
化学	78	92	56	39	32	78	64		

- (任意課題) 名前と得点のリストからの検索
  - 60点未満の学生名を出力せよ
  - 名前から得点を検索せよ(キーボードからの入力は不要)

名前	Sato	Matsuno	Tanaka	Sakagami	Kirisima
得点	100	56	43	55	78

名前	Yamaguchi	Yuba	Shinoda	Nagata	Akiyama
得点	92	85	64	67	92



# 講義

---



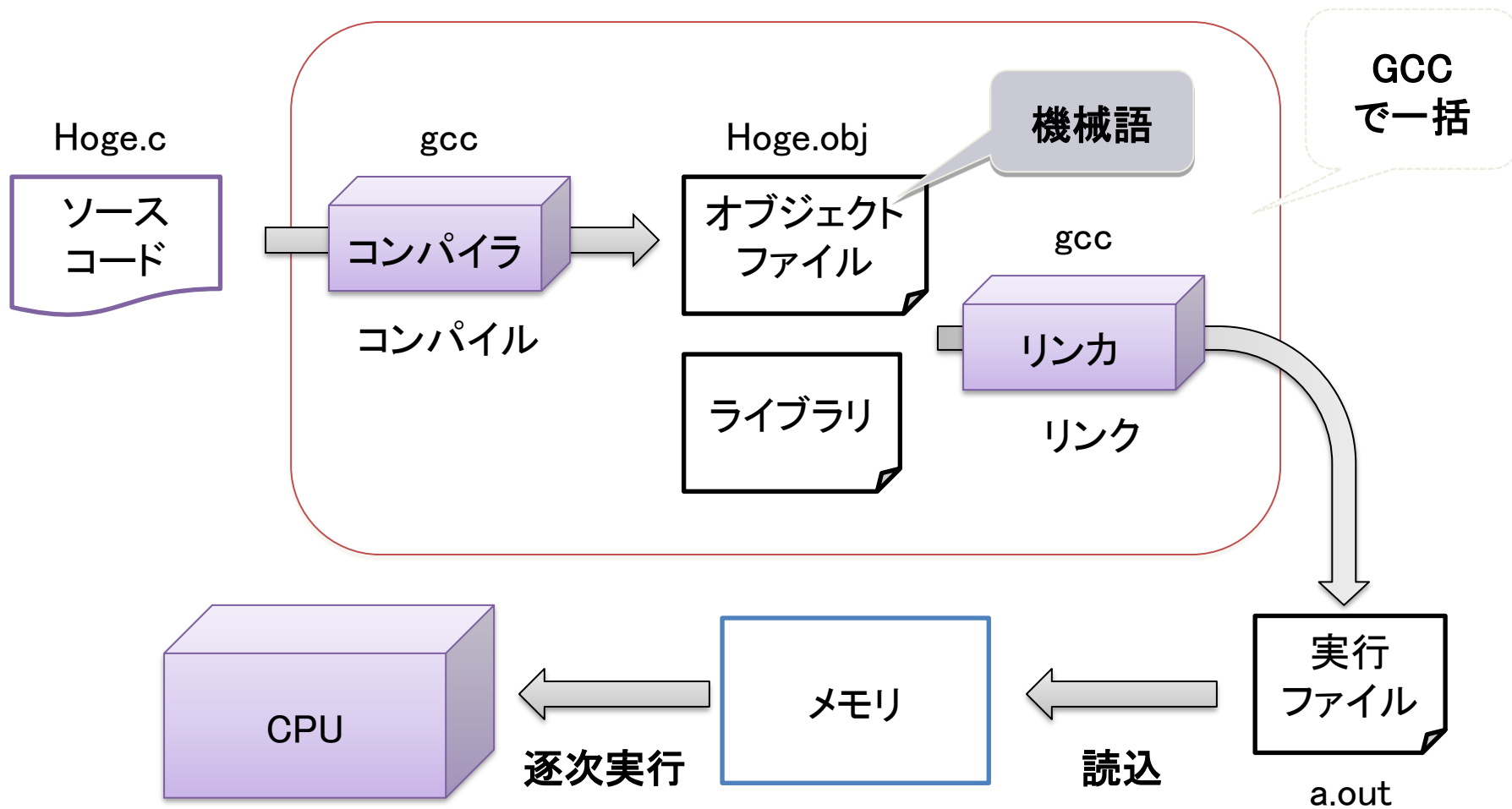


# Javaとは

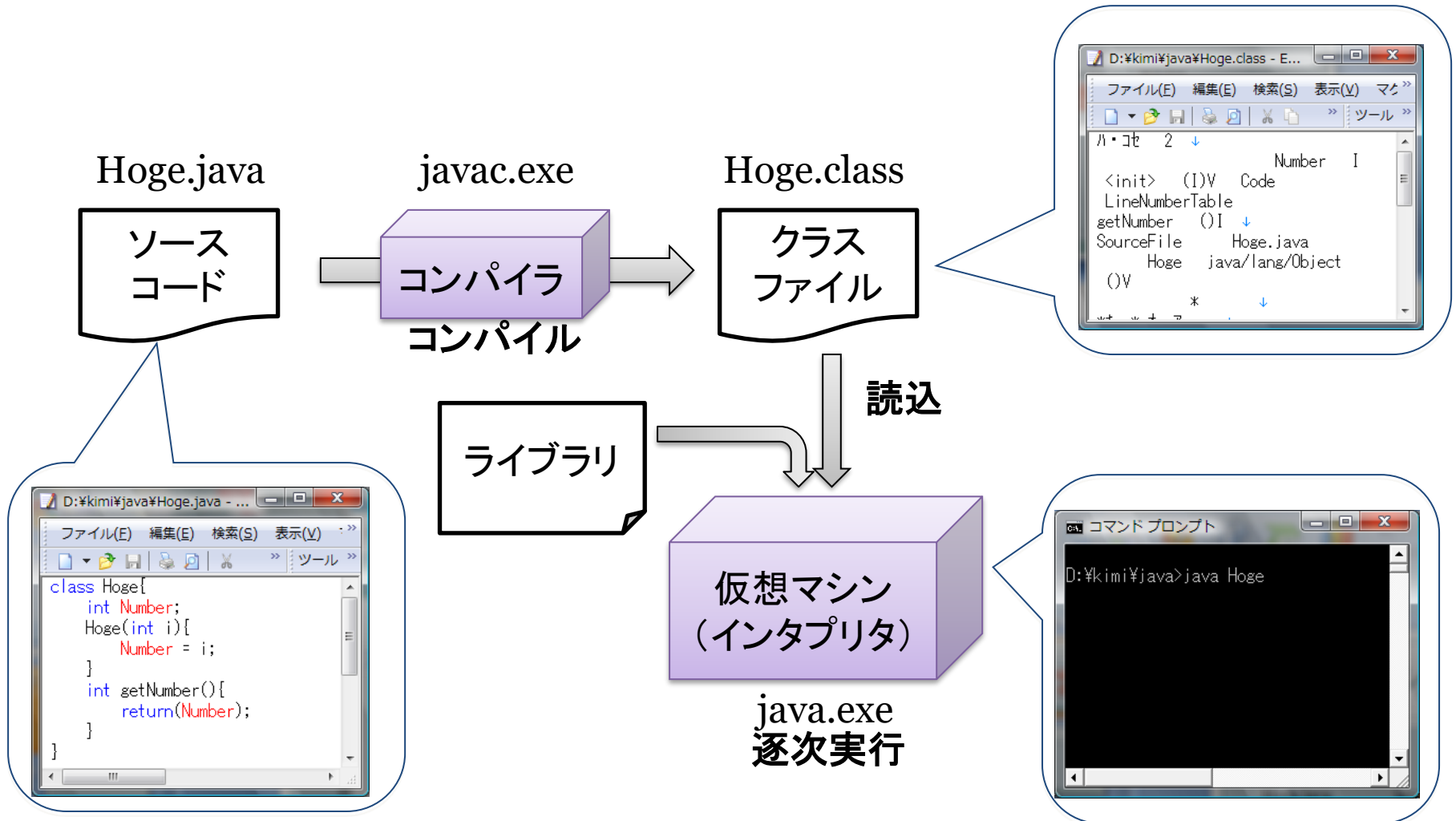
---

- プラットフォームに依存しない
  - インタプリタ型
    - 正確にはジャストインタイムコンパイル方式
  - 仮想マシン
  - 中間コードの生成(クラスファイル)
- クラスベースのオブジェクト指向言語
- メモリ管理が自動(ガベージコレクション)
- 豊富なライブラリ
  - ソケット、スレッド、XML、その他いろいろ
- 容易なGUI実装

