第1章 Java

1.1 Javaとは

1995 年、Sun Microsystems 社から公表されたプログラミング言語である。（Sun Microsystems 社は 2010 年に Oracle Corporation に吸収合併された。）Java の文法は、C 言語と似ているが、**互換性はない**。（一方、 C++ 言語はもともとは C 言語の拡張として設計された。）C++ と同様、＿＿＿＿＿＿＿＿言語 であるが、C++ に比べて**シンプルな**仕様になっている。 なお、＿＿＿＿＿＿ (ECMAScript) とは文法は似ているが、それ以外の関係はなく全く別の言語であるので注意する必要がある。

Q 1.1.1 次の文章のうち、正しいものに○、誤っているものに×をつけよ。

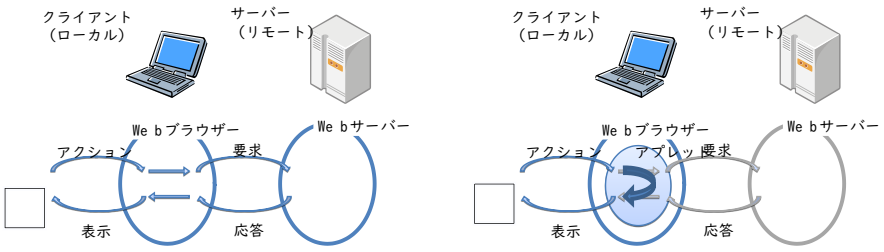
1. ☐ Java の文法は C 言語に似ており、Java のコンパイラーは C 言語のソースファイルをコンパイルすることも可能となっている。
2. ☐ Java はオブジェクト指向言語であるが、C++ 言語との互換性は持っていない。
3. ☐ Web ブラウザー上でインタプリター方式で実行する Java プログラムのことを JavaScript と呼ぶ。

1.2 Java の特徴

歴史から見た Java の特徴

Java は誕生当時は Web ページにアニメーションとインタラクティブ性をもたらすための仕組みである、アプレットのための言語として、世に広まった。

WWW の基本的な応答　　アプレットを使った場合の応答



HTML だけを用いて書かれた Web 文書の場合は、ユーザーがマウスをクリックするなどのアクションがあると、ブラウザーはそのアクションを遠隔地の WWW サーバーに伝え、その応答を待って新しい表示をする必要がある。

アプレット (applet) と呼ばれる Java のプログラムを HTML のなかで使っている場合は、アプレットがサーバーからブラウザーへダウンロードして実行される。するとユーザーのアクションに対して、ブラウザーの中で実行されているアプレットが即時に反応することができる。こうして、インタラクティブ性の高いページを記述することが可能になった。

Java の情報のページ
https://www.oracle.com/java/technologies/ （Java の本家）

このように、Java アプレットはネットワークを通じて別のコンピューターに移動して実行されることになる。このような使い方をするためには**安全性**と**可搬性**という特徴が重要になる。

安全性

これは、簡単にいえばアプレットを使って他人のコンピューターに悪戯をすることができない、ということである。もし、Web ページに任意のプログラムを埋め込んでブラウザー上で実行させることができれば、ハードディスク中のデータを消去してしまうなどのイタズラが簡単に行なえる。

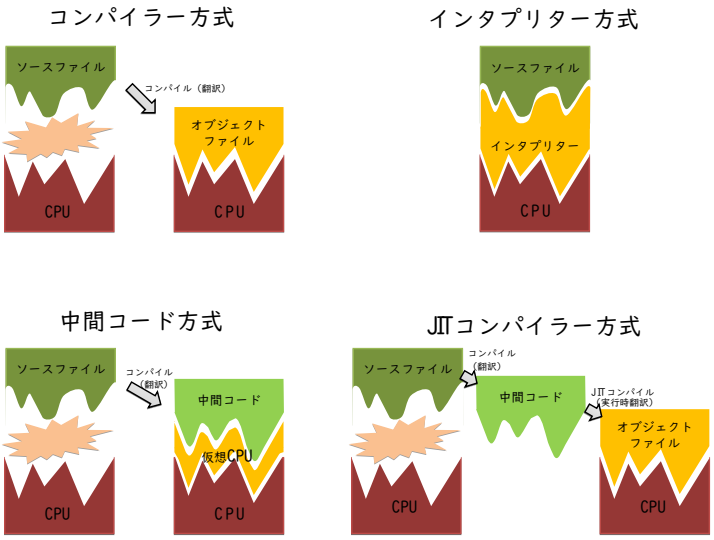
安全性を保障するためには、まずプログラムにファイル操作などをさせない、などの制限を課する必要があるが、C のような言語では、ポインター（アドレス）演算や無制限な型変換などの仕組みを通じて、悪意のあるプログラマーが抜け道を作ることができ、言語処理系の設計でこれを防ぐのは容易ではない。Java はポインター演算を明示的に提供せず、型変換（キャスト）をきちんとチェックするなど、このような明らかな安全性の抜け道がないよう設計された。このような設計は安全性だけではなく、プログラムのバグを未然に防ぐためにも有用である。

