プログラミング言語

名倉 正剛
日本大学 工学部 情報工学科
70号館7044号室

プログラミング言語　第2回

- プログラミング言語の歴史と体系

自然言語 vs. プログラミング言語

- 自然言語
  - 人間が，互いのコミュニケーションを図るために，自然に獲得した言語
  - 人間同士の相互理解に利用，文法規則が複雑，意味が文脈（コンテキスト）によって定まる場合がある
- プログラミング言語
  - コンピュータが実行すべき処理の内容を，人間が，正確に，曖昧なく指令するための人工言語
  - コンピュータの実行命令に正確に変換できる必要があるため，文法的な正確さを重視.
    - Syntax（統語論：文法）と Semantics（意味論）
  - 文脈によって意味が変化しないように設計されている.
  - 利用分野，処理内容に応じて，多種多様な言語が開発されてきた

プログラミング言語の例

- 利用分野，処理内容に応じて，多種多様な言語が開発されてきた
  - Fortran
  - COBOL
  - Pascal
  - Smalltalk
  - C / C++ / C#
  - Java
  - Lisp
  - Perl / Ruby / Python

プログラミング言語の歴史（前史）

- 最初のプログラム
  - Charles Babbage (1791～1871 : 英) の解析機関 (analytical engine : 歯車式の自動計算機)
    - 都度入力の手間を減らすため，パンチカードによる計算手順の入力
    - 入力装置，出力装置，演算装置，記憶装置から成り立ち，今日のコンピュータとほとんど同じ構成
  - Ada Lovelace (1815～1852 : 英) によるバベッジ解析機関のためのプログラム
    - 条件分岐，サブルーチンがすでに考えられていた

世界最初の汎用電子的コンピュータ

- ENIAC（1946年，米）
  - 第二次大戦中から，各国でコンピュータ開発が行われる（弾道計算用，暗号解読用．．．）
  - 暗号解読や，方程式計算などに特化したコンピュータは，大戦中に存在
  - 世界初の任意のプログラムを実行することができるコンピュータ(チューリング完全)
  - 配線を切り替えることにより，計算手順を指示した（これがプログラム）
- EDVAC（1949年，米）
  - プログラム内蔵方式の電子計算機（今と同じ）
  - アセンブラのコードをメモリにロードして実行できた．

チューリング完全

- （正確な定義ではないが）実現可能なアルゴリズムや手続きを全て処理できる性質のこと
  - プログラムにできる全てのことが実現可能（※注）
  - たとえば HTML や CSS や SQL は，反復構造を記述できない＝チューリング完全ではない

※ 計算時間や，利用するリソース量（メモリ量など）を問わなければ，実現可能

  - 純粋に，プログラム言語として，できる／できない，ということがない

機械語／アセンブリ言語

- 機械語
  - コンピュータの基本動作を1つ1つ指示
    - レジスタ／メモリ間のデータ移動
    - 演算命令，分岐等のプログラム制御
  - 命令，演算対象を，数値コードで識別（バイナリコード）

- アセンブリコード
  - （機械語だと人間が書けないので）機械語と 1対 1 に対応した「ニーモニック」としてオペコード（演算子）を表現
    - add，sub, mul, div, mov など.
  - 演算回路（CPU に用意された命令セット）に対応してニーモニックが用意される.
    - x86 用アセンブラ，PowerPC 用アセンブラ，ARM アーキテクチャ用アセンブラ. . . .

アセンブリ言語のプログラム例 (hello.asm)

- x86 アセンブラで記述した，“Hello, World!!” プログラム
- 機械語命令に1:1対応．
  - 長い．．．．
- レジスタに命令をセットして実行
  - eax：システムコール番号
  - 引数は，逆順にスタックに Push 最後にシステムコール番号も Push
- レジスタに命令をセットしたら，割り込み0x80を発生（システムコール実行）

```
section .data       ; データセクションの定義
msg:    db   "Hello, world!!", 0x0A
len:    equ  $-msg ; msg とのアドレス差分を定義

section .text       ; 機械語セクションの定義
global _main        ; エントリーポイントの指定

_main:
 push   dword len   ; 文字列の長さを指定
 lea    eax, [msg] ; 文字列の場所を取得
 push   dword eax   ; 取得した値を指定
 push   dword 1     ; 標準出力を指定
 mov    eax,  4     ; 出力のシステムコールを指定
 int    0x80        ; システムコール実行

 push   dword 0     ; 正常終了の 0 を指定
 mov    eax,  1     ; 出力のシステムコールを指定
 int    0x80        ; システムコール実行
```