# Cでの実行までの流れ



# Javaでの実行までの流れ



# 手続き型とオブジェクト指向

## ■ 手続き指向(型)プログラミング

手続きが主体

- 処理すべき内容を一般化し、手順(手続き)に分解
- 手続きを順に行い目的となる課題を解決する

```
curry=boil(fry(potato,carrot,meat),spice);
```

## ■ オブジェクト指向型プログラミング

モノが主体

- 課題に登場する要素を切り分ける(=**オブジェクトの整理**)
- 要素が持つ要素と機能を設計する(=**オブジェクトの設計**)
- オブジェクト同士を関係付けて課題を解決する

```
Pot.add( potato, carrot, meat )  
Pot.fry(5);  
Pot.boil(30);
```



# オブジェクト指向の特徴

---

- カプセル化

- ▲ 変数をプライベートにしてメソッドでアクセス

- アクセス方式の唯一化によるデータ保護

- 継承

- ▲ 哺乳類クラスを継承して犬クラスと猫クラスを...

- 処理の共通化を実現する技術

- ポリモーフィズム（多態性）

- ▲ intもStringも受け入れるprint文

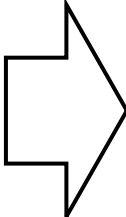
- インタフェースによる実装の抽象化



# Hello World!

- First.javaというファイル名でプログラムを記述

```
public class First{  
    public static void main(String args[]){  
        System.out.println("Hello World!");  
    }  
}
```



C言語よりやたらと文字を打たないといけない！？  
Print文だけでこんなに書くことがあるの？

Java独特の書き方：手続き型言語との共存が目的



# Javaの言語構造

---

- C言語に近い言語構造
  - 一行はセミコロン「;」で終わる
  - 改行文字・空白文字が区切りトークン
  - { }による入れ子構造
- 条件分岐
  - if文およびswitch-case文
- 繰り返し
  - for文とwhile文
  - foreachに対応する拡張forループ
    - わかってくるとこのループはかなり便利です

# If文の構造

```
boolean bool = true;           boolean型の変数boolの宣言と値の代入
if(bool){                       if文によるブロックの実行判定
    // boolが真の時に実行されるブロック
}

int big = 100;
int small = 5;                  int型の変数big,smallの宣言と値の代入
if(big > small){                 演算子によるboolean型の算出
    // bigのほうが大きい時に実行される
}
```

- if文の引数はboolean型
  - 演算結果
  - メソッドの戻り値

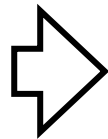


# forループ

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

int型で変数iを0で定義  
1ずつ増加させながら10未満の間  
繰り返す

```
for(式1;式2;式3){  
    ブロック内のコード  
}
```



1. 式1を実行
2. 式2が成立したらブロック内を実行
3. 式3を実行し、2に戻る

- for内でループ回数を利用する場合に有効
  - 配列などの処理で前後の関係を使う場合
    - フィボナッチ数列 ( $a[n] = a[n-1] + a[n-2]$ )

# Whileループ

```
int i=0;
```

```
while(i<10){
```

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

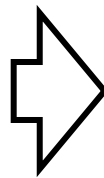
```
    i++;
```

```
}
```

int型で変数iを0で定義  
iが10未満の間繰り返す

iを1増加させる

```
while(boolean){  
    ブロック内のコード  
}
```



1. boolean値が真ならばブロック内のコードを実行
2. 1に戻る

- whileは以前配列の処理に多く使われていた
  - 今はもっと楽な書き方ができるようになった

# 拡張forループ (for eachループ)

- 配列をforループにより処理
  - 各要素に**同一の処理**を行う
  - 利用機会が非常に多い

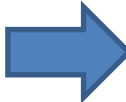
```
int[] xValues = {1,2,3,4,5};  int型を格納する静的な配列を定義
int sum = 0;                  int型で合計値を格納する変数を定義
```

```
for(int v : xValues){        int型変数vに配列の要素を順に代入して
    sum += v;                繰り返す
}
```

- ループ用の変数の準備が不要
  - 多重ループでのi,j,kなどわかりにくい変数を必要としない
  - 動的な配列への対応が容易



# Javaプログラミングの注意事項

- Javaのファイル名ルール
    - 始めにくい第一歩
  - Sample1.javaファイルでは、“Sample1”というクラスをpublicで宣言しなければならない
    - ファイル名と同じ名前のpublicクラス
      - 拡張子部分は不要
    - 大文字小文字を区別する
      - Windows環境では注意が必要
    - 同一ファイルでは他のpublicクラスを宣言できない
      - publicでなければ自由に他のクラスを宣言できる
-  面倒な作業はソフトウェア(開発環境)にやらせよう

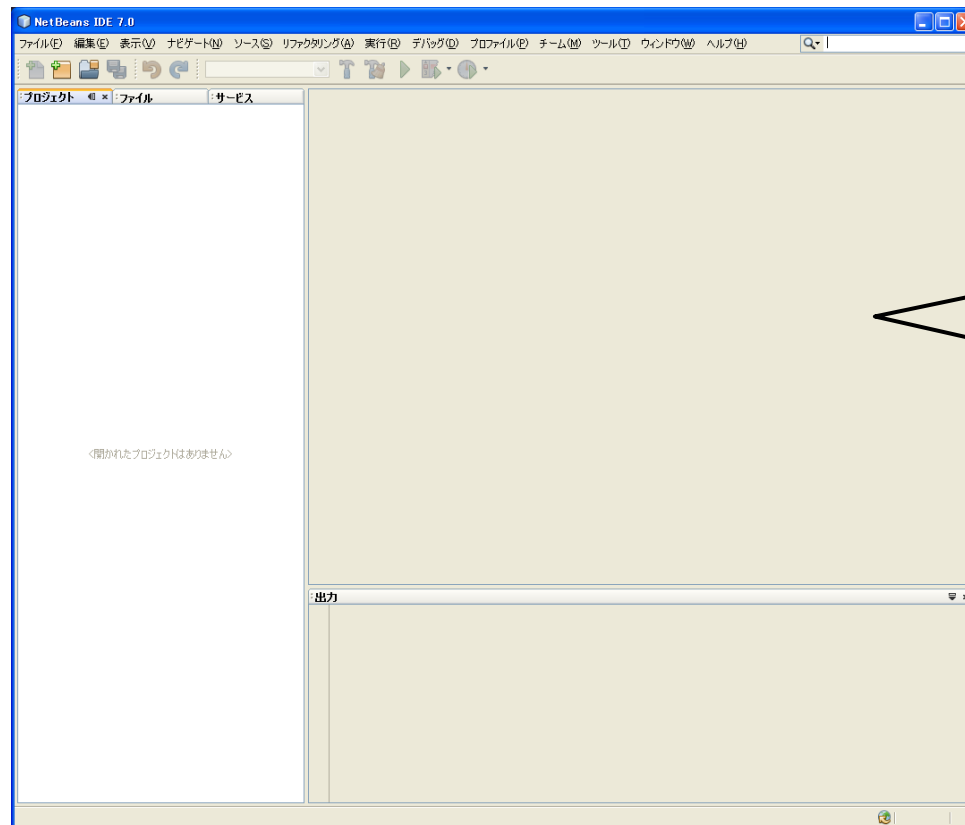


# 開発環境: NetBeans

- Sunが制作した無償の開発環境
  - オープンソースプロジェクトとして開発されている
  - GUIの作成が簡単にでき、標準で多言語(日本語)対応
  - Webページも日本語化されている
- ダウンロードとインストール
  - <https://ja.netbeans.org/>
  - 開発環境はインストール済み
- 他の開発環境
  - Eclipse: 古くから使われてきた多機能なSDK  
Androidの開発などはこちらで行うとよい  
⇒ Androidにも専用の開発環境ができました