注: しかしながら 2017 年にリリースされた Java 9 から、Java アプレットをブラウザーで実行するための Java ブラウザープラグインが deprecated（非推奨）とされた。理論的には、Java の処理系や基本ライブラリーが適切に設計されていれば、Java アプレットは安全に実行できるはずなのだが、 JVM 自体や一部のライブラリーは Java ではなく C 言語などで実装する必要がある巨大なシステムであり、完璧に設計することは難しかった。結局、Java の処理系やライブラリーの脆弱性（不具合）をついた不正アプレットが多く現れ対処しきれなくなった。 こうして Java アプレットは、その役割を終えた。

可搬性

Web ページに埋め込まれるということは、さまざまな機種のコンピューターで実行される可能性があるということである。つまり、Java のアプレットに機種依存性があってはいけない。＿＿＿＿＿＿を用いる実行方式ではプログラムが機械語に翻訳されるため、機種依存性は避けられない。一方、＿＿＿＿＿＿を用いる方式では、各機種毎にインタプリターを実装するだけで良いが、効率が犠牲になる。このため、Java では＿＿＿＿＿＿という方法をとる。

Java のプログラムは Java コンパイラーによって＿＿(Java Virtual Machine, Java 仮想機械) という仮想 CPU のコードに翻訳される。この仮想コードを各 CPU 上の JVM エミュレーター（一種のインタプリター）が解釈・実行する。



この方法は Java プログラムを直接インタプリターで解釈・実行するよりは高速である。しかし、現在では JVM コードをより高速に実行するために **Just-In-Time** (JIT) コンパイラー というものを用いて、JVM コードを実行しながら各 CPU の機械語へ翻訳する、という方法を用いる。

また、グラフィックスやネットワーク、スレッドに関する標準ライブラリーを持つことも、それまでのプログラミング言語にはなかった重要な特徴である。

このように当初、Java はアプレットを作成するための言語として広まった。しかし、現在では、インタラクティブな Web ページを作成するためのブラウザー側の仕組みとしては、インタプリター方式の JavaScript などが主流となって、Java アプレットは使用されていない。一方で Java の安全性・可搬性などの性質は、他の分野のアプリケーションでも役に立つため、現在はむしろアプレット以外のアプリケーション（例えば WWW **サーバー側**で動作してウェブページなどを動的に生成する**サーブレット** (Servlet) などのプログラム）を作成するために、広く用いられるようになってきている。しかし、オブジェクト指向など Java のさまざまな特徴を理解するためには、グラフィカルユーザーインタフェイス (GUI, Graphical User Interface) のプログラムは現在でも良い教材である。

WWW サーバー側プログラム用のプログラミング言語としては、Perl, PHP, Python, Ruby なども有名だが、これらは**動的型付け**を採用している。つまり、実行時まで型エラーは検出しない。Java はこれらと違い**静的型付け**を採用している。つまり、実行前（コンパイル時）に型エラーを検出する。一般に静的型付けは大規模で信頼性が必要とされるシステムの記述に適している。最近の言語では Go 言語も静的型付けを採用している。

> **Q 1.2.1** Java の中間言語を実行する仮想 CPU をアルファベット 3 文字の略称で何と呼ぶか？
>
> 答: ____

> **Q 1.2.2** 次の文章のうち、正しいものに○、誤っているものに×をつけよ。

1. ☐ Java は動的な型付けを採用し、コンパイル時には型のチェックは行わない。
2. ☐ Java はポインター演算を明示的にプログラマーに提供していない。
3. ☐ 一般的な Java の処理系は純粋なコンパイラー方式でもインタプリター方式でもなく中間言語方式をとる。

1.3 オブジェクト指向プログラミング

Java はオブジェクト指向プログラミング (Object-Oriented Programming, OOP) 言語である。＿＿＿＿言語 ・ ＿＿＿言語 ・ ＿＿＿言語 ・オブジェクト指向言語などと、プログラミング言語を分類することがあるが、このような言語の分類は、主にプログラミングパラダイム（プログラミング言語が備える部品化の仕組み）に基づいている。

オブジェクト指向言語に限った話ではないが、プログラムの部品を**設計**することは、単に部品を**利用**することよりも格段に難しい。まずは、自分で独自のプログラム部品を設計するよりも、オブジェクト指向という仕組みのおかげで豊富に用意された Java の部品群を利用することを学ぶことが必要であろう。この節では、まず、オブジェクト指向言語が用意する部品を**利用する**ために必要な用語を紹介する。

オブジェクト指向 (object-oriented) とは簡単に言えば、従来の手続きを中心としたプログラム部品（**関数**、**サブルーチン**） の利用に加えて、データを中心とした部品（＿＿＿＿＿＿）の利用を支援することである。関数（サブルーチン）はいくつかの手続きをまとめて一つの部品としたものだが、オブジェクトは、いくつかのデータをまとめて一つの部品としたものである。