前のスライドのプログラムの命令セット

- 演習室の MacOSX から，次のコマンドで実行可能

```
% nasm -f macho hello.asm
% gcc -m32 -mmacosx-version-min=10.5 -o hello hello.o
% ./hello
```

- 命令セットは次の通り:

```
db:   オペコードに続くデータ（オペランド）で初期化する.
equ:  オペランドの値を定義する（C の #define と同等）
push:第一オペランドで指定されたビット長のデータ（第二オペランドで指定）を，スタックにPush.
      （dword は，4 バイト＝32ビット）
lea:  第二オペランドのアドレスを計算し，第一オペランドに代入する.
mov:  第二オペランドを，第一オペランドにコピーする.
int:  ソフトウェア割り込み（命令実行中の CPU に対して，発生する割り込み）を生成する.
      Unix では，割り込み番号 0x80（=128）によって，OS カーネルに割り込みを指定する.

他にも，前述のように，add(加算)，sub(減算)，mul(乗算)，div(除算)，jmp(Jump)，
jz（Jump on Zero），jnz（Jump on Non-Zero）などなど. . . .
```

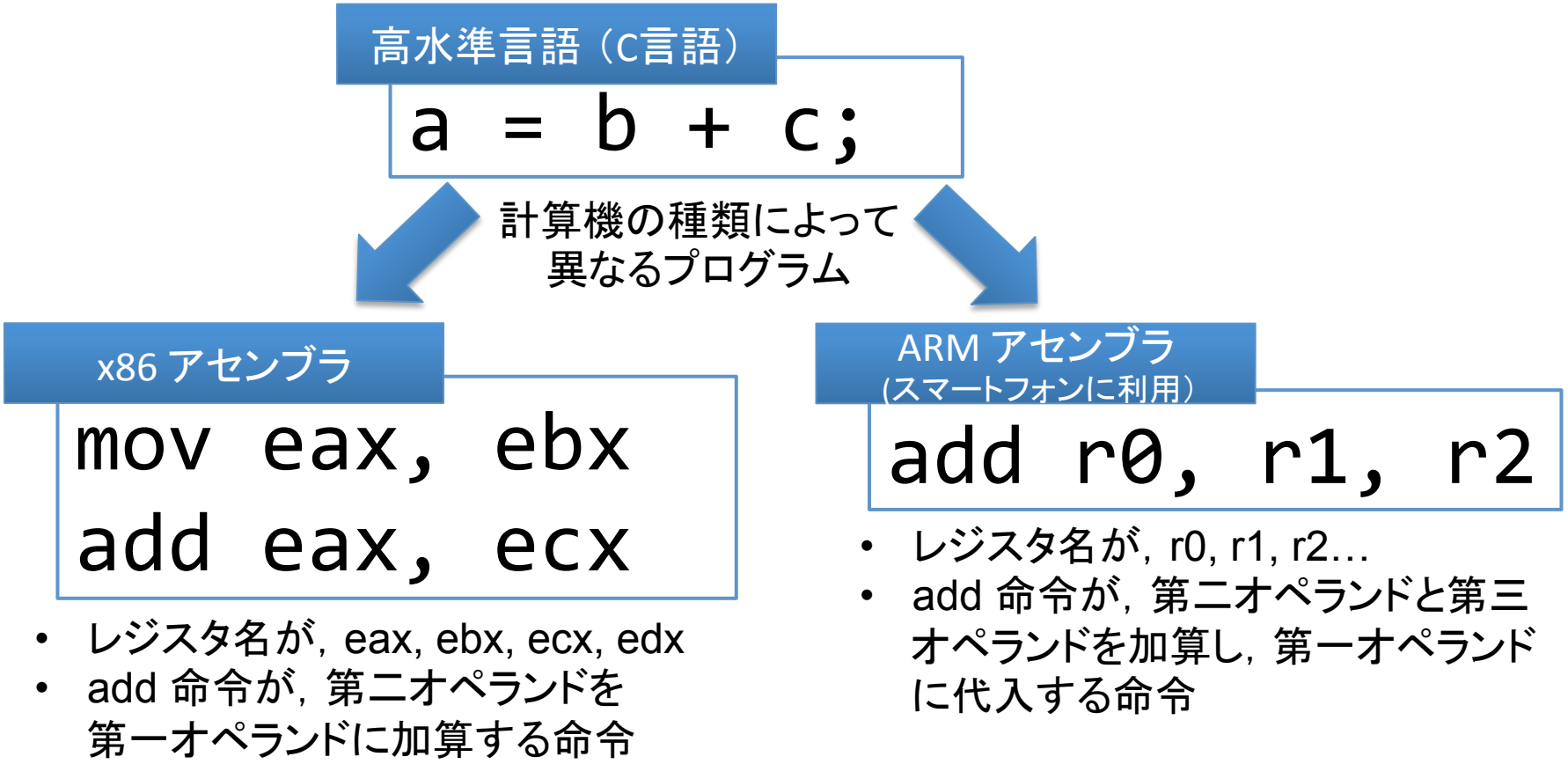
機械語／アセンブラプログラムの問題点

- 文法が，計算機の種類（命令セットの種類）により異なる
  - 前のスライドは，intel x86 命令.
- 命令セットごとにニーモニックを指定するため，
  - 簡単な処理でも，コード量が多くなる.
  - 記述が複雑になる.
  - デバッグが大変になる.

→ 高水準言語の出現
（アセンブラまでは，低水準言語）

低水準言語の問題

- 単純に加算をするプログラムを書いてみる.



高水準言語

- 機械語と1:1に対応しない，より人間の理解しやすい抽象的な言語
- 計算機の機種（プロセッサ）に依存しない，抽象的な計算手続きを記述可能.
  - メモリ制御，IO制御 等の低水準の操作も意識しなくてよい.
- 機械語に変換するための，「コンパイル」が必要になる
- 初期の高水準言語：Fortran，COBOL など.

FORTRAN

- 初の高水準言語 (1954, IBM *John Backus*)