# Net Beansの起動

## ■ 起動直後の画面



ここに使い方などの説明が出る場合がある

タブの「×マーク」で閉じることができる



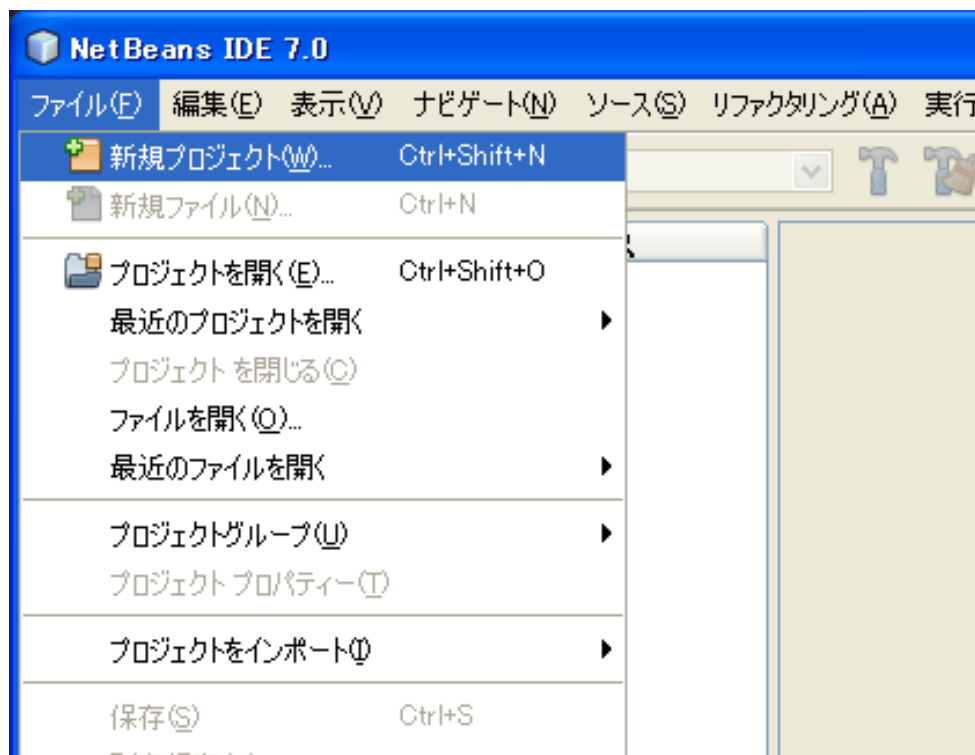


# 最初のプログラムを作ってみよう！

- NetBeansでは複数のソースコード(クラス)をまとめてプロジェクトとして管理する
    - クラスを作る前にプロジェクトを作る必要がある
    - 主クラス(main関数が入ったクラス)を決めておくと、そのクラスから実行を始めることができる
    - main関数そのものを自動で作成することも可能
  - 課題解決用のプログラムを作ろう
    - プロジェクト名: 4EJ\_Kadai1\_(名列番号)
    - 主クラス名: Sample1.java
- として作ってみよう

