目次



# ★1　Java とは

注) この資料中に説明なくページ数が出てきた場合，教科書のページ数を表します．
この資料は，講義中の板書や説明を全部ここに書き込めるほどスペースが広くないかもしれません．「ソースの解説なんかは書き込み式の方がわかりやすいやろ」という思いと「全部書き込み式にするのはいくらなんでも大学の授業としてどうなんや（小学校じゃあるまいし）」という思いが交錯して，中途半端な作りになることがしばしばで…．

Java は，次のような特徴をもったプログラミング言語である．

1.

2.

3.

今回の授業では，1. について考える．2. や 3. については，この授業全体を通して学んでいくことになる．

Java の特徴については，知っておくべき大事なことが教科書第 1 章にいろいろ記されている．読んでおくこと (☆ 1)．

☆ 1) この科目の FAQ のページにも情報がある．自分の PC で Java プログラミングできるようにする方法とか．

## ★1.1　とりあえず Java プログラミング

### その前に C 言語プログラミングの例

**G01Hello.c**

```
 1  /* C 言語のプログラムの例 */
 2
 3  #include <stdio.h>
 4
 5  int main(void){
 6    printf("こんちは\n");
 7    printf("a01055 の");
 8    printf("ほげほげおでおま\n");
 9    return 0;
10  }
```

上記のソースファイル G01Hello.c がカレントディレクトリにある場合，計算機室の Linux 環境でこれをコンパイルするには次のようにすればよい．

```
$ cc G01Hello.c
```

このようにコマンド cc を実行すると，G01Hello.c がコンパイルされ，a.out というオブジェクトファイルが作られる．a.out を実行するには次のようにすればよい．

```
$ ./a.out
```

実行結果は次のようになる．

```
こんちは
a01055 のほげほげおでおま
```

### はじめての Java プログラミング

**G01Hello.java**

```
 1  /** はじめての Java プログラム */
 2
 3  public class G01Hello{
 4
 5      public static void main(String[] args){
 6
 7          System.out.println("こんちは");
 8          System.out.print("a01055 の");
 9          System.out.println("ほげほげおでおま");
10
11      }
12
13  }
```

注意: 「クラス」,「メソッド」とは何か，3,5 行目はどういう意味か，等は今後説明します．(☆ 2)

☆ 2) コメントが /** と星 2 つで始まってることには意味がありますが，授業では説明しません．気になる人は教科書を参照してね．

上記の**ソースファイル** G01Hello.java がカレントディレクトリにある場合，計算機室の Linux 環境でこれを**コンパイル**するには次のようにすればよい．

```
$ javac G01Hello.java
```

すると，G01Hello.java がコンパイルされ，**クラスファイル**が作られる．ls コマンドで確認してみよう．

```
$ ls
G01Hello.class  G01Hello.java
```

このクラスファイルを**実行**するには，次のようにすればよい．

```
$ java G01Hello
```

## ★ 1.2 コンパイラ，インタプリタ，仮想マシン

コンピュータのハードウェアや OS(☆ 3) などを合わせた，プログラムが動作する環境のことを，**プラットフォーム**という．ハードウェアの違い (☆ 4)，CPU アーキテクチャの違い，OS の違いなどによって様々なものがある．

☆ 3) OS: オペレーティングシステム

☆ 4) PC，携帯電話，スーパーコンピュータ，etc.

プラットフォームが異なれば実行可能な機械語プログラムの形式も異なるので，機械語プログラムはプラットフォーム毎に用意しなければならない．いわゆる高級言語プログラミングでは，読みにくい機械語プログラムをプラットフォーム毎にいくつも作る，という作業から人間を解放するために，読みやすくかつプラットフォームに（あまり）依存しない**ソースコード**を作成してから機械語プログラムに翻訳する，という手順を踏む．

そのやり方としては，次の二つが代表的である (☆ 5)．

(1) **コンパイラ**を用いる:ソースをまとめて機械語に翻訳する．機械語を直接実行するので，インタプリタを用いるより実行が速い．
(2) **インタプリタ**を用いる: 実行時に逐次的にソースを機械語に翻訳していく．翻訳しながら実行するので遅い．

☆ 5) C 言語ではコンパイラを用いることが一般的．一方，Perl や Ruby のようなスクリプト言語は，インタプリタ型言語である．

開発者が作成したソフトウェアを，多様なプラットフォームを用いる多数のユーザに配布する，という状況を考えると，実行の速さでは (1) に軍配が上がるが，開発や保守の容易さでは (2) が有利である．なぜなら，(1) では開発者がプラットフォームに合わせて個別にコンパイル済みプログラムを用意しなければならないのに対して，(2) ではソースを配布するだけで済むからである (☆ 6)．

☆ 6) ただし，ユーザの方でその言語のインタプリタを自分の環境にインストールしておかねばならないという欠点がある．また，売り物だから等の理由でソースを公開しない，ということが難しいのも欠点となる．

これに対して，Java では，コンパイラとインタプリタの「いいとこどり」をして，次のような手順を採用している．

1. コンパイル時: ソースファイルを，プラットフォームに依存しない中間的な言語（**Java バイトコード**）のプログラムに翻訳する．翻訳してできるファイルを**クラスファイル**という．
2. 実行時: プラットフォーム毎に用意された**仮想マシン** (☆ 7) 上でバイトコードを実行する．

このようにすることで，プラットフォームに依存せずにプログラムを高速に実行できる．

☆ 7) JVM (Java Virtual Machine) と呼ばれる．JVM はインタプリタの一種と考えることもできる．
［発展］近年の Java 仮想マシンでは，プログラムの実行時に動的にバイトコードの一部を機械語にコンパイルする技術（Just In Time Compilation）を採用して性能向上をはかっている．