- 名前は,「**for**mula **tran**slation（数式を機械語へ翻訳）」に由来
- 科学技術計算分野で,
  現在も利用される.
  - 科学技術計算分野向けに作られていて, たくさんの解析ソフトウェアが存在.
  - 言語構造的に最適化が行いやすい.

1から100までの総和を求めるプログラム

```
PROGRAM EX1
INTEGER I,SUM
SUM=0
DO I=1,100
SUM=SUM+I
END DO
WRITE(*, '(I5)') SUM
STOP
END
```

COBOL

- 事務処理用言語 (1959，CODASYL という，米国標準化団体)
- 名前は，「**CO**mmon **B**usiness **O**riented **L**anguage（共通事務処理用言語）」に由来
  - 部，節，段落，文という階層で記述．
  - 金額計算などに，今でも利用される．

1から100までの総和を求めるプログラム

```
IDENTIFICATION    DIVISION.
PROGRAM-ID.       EX1.
DATA              DIVISION.
WORKING-STORAGE   SECTION.
01 CNT   PIC 9(3) VALUE 0.
01 SUM   PIC 9(5) VALUE 0.
PROCEDURE         DIVISION.
    PERFORM 100 TIMES
        ADD 1 TO CNT
        ADD CNT TO SUM
    END-PERFORM
    DISPLAY SUM
    STOP RUN.
```

BASIC

- 初心者向けコンピュータ言語 (1964，米ダートマス大学 John *John Kemeny，Thomas Kurtz*)

- 「**b**eginner‘s **a**ll-purpose **s**ymbolic **i**nstruction **c**ode（初心者向け汎用記号命令コード）」の略
  - 行番号の概念の導入.
  - GOTO 文と行番号を利用し，プログラム上のどこへでもジャンプ可能.
    (アセンブラのジャンプ命令と同様の機能）
  - 1980 年代に，多くのパソコンに搭載されて普及.
  - 現在でも利用されることが多い.
    （Visual Basic)