# プロジェクトの作成

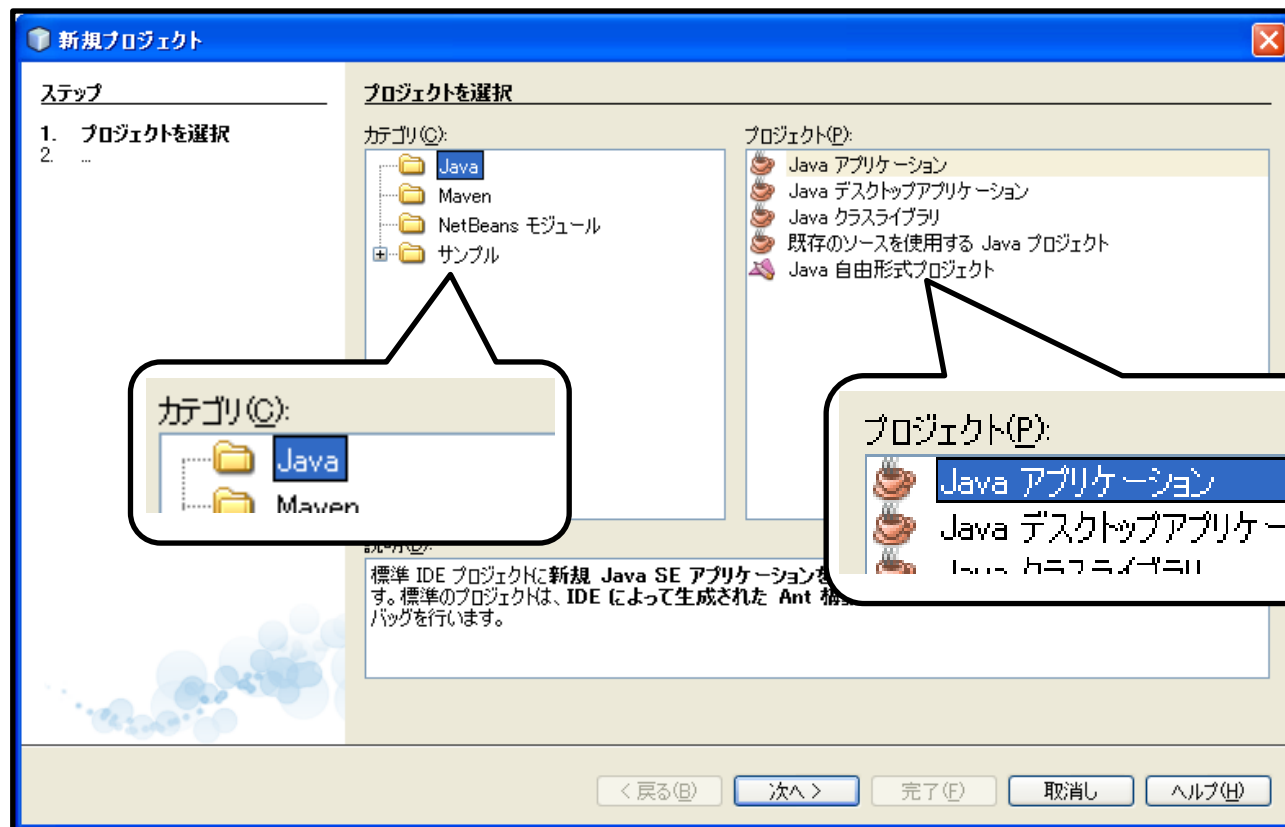
- メニューの「ファイル」から「新規プロジェクト」を選択





# プロジェクトの種類を選択

- カテゴリは「Java」
- プロジェクトは「Javaアプリケーション」を選択
- 最後に「次へ」を押す



# プロジェクト名の決定

- 任意の名前を設定し、「完了」をクリック
  - 主クラス(main関数が書かれているクラス)を作るオプションを入れておくと作業が楽

プロジェクト名

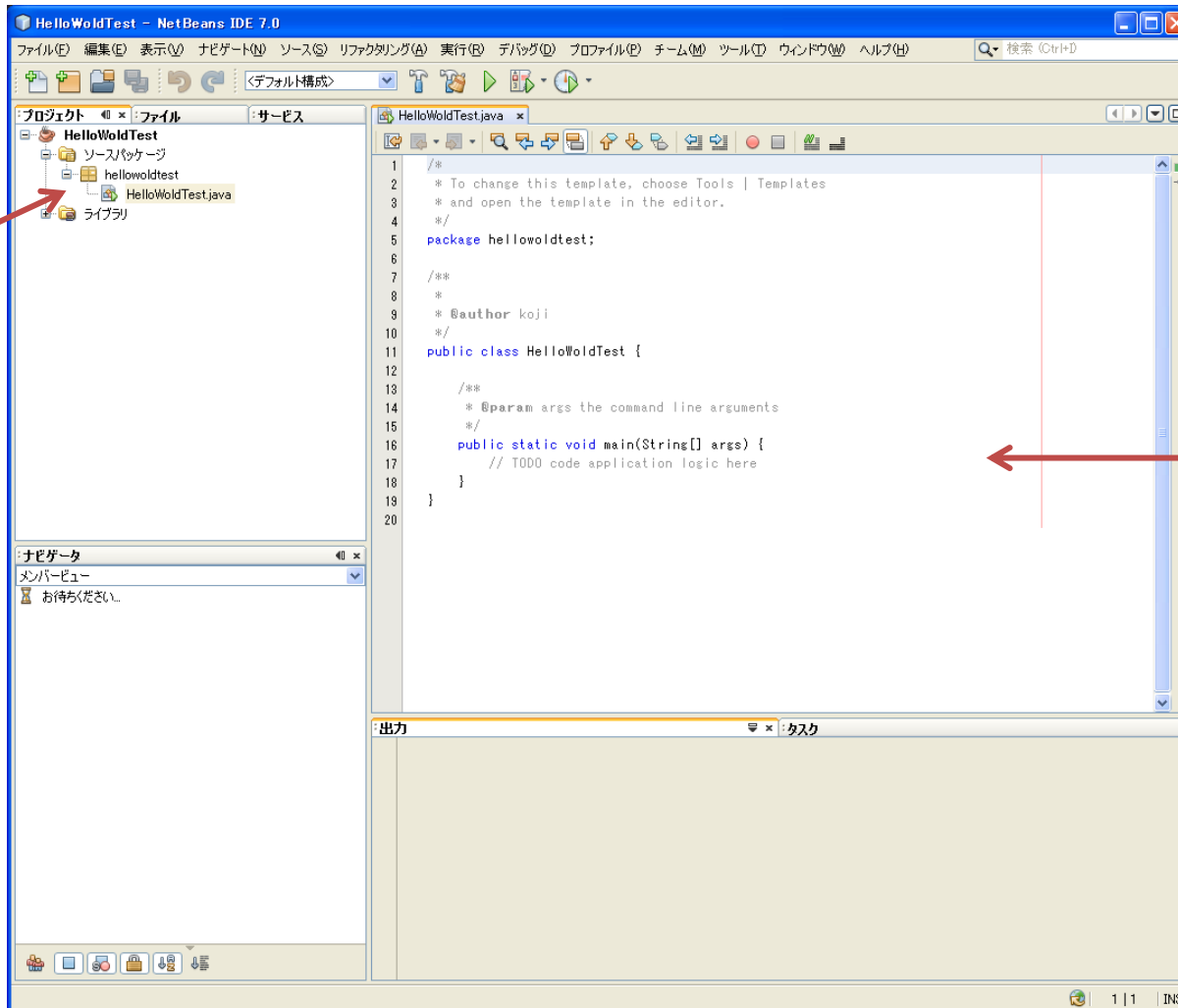
4EJ\_Kadai1\_(名列番号)

主クラス名

Sample1

# ソースコードの作成

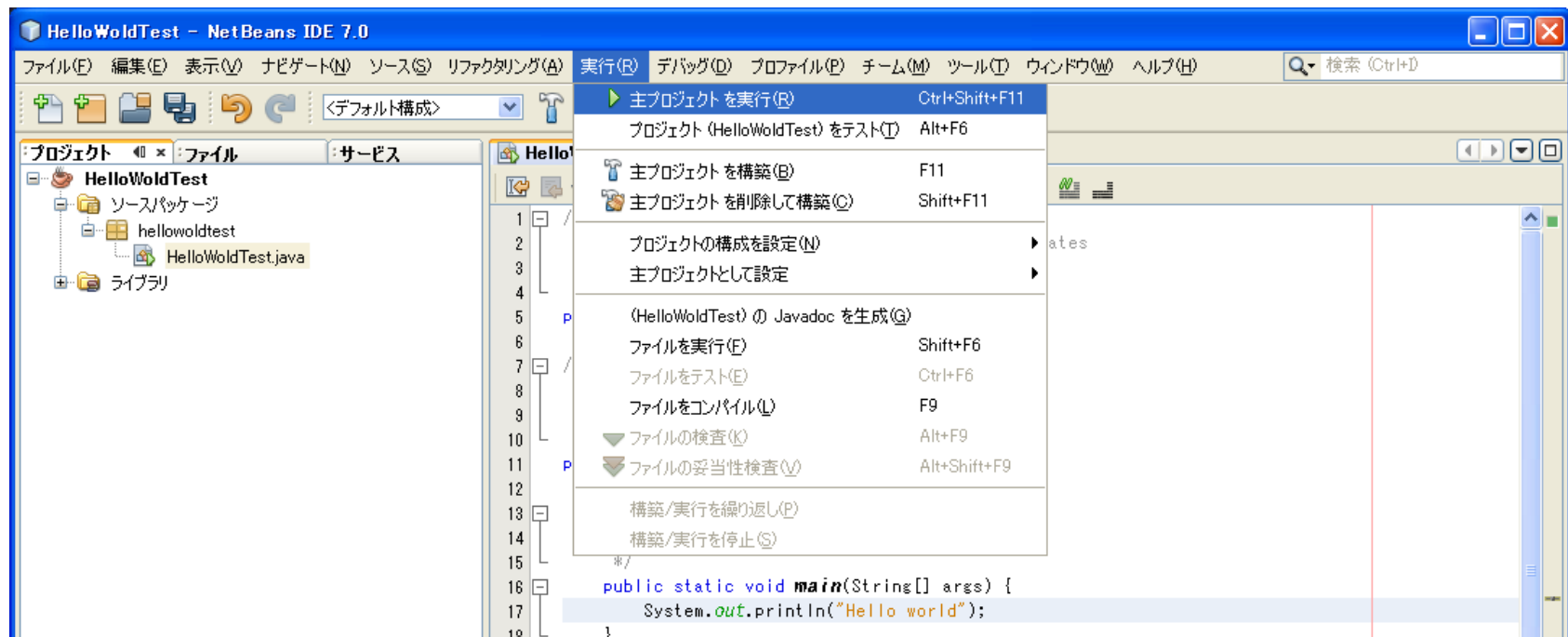
編集する  
ファイル  
を選ぶ



ファイルを  
編集

# プログラムの実行

- メニューの「実行」から「主プロジェクトを実行」または「ファイルを実行」を選択
  - GUIの「▶」ボタンでも同じ効果





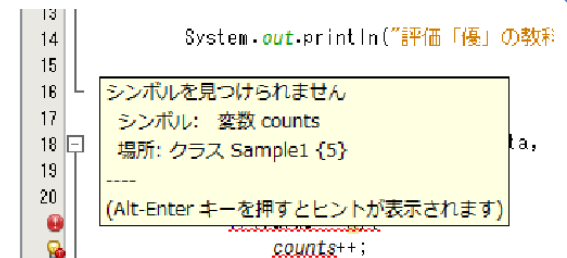
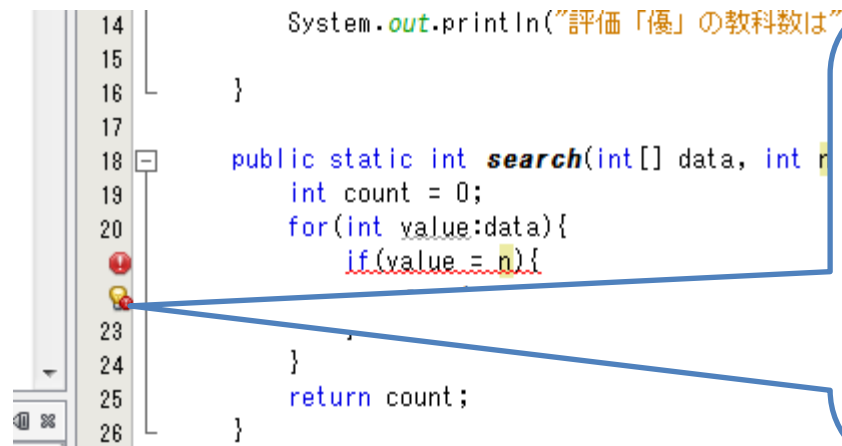
# 先ほどのプログラムを作ろう

- **Sample1.java**からプログラムを転記(打ってもOK)

```
public class Sample1 {  
    public static void main(String[] args) {  
        // The application start from this method.  
        int[] data = {47, 98, 39, 65, 71, 57, 77, 80, 90};  
        int c = search(data,80);  
        System.out.println("評価「優」の教科数は"+c+"個でした。");  
    }  
  
    public static int search(int[] data, int n) {  
        int count = 0;  
        for(int value:data){  
            if(value == n){  
                count++;  
            }  
        }  
        return count;  
    }  
}
```

