Java基本

プログラミング入門 - Write Once, Run Anywhere

基本文法
変数・制御構文

オブジェクト指向
クラス・継承

実践応用
Web開発基礎

対象レベル

- プログラミング初心者（他言語経験者も歓迎）
- Javaの基礎を体系的に学びたい方
- エンタープライズ開発への第一歩を踏み出したい方

学習目次

1. はじめに - Javaとは
   言語特徴・適用領域
2. オブジェクト指向概念
   クラス・継承・ポリモーフィズム
3. 環境準備
   JDK・IDE・Hello World
4. プログラム構造
   パッケージ・クラス・メソッド
5. JVM動作原理
   コンパイル・実行プロセス
6. 基本文法
   型システム・制御構文
7. クラス設計
   インターフェース・継承
8. データ取扱
   ファイルI/O・DB連携
9. 実践応用
   Webアプリケーション開発
10. スキルアップ
    ロードマップ・認定資格
11. よくある質問
    FAQ・トラブルシューティング
12. 参考資料
    リンク・次のステップ

学習時間の目安

詳細学習: 45-60分
概要把握: 20-30分
復習: 15-20分

Javaとは

Java言語の特徴

- **Write Once, Run Anywhere (WORA)**
  一度書けばどこでも実行可能なプラットフォーム独立性
- **オブジェクト指向プログラミング**
  再利用性と保守性を重視した設計思想
- **強い型システム**
  コンパイル時エラーチェックによる安全性
- **自動メモリ管理**
  ガベージコレクションによるメモリリーク防止
- **豊富なライブラリ**
  標準APIとオープンソースエコシステム

> **Java開発の歴史**
> 1995年にSun Microsystems（現Oracle）が開発。 Web技術の発達とともにエンタープライズ主力言語となりました。