1から100までの
総和を求めるプログラム

```
10 SUM=0
20 FOR I=1 TO 100
30 SUM=SUM+I
40 NEXT
50 PRINT SUM
60 END
```

スパゲッティプログラムの問題

- アセンブラでは，ジャンプ命令によってプログラムを制御.
  - 反復処理も，ジャンプ命令によって実現
- 初期の高水準言語でも，ジャンプ命令を用意.
  - プログラム上の任意の場所に処理が遷移する（BASIC の GOTO 文など）
- スパゲッティプログラムの問題が起こる
  - プログラムの流れがわかりにくくなる（可読性が低い）.
- 繰り返しや，分岐などの「構造化定理」に即した文法を提供する「構造化プログラミング」へ.

C 言語

- UNIXのシステム記述用言語 (1972，米AT&Tベル研究所 *Brian Kernighan*，*Dennis Ritchie*：K&R という呼び名が有名)
- 構造化されたプログラムを記述できる.
  - 入れ子構造で手順を記述できる.
  - goto 文を利用しなくても記述可能な言語構造.
  - スコープの概念を持つ.
- 採用ソフトウェア分野が広い.
- 派生言語も多い.
  - C++，C#，Java，Objective-C
- 年ごとに進化して標準規格化
  - K&R，C89/C90，C99，C11
  - gcc は，C99 にほぼ対応
    例）変数宣言が，ブロックの先頭でなくても良くなった.

1から100までの総和を求めるプログラム

```
#include <stdio.h>

int main(void){
  int i, sum = 0;
  for (i = 1; i <= 100; i++)
    sum += i;
  prinntf("%d\n", sum);
}
```