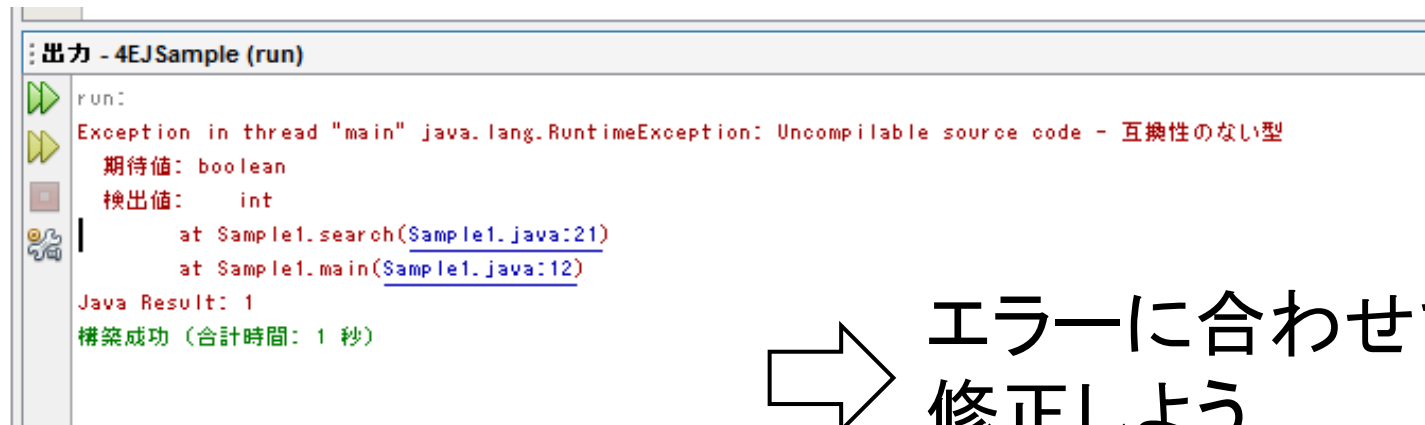
# エラーになる場合

- ソースコードに直接表示(コンパイルエラー)



マウスを載せると  
内容が分かります

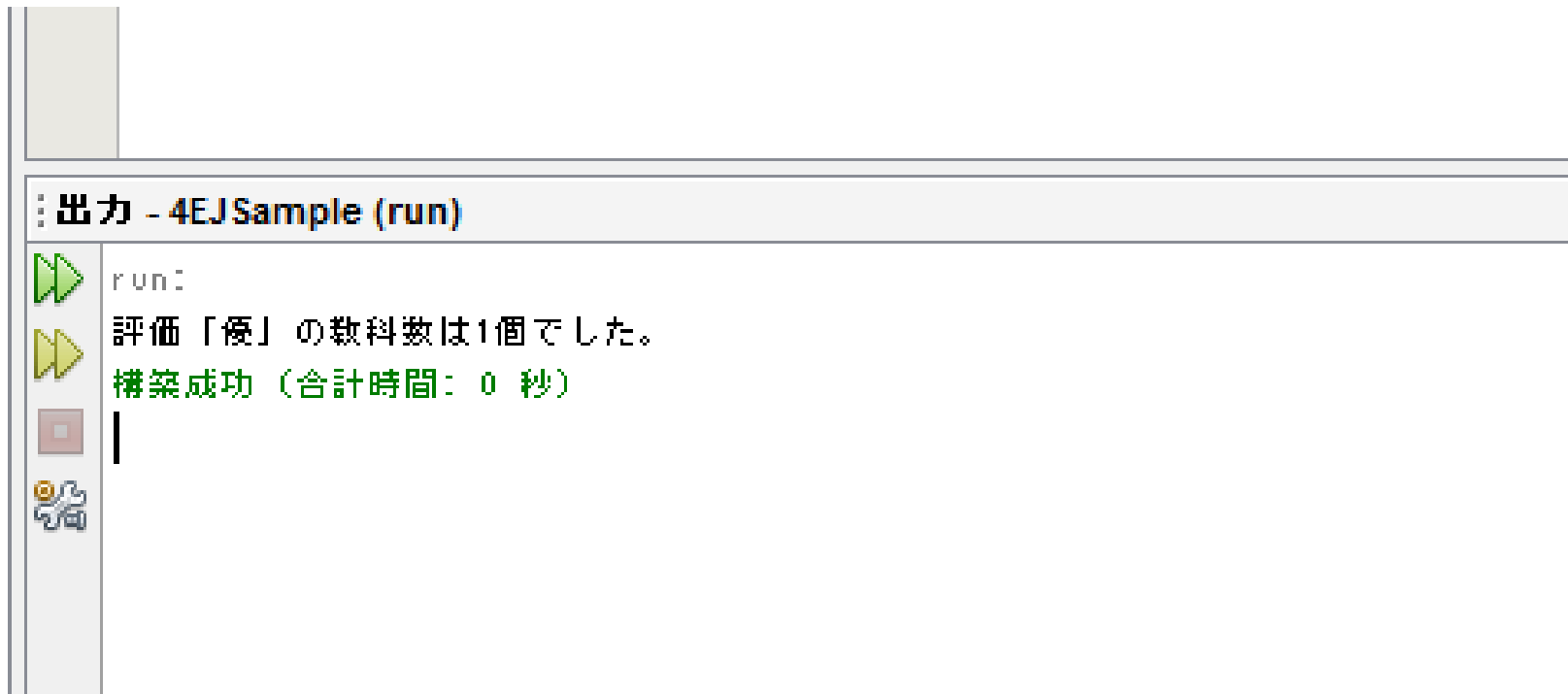
- 出力結果に表示(実行時エラー)



エラーに合わせて  
修正しよう

# 実行結果の確認

- CUI(System.out,System.err)の出力結果は、画面下部の「出力」内に表示される





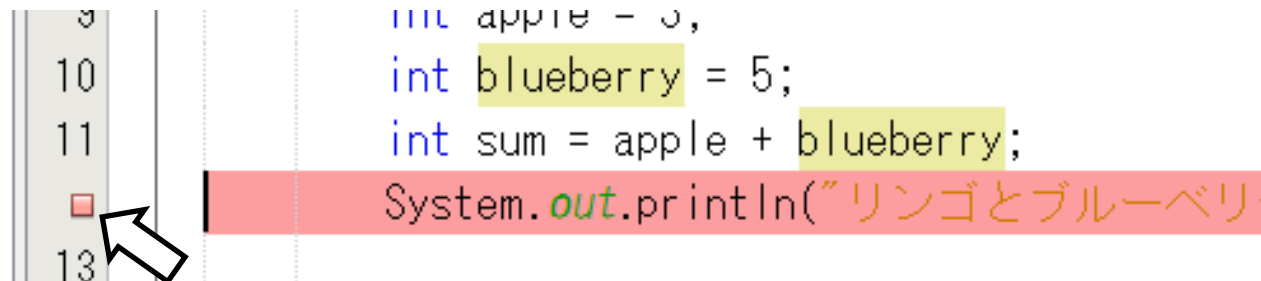
# コンパイルエラーとデバッグ


- IDE(統合開発環境)の本格的な利用
- プログラムのコンパイルエラー
  - Javaの場合は、1行追加するごとに自動でチェック
  - エラーはGUI上に、修正案と共に表示される
- プログラムの動作が不信な場合のデバッグ
  - プログラムを実行中に一時停止し、変数の状態や、関数の戻り値などをチェックしたいことがよくある
  - CUIからのコンパイル、実行ではprint文をいれて対応しているが...