関数	**オブジェクト**
代入文, 繰り返し文, 条件判断文 … などの文や式をひとまとめにしたもの	整数, 実数, 文字列 関数（メソッド） … などのデータをひとまとめにしたもの

実際には、プログラム部品として提供されるのは、オブジェクトそのものではなく、オブジェクトの雛型とでもいうべき ＿＿＿(class) である。クラスは、そこから生成されるオブジェクトが、具体的なデータ —つまり、1 とか 3.14—ではなく、どのような名前と型の構成要素を持つか、のみを指定したものである。クラスを具体化（instantiate — つまり、`x` という名前の `int` 型の構成要素は 1 で、`y` という名前の `float` 型の要素は、3.14 などと定めること）したものがオブジェクトである。このとき、このオブジェクトはもとのクラスの ＿＿＿＿（instance, 具体例）である、という。

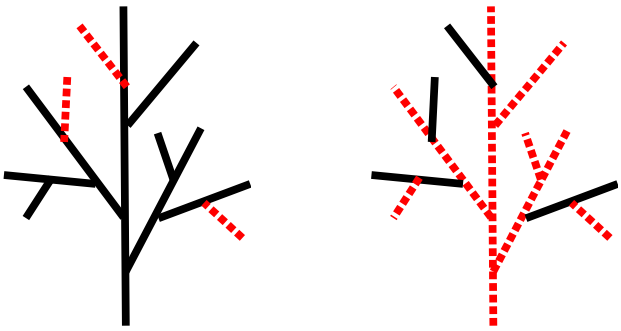
オブジェクトを構成している個々の構成要素を ＿＿＿＿(field) あるいは**インスタンス変数** (instance variable)、**メンバー** (member)、という。ただし、関数型

の要素は＿＿＿＿(method) と呼ぶのが普通である。オブジェクトのメソッドを起動することを、擬人的にオブジェクトに**メッセージ** (message) を送る、と表現することがある。

> **注:** メソッドについては、オブジェクト（インスタンス）ごとにコードを定義するのではなく、通常はクラスに対してコードを定義する。ただし、メソッドから参照されるフィールドはインスタンス毎に異なるものである。オブジェクトは各フィールドのデータの他に、自分がどのクラスに属しているか、という情報を持っていて、それによって適切なメソッドのコードが起動される。

複数のオブジェクトがフィールドに内部状態を保持し、互いにメッセージを交換して、その内部状態を変更していく、というのがオブジェクト指向のプログラムの実行のイメージである。

従来型言語では、部品の再利用方法は、既存の部品を関数・サブルーチンとして呼び出すだけだったが、オブジェクト指向言語では、それに加えて既存の部品（つまりクラス）に少しだけ機能を追加したり、一部を置き換えたり（＿＿、インヘリタンス、 inheritance）して新しいクラス（**サブクラス**, subclass）を作る、という形の再利用の方法が可能になる。サブクラスに対して、元の継承されたほうのクラスのことを**スーパークラス** (superclass) と呼ぶ。サブクラスは、いわばスーパークラスの能力をすべて引き継いでいるので、スーパークラスのオブジェクトを期待しているところにサブクラスのオブジェクトを渡すことができる。言い換えれば、サブクラスのオブジェクトはスーパークラスのオブジェクトの代わりをすることができる。 プログラムの構造を木のように考えると、手続き型言語ではプログラムの“幹”の部分を変えて“枝”の部分だけを再利用することができたが、オブジェクト指向言語では、“枝”の部分を変えて“幹”の部分を再利用することもできるのである。



手続き型言語のイメージ　　オブジェクト指向言語のイメージ
（赤色・破線が再利用できる部分）

最近のソフトウェアではユーザーインタフェースの部分（“枝”の部分）が重要であることが多いので、オブジェクト指向という考え方が特に必要となってきている。オブジェクト指向言語は GUI 部品（ボタンやテキストフィールドなど）のような特定の用途の多種のデータ型が必要とされるアプリケーションプログラム、ゲームやシミュレーションなどに適している。

Q 1.3.1 オブジェクト指向言語で、プログラムの基本部品となる、オブジェクトの雛型のことを何と呼ぶか？

答:

Q 1.3.2 オブジェクト指向言語で、クラスに少しだけ機能を追加したり、一部を置き換えたりして新しいクラスを定義することを何というか？

答:

（参考） Java にはさらに **インタフェース (interface)** という、クラスの継承とは別の形でクラスの定義を補助する仕組みがある。インタフェースはクラスのメソッドの型だけを定めたもので、あるクラスがインタフェースの定めるメソッドをすべて持っているとき、そのクラスはそのインタフェースを実装している、という。インタフェースとその実装クラスの関係は、当面はスーパークラスとサブクラスの関係と同じようなものと考えてよい。つまり、あるインタフェースを期待している箇所には、そのインタフェースの実装クラスのオブジェクトを渡すことができる。

キーワード:

Java, C, C++, JavaScript, オブジェクト指向、アプレット、中間言語方式、JIT コンパイラー、サーブレット、オブジェクト、クラス、インスタンス、フィールド（インスタンス変数）、メソッド、継承