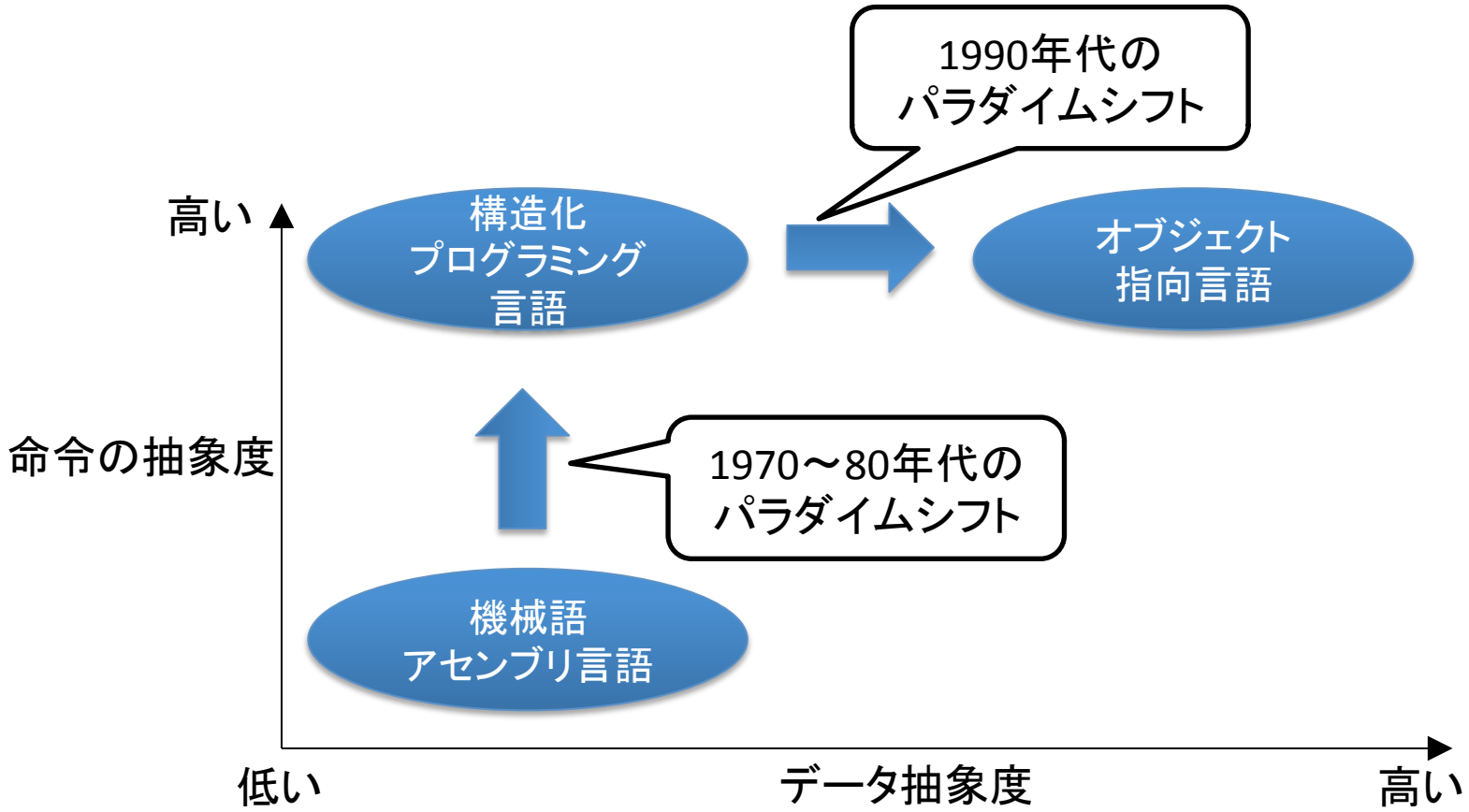
ソフトウェア危機（Software Crisis）

- 1968年にに開催された NATO ソフトウェア工学会議での議論によって生み出された用語.
- ソフトウェアの複雑化，大規模化.
  - ソフトウェア開発コストが上昇
  - プログラムをうまく管理できない

- ソフトウェア危機への対応：
  - ソフトウェア開発工程の体系化（ソフトウェア工学）
  - ソフトウェアの再利用，部品化を意識した仕組みの開発

プログラミング言語の抽象度

- 手続きとデータをまとめて部品化することで，ソフトウェアの再利用，部品化を容易にする．



オブジェクト指向

- データ＋手続き＝オブジェクト.
  - カプセル化
    - 必要なものは一箇所にまとめる.
    - 外部から不要な操作ができないようにデータの隠蔽.
  - 独立性の高い部品化.
- 継承の仕組み
  - 既存の部品を拡張した新しい部品を作成.

- 代表的な言語
  - Smalltalk（1972 年）
  - C++（1979 年），Java（1991 年）

C++ 言語

- UNIXのシステム記述用言語 (1983，米AT&Tベル研究所 *Bjarne Stroustrup*)
- C 言語にオブジェクト指向の考えを追加したもの
- C の既存機能を拡張，文法的に互換性を持っていた.
  - しかし，C++制定後に標準化された C の機能 (C99など) については，互換性がない
- オブジェクト指向でない書き方も許容されるため，言語仕様が複雑になった
  - オブジェクト指向を強化した Java へ.

C++ 言語 (cont'd)

1から100までの総和を求めるプログラム

```
#include <iostream>

class CalcSum {
    int sum;
  public:
    CalcSum(int start, int end){
      sum = 0;
      for (int i = start; i <= end; i++)
        sum += i;
    }
    int getSum() { return sum; }
};
int main(void){
  CalcSum sum(1, 100);
  std::cout << sum.getSum() <<"\n";
}
```

クラス定義

可視性を指定する

コンストラクタ

main 関数部分は，C言語と共通

インスタンスを生成して呼び出す

Java 言語

- オブジェクト指向プログラミングの考え方に基づいて設計された言語(1991，米サン・マイクロシステムズ *James Gosling*)
  - 完全にクラスベースなオブジェクト指向プログラミング
  - 高水準言語（C など）では，機種間でのプログラムの互換性はあったが，機械語プログラムに変換すると互換性がなかった
    → 機種間でバイナリ互換性を保つために，仮想マシン（VM）上で動作する.
- C/C++ の文法から多くを引き継いでいる.
  - 文法的には，ポインタはない.
  - しかし，オブジェクトインスタンスの代入は，参照により行われる.

```
List list1 = new ArrayList();
list1.add("A");
list1.add("B");
List list2 = list1;
list2.clear();
if (list1.isEmpty())
  System.out.println("empty");
```