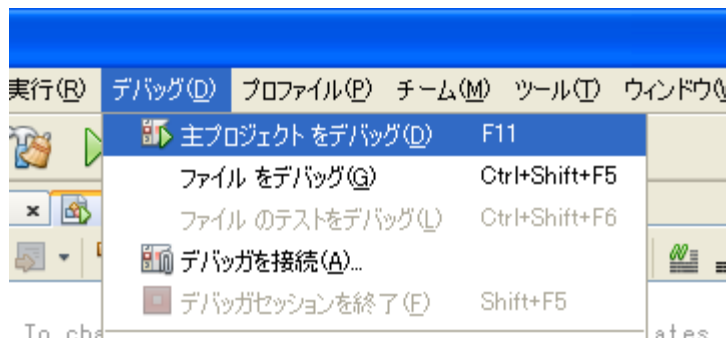


# プログラムの修正とデバッグ

- プログラムの12行目に「ブレークポイント」を設置
  - 行番号をクリックする
  - この行に処理が行われる前に一時停止する



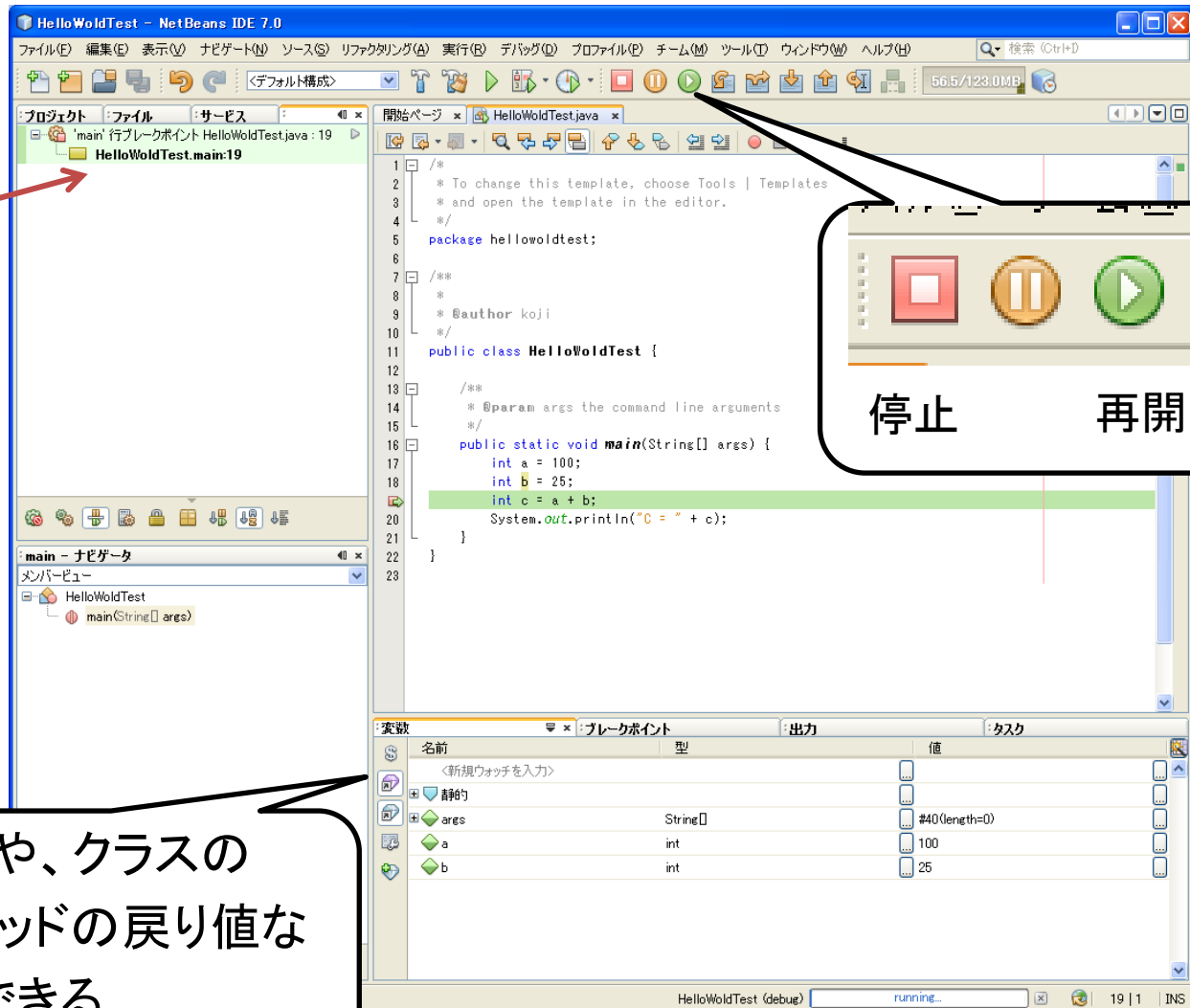
- メニューの「デバッグ」から、「主プロジェクトをデバッグ」
  - 「」ボタンでも同じ効果




# デバッグ中の様子

どのブレーク  
ポイントで停止  
しているか表示

変数の値や、クラスの  
状態、メソッドの戻り値な  
ども確認できる



- 
- 
- 以上でガイダンスは終わりです
  - 課題解決のためのヒントをみながら  
基本課題に挑戦しましょう



# 今日の課題2: プログラミング

- 2つの得点リストを結合し、昇順に並び替えよ

数学	100	56	43	55	78	92	85	64	67
化学	78	92	56	39	32	78	64		

- (任意課題) 名前と得点のリストからの検索
  - 60点未満の学生名を出力せよ
  - 名前から得点を検索せよ(キーボードからの入力は不要)

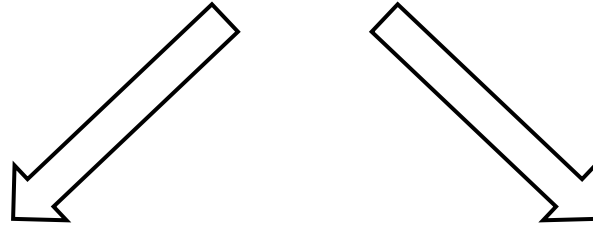
名前	Sato	Matsuno	Tanaka	Sakagami	Kirisima
得点	100	56	43	55	78

名前	Yamaguchi	Yuba	Shinoda	Nagata	Akiyama
得点	92	85	64	67	92



# オブジェクト指向という考え方

## 課題の解決方針



- ソート関数を作る
- 結合用の関数を作る
- 関数の引数を考える



手続き型の思考

- 集合を表すリストを作る
- リストを結合する機能を付ける



オブジェクト思考



# リストを表すプログラムを作ろう

- 上手なプログラムの秘訣

いろいろ新しいものを作らない！



元からあるパーツで代用できないかをまず考える

- みなさんの欲しいプログラムの基本は大抵用意されています



# リストを管理するためのクラス

## ■ クラスって何？

- 皆さんの知っている言葉で言えば、「型」に近いです
- クラスは特定の要素の形式を示します

## ■ intという型

- 整数を保持できる
- 四則演算が適用できる

← ほぼ同じ  
ニュアンス

## ■ ArrayListクラス

- 可変サイズのリストを保持できる
- 中に入る要素のクラスを決められる
- 並び替えやその他の機能を持つ



# ArrayListクラスを使ってみよう

- 新たなプロジェクトを作しましょう
  - 名前は任意です
  - プログラムはmain関数の中に書きます
- 作成するプログラムのポイント
  - リストの定義と初期化

```
ArrayList mylist = new ArrayList();
```

- リストへの要素の追加

```
mylist.add(10);
```

- リストの要素を取出して表示

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





# リストを使ったプログラム

```
public static void main(String args[]){  
    ArrayList mylist = new ArrayList();  
    mylist.add(10);  
    mylist.add(30);  
    mylist.add(28);  
  
    System.out.println(mylist.get(0));  
    System.out.println(mylist.get(1));  
    System.out.println(mylist.get(2));  
}
```

- 1行目がエラーになります
- 「インポートに追加」が必要です



# クラスの初期化とインスタンス

- クラスは初期化(=メモリの確保)してから使います
- 初期化には new 演算子を使います
- メモリを確保した変数をインスタンスと呼びます
- 変数定義と同時に初期化することが一般的です

`ArrayList mylist;` ← 変数の定義

この段階ではメモリ  
は確保されていない

`mylist = new ArrayList();`

これで初めてメモリが  
確保されます



# クラスは一つ、インスタンスは複数

- インスタンスが異なればメモリも異なります

```
public static void main(String args[]){  
    ArrayList mylist1 = new ArrayList();  
    ArrayList mylist2 = new ArrayList();  
    mylist1.add(10);  
    mylist2.add(30);  
  
    System.out.println(mylist1.get(0));  
}
```

- 何と表示されましたか？



# クラスのメソッド

- クラスはただデータ用のメモリを確保するだけではありません
  - 機能もセットになっています
  - クラスの持つ機能のことをメソッドといいます
  - 1つのクラスには多くのメソッドがあります
  - メソッドは  
「インスタンス名.メソッド名」  
で実行します

```
mylist1.add(10);
```

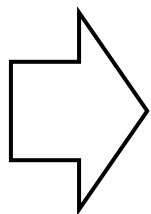
メソッドの名前

メソッドの引数



# どんなメソッドがあるのか？

- ArrayListには様々なメソッドがあります
  - add:要素の追加
  - get:要素の取得(取得しても削除はされない)
  - set:要素の書き換え
  - insert:要素の挿入
  - remove:指定した要素の削除
  - size:要素数の取得
  - contains:指定した要素があるかを確認



- 全部覚えるの??
- マニュアル(リファレンス)があります

# JavaDoc

## API辞典

Overview (Java Platform SE 8) x

docs.oracle.com/javase/jp/8/docs/api/index.html?java/util/ArrayList.html

Java(tm) Platform Standard Edition 8

概要 パッケージ クラス 使用 階層ツリー 非推奨 索引 ヘルプ

前次 フレーム フレームなし

### Java(tm) Platform, Standard Edition 8 API仕様

このドキュメントはJava(tm) Platform, Standard EditionのAPI仕様です。

参照: 説明

#### プロファイル

- compact1
- compact2
- compact3

#### パッケージ

パッケージ	説明
java.applet	アプレットを作成するために必要なクラス、およびアプレットがそのアプレット・コンテキストとのやり取りに使用するクラスを提供します。
java.awt	ユーザー・インタフェースの作成およびグラフィックスとイメージのペイント用のすべてのクラスを含みます。
java.awt.color	カラー・スペースのクラスを提供します。
java.awt.datatransfer	アプリケーション間またはアプリケーション内のデータ転送のためのインタフェースとクラスを提供します。

#### ArrayList

```

public class ArrayList<E>
    extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, Serializable

List インタフェースのサイズ変更可能な配列の実装です。リストのオプションの操作をすべて実装し、null を含むすべての要素を許容します。この
は、List インタフェースを実装するほか、リストを格納するために内部的に使われる配列のサイズを操作するメソッドを提供します。(このクラスは
期化されないことを除いて Vector とほぼ同等。)
```

size、isEmpty、get、set、iterator、および ListIterator の処理は、一定の時間で実行されます。add の処理も、償却定数時間で実行されます。つ
n 個の要素を追加するには O(n) 時間が必要です。ほとんどの場合、ほかのすべての処理も比例的な時間で実行されます。定数の係数は、LinkedList
の場合より小さくなります。

## Javaの言語解説

# ライブラリのインポート

- ライブラリを利用するための最初の手順
  - 例: Socketクラス

クラス名

格納場所

概要 パッケージ クラス 使用 階層ツリー 非推奨 索引 ヘルプ

前のクラス 次のクラス フレーム フレームなし

サマリー: ネスト | フィールド | コンストラクタ | メソッド 詳細: フィールド | コンストラクタ

compact1, compact2, compact3

java.util

クラスArrayList<E>

java.lang.Object

java.util.AbstractCollection<E>

java.util.AbstractList<E>

java.util.ArrayList<E>

すべての実装されたインタフェース:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

直系の既知のサブクラス:

AttributeList, RoleList, RoleUnresolvedList

- ソースファイルの先頭にimport文を記述

```
import java.util.ArrayList;
```

ArrayListクラスのインポート

# クラスのインスタンス化

- インスタンス化
  - コンストラクタを呼び出す
  - コンストラクタは何種類か用意されていることがある
- コンストラクタに合わせた引数でインスタンス化

## コンストラクタのサマリー

### コンストラクタ

#### コンストラクタと説明

`ArrayList()`

初期容量10で空のリストを作成します。

`ArrayList(Collection<? extends E> c)`

指定されたコレクションの要素が含まれているリストを、要素がコ

`ArrayList(int initialCapacity)`

指定された初期容量で空のリストを作成します。

```
ArrayList list = new ArrayList();
```

空のリストを作成

```
ArrayList list2 = new ArrayList(list);
```

listを入れた新しいリスト作成



# メソッドの利用

- 呼び出し時は型に注意
  - 引数の有無
  - 戻り値の有無
- staticが付いたメソッド
  - 引数だけで使うメソッド
  - newしなくても使える
  - 詳細は来週でやります

すべてのメソッド	インスタンス・メソッド	具象メソッド
修飾子と型	メソッドと説明	
boolean	<b>add(E e)</b> このリストの最後に、指定された...	
void	<b>add(int index, E element)</b> このリスト内の指定された位置に...	
boolean	<b>addAll(Collection&lt;? extends ...&gt; c)</b> 指定されたコレクション内のすべての要素をこのリストに追加します。	
boolean	<b>addAll(int index, Collection&lt;? extends ...&gt; c)</b> 指定されたコレクション内のすべての要素をこのリストの指定された位置に追加します。	
void	<b>clear()</b> このリストからすべての要素を削除します。	

```
ArrayList list = new ArrayList();
```

```
list.add(10);
```

ソケットのバインド

```
list.clear();
```

ポート番号の取得

# メソッドの表し方

## Stringクラスのメソッド

文字列中に、ある文字列が出てくる位置を返すメソッド

**int indexOf(String str, int fromIndex)**

パラメータ:

str - 検索対象の部分文字列

fromIndex - 検索開始位置のインデックス

戻り値:

指定されたインデックスから検索を開始して、最初に指定された部分文字列が出現する、この文字列内のインデックス

戻り値はint型で、  
引数一つ目はString型  
引数二つ目はint型

説明上の変数名であり  
名前に意味はない  
重要なのは型だけ

```
String str1 = "hello world! Hello!";
```

```
int index = str1.indexOf("llo",6);
```

6文字目以降で最初に出現する  
"llo"のインデックスを取得



# リストには何でも入る？

## ■ 次のプログラムを試してみましょう

```
ArrayList mylist = new ArrayList();  
mylist.add(10);  
mylist.add(3.14);  
mylist.add("This is a pen");  
  
System.out.println(mylist.get(0));  
System.out.println(mylist.get(1));  
System.out.println(mylist.get(2));
```

■ 何と表示されましたか？

■ これはさすがに気持ち悪い...



# 要素の種類を固定したリスト

- 次のプログラムを作しましょう
  - 一部エラーになります

```
ArrayList<Integer> mylist = new ArrayList<Integer>();  
mylist.add(10);  
mylist.add(3.14);  
mylist.add("This is a pen");  
  
System.out.println(mylist.get(0));
```

- <>の中で整数専用のリストであることを示します
- Stringと書けば文字列、Doubleと書けば少数です



# 要素の種類を固定するメリット

- forループ等で値を順に参照できる
  - 特定の型の要素として取り出せる

```
ArrayList<Integer> mylist = new ArrayList<Integer>();  
mylist.add(10);  
mylist.add(3);  
mylist.add(234);  
for(int i:mylist){  
    System.out.println(i);  
}
```

← Integerのリストは  
int型に順に代入可能



# クラスを使うとなぜ便利1

- 欲しい機能があれば、わざわざ**プログラムを書かなくてもよい**
- ArrayListクラス
  - 結合の機能を持っている
  - 複数のリストをつなげることができる

```
ArrayList<Integer> mylist1 = new ArrayList<Integer>();  
ArrayList<Integer> mylist2 = new ArrayList<Integer>();  
mylist1.add(10);  
mylist2.add(3);  
mylist2.addAll(mylist1);
```

⇐ リスト結合用のメソッド



# クラスを使うとなぜ便利2

- 欲しい機能があれば、わざわざ**プログラム**  
**を書かなくてもよい**
- Collectionsクラス
  - ソート機能も持っている
  - ArrayListの並び替えはたった1行

```
ArrayList<Integer> mylist = new ArrayList<Integer>();  
mylist.add(10);  
mylist.add(3);  
mylist.add(234);  
Collections.sort(mylist);    ⇐ ソート用のメソッド
```



# Javaでコードを書くときは・・・

- やりたいことをまず考えます
  - ネットで検索してもよい
  - クラス名を知るのが目的
    - ソースコードのコピーは目的ではない
- 目的に合わせたクラスをリファレンスで調査
  - 使い方がわかれば大丈夫です
- 目的に合わせたクラスがなかったら・・・
  - ここで初めてプログラミングが必要です
  - 今日は使ってみるだけにします





# 以上で説明は終わりです

---

- 課題に取り組んでください
- 基本課題ができたなら結果を見せてください
  - 基本課題のソースコード(プロジェクト)は、アップロードして提出してください
  - 提出できているかを確認します
- 基本課題が終わればOKです
  - 時間が余る人は発展課題もやりましょう
  - 発展課題のヒントは自分で読んでください