真になる
（empty が表示）

Java言語 (cont'd)

1から100までの総和を求めるプログラム

```
class CalcSum {
  private int sum;
  public CalcSum(int start, int end){
    sum = 0;
    for (int i = start; i <= end; i++)
      sum += i;
  }
  int getSum() { return sum; }

  public static void main(String args[]){
    CulcSum sum = new CulcSum(1, 100);
    System.out.println(sum.getSum());
  }
}
```

クラス定義

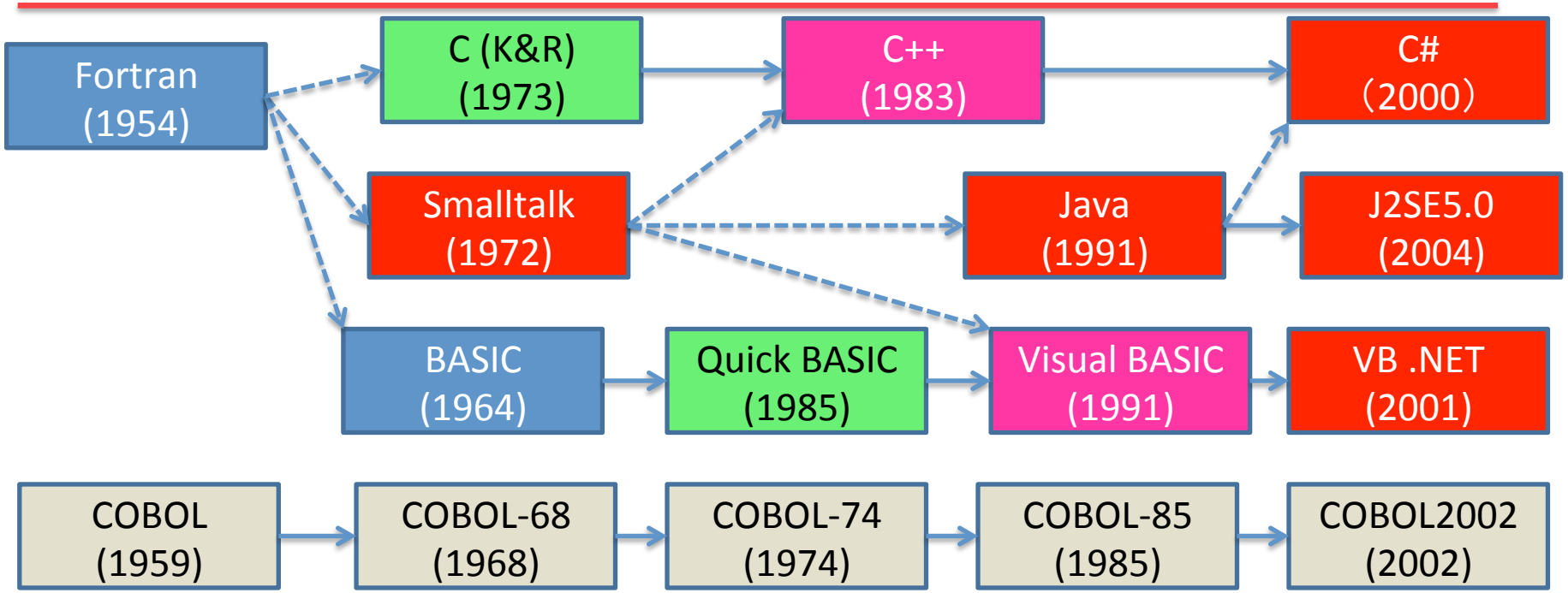
可視性を指定する

コンストラクタ

main 関数部分も，
クラス内に記述

インスタンスを
生成して呼び出す

高水準言語の系譜



- 枠の凡例
  青：手続き言語　緑：構造化言語
  ピンク：部分的なOO言語　赤：OO言語
  灰色：事務処理向け（COBOL）の系譜
- 線の凡例
  実線：直接の後継　破線：影響を受け，同等機能を取り込み