# 第1回追加資料

---

# よく使うもう一つのクラス

- マップ
  - HashMapクラス
  - key-Value テーブルを保存できる
  - あるキーワードで値を引くことができる表
- マップの例：名簿

key	学籍番号	氏名	value
	2014E01	天野 晴香	
	2014E02	伊藤 裕也	
	2014E03	内田 真	
	2014E04	遠藤 太郎	
	2014E05	小野寺 幸一	
	2014E06	加藤 真一	



# HashMapを使ってみよう

```
HashMap<String,String> mymap;  
mymap = new HashMap<String,String>();  
mymap.put("2014E01", "天野 晴香");  
mymap.put("2014E02","伊藤 裕也");  
  
System.out.println(mymap.get("2014E02"));
```

- HashMapのよく使うメソッド
  - put:値の挿入(関連づけ)
  - get:値の取得
  - containsKey:そのキーを持つ値があるかを調べる



# HashMapの注意

- Keyが重複するとどうなるか？
  - Valueが上書きされます
- 同じkeyに複数の値を入れるにはどうしたらいいの??
  - ArrayListをValueに入れば大丈夫
  - 次のような構造のMapも作れます

```
HashMap<String,ArrayList<String>> mymap;  
mymap = new HashMap<String,ArrayList<String>>();
```



# 任意課題のヒント1

- こんなMapの構造を想定しています

```
HashMap<String,Integer> Tokuten;  
Tokuten = new HashMap<String, Integer>();  
Tokuten.put("Sato",100);  
...
```



## 参考資料

---

今回の授業内容とは直接関係ありませんが、知っているのと役に立つかもしれない情報をまとめました



# 家で環境を作る

- 実験室のIDEと最新版のIDEにはずいぶん差ができてしまいました
  - 実験のための安定した環境を作るには次のソフトウェアを入れれば大丈夫です
    - Java SDK(開発用パッケージ)
      - JDK 1.8 (Java SE 8u241 64bit版)
      - JDK 14 (Java SE 14 最新版)
- ※どちらもオラクルのアカウントが必要
- <https://www.oracle.com/jp/java/technologies/javase/documentation.html>
- Apache NetBeans 12.1
    - <http://netbeans.apache.org/>





# 開発環境の利点と欠点

- テキストエディタとコマンドプロンプトでの作業
  - コンパイラのエラーを直接確認できる
  - 言語を理解する上での用語を一つずつ学べる
  - 中・大規模なプログラミングには不向き
- 開発環境を使うと・・・
  - 多数のソースコードの管理
  - 複雑なインポート宣言の編成
  - デバッグ
  - 外部ライブラリの参照
  - JavaDocやJarファイルとしてエクスポート

等の処理が容易に行える



# コマンドプロンプトを使った実行

- この方法は今回は行いません
  - 以下のコマンドを使えばプロンプトからもコンパイルと実行が可能です
  - コンパイル

```
>javac コンパイルしたいソースファイル.java
```

- 実行

```
>java コンパイルしたクラス名
```

# switch-case文の構造

```
int number = 2;
switch(number){
    case 1:
        System.out.println("優勝です");
        break;
    case 2:
        System.out.println("準優勝です");
        break;
    default:
        System.out.println("順位外です");
}
```

**switch文**による分岐

numberが**1**の場合の実行開始位置

**break文**による**ブロック**のからの脱出

numberが**2**の場合の実行開始位置

全**case**にあてはらない場合 **default**

- switch文はプリミティブ型の値でしか分岐できない
  - 文字列による条件分岐などは不可能
  - 次回のバージョンアップで対応を検討されている
- breakをしない場合次のCase内の内容も実行される
  - Cと同様だが注意が必要



# オブジェクトとプリミティブ

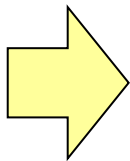
- プリミティブの許容
  - Javaのオブジェクト指向の不完全性
  - C言語ユーザにとっては比較的使いやすい

```
int Number;
```

```
Number = 1 + 2 + 3 + 5;
```

- int型の整数はオブジェクトではない！

new演算子で初期化しない



メソッドを持たない

引数にオブジェクトをとるメソッドには入れられない



# プリミティブな変数の種類

## ■ プリミティブの種類

boolean	真偽値	True, false
byte	整数(8bit)	-128 ~ 127
short	整数(16bit)	$-2^{15} \sim 2^{15}-1$
int	整数(32bit)	$-2^{31} \sim 2^{31}-1$
long	整数(64bit)	$-2^{63} \sim 2^{63}-1$
float	単精度浮動小数点	32bitの浮動小数点
double	倍精度浮動小数点	64bitの浮動小数点
char	文字	'a', 'い', '漢' などの文字

# プリミティブとオブジェクトの対応

- プリミティブ型に対応するオブジェクト型
  - オブジェクト型しか引数に取れない場合への対応
    - List, Map, Iterator など
  - プリミティブ型の補助的な要素
    - 最大値や最小値の保持
  - プリミティブ型とオブジェクト型の仲介
    - 数字と文字列との相互変換

プリミティブ型	オブジェクト型
boolean	Boolean
byte	Byte
short	Short
int	Integer

プリミティブ型	オブジェクト型
long	Long
float	Float
double	Double
char	Character



# Integerクラスの利用例

```
List memberList = new List();
```

```
Integer obj = new Integer(01258036);
```

```
memberList.add(obj);
```

Integerクラスのオブジェクトを定義

Listオブジェクトの要素にobjを追加

```
int todaysHiscore = Integer.MIN_VALUE;
```

int型の最小値を変数に設定

```
String userInputValue = "12345";
```

```
int input = Integer.parseInt(userInputValue);
```

Stringオブジェクトを  
int型に変換

```
String userInputValue2 = "1A5";
```

```
int input2 = Integer.parseInt(userInputValue2,16);
```

int型に基底16で変換

```
int value = 128;
```

```
String hexString = Integer.toString(value,16);
```

16進数表現の文字列に変換

# オブジェクト型の欠点

- **new**演算子により生成する
- **メソッド**により値を変更する
- 演算子による演算は出来ない

```
Integer a = new Integer(2);
```

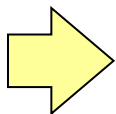
```
Integer b = new Integer(5);
```

```
Integer sum = new Integer();
```

```
sum.setValue(a.getValue() + b.getValue());
```

getValueメソッドで値を取り出し  
setValueメソッドで値を代入する

- **new** するのを忘れると**メソッド**は使えない
- 演算子で直接つなぐと書式エラー



Java5.0 以降は**オートボックス機能**により面倒な処理が不要に！



# オートボクシングとオートアンボクシング

- プリミティブとオブジェクトの変換を自動で行う機能
- クラスによる動的な配列などを利用する際に便利
  - 動的な配列については来週詳しく説明する

```
Integer a = 2;
```

Integerクラスにオートボクシング

```
Integer b = new Integer(5);
```

```
int sum = a + b;
```

int型にアンボクシング

```
Integer value = a + b;
```

オートアンボクシングされ計算  
その後オートボクシングされ代入

# 特殊なオブジェクト: 配列

- プリミティブまたはオブジェクトの集合
- Javaではオブジェクトとして扱われる

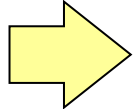
`int[] numbers = {1,2,3,4,5};`      `int`型の要素をもつ静的な配列の定義

`double[] average = new double[30];`      `new`演算子による配列の初期化  
`average[0] = 30.5;`      添え字を指定して値を代入  
`average[1] = 68.2;`

`String[] names = new String[3];`      `String`オブジェクトを格納する配列の定義  
`names[0] = new String("Taro");`  
`names[1] = "Hanako";`

`int count = numbers.length;`      メンバ変数「`length`」の取得  
`double value = average[0];`      添え字を指定して値を取得  
`int len = names.length;`

# 特殊なオブジェクト:String

- 文字列 (Stringクラス) は利用頻度が非常に高い
  - newで毎回生成
  - concatで文字列を接続 実用上不便
- Stringクラスは ” (ダブルクォート) でインスタンス化可能
- Stringクラスは + 演算子 で連結可能

```
String str1 = "Hello ";
```

```
String str2 = "World.";
```

```
System.out.println(str1 + str2);
```

文字列をつなげて出力

```
int place = 1000;
```

```
System.out.println("値段は" + place + "円です");
```

intやdoubleも文字列として  
連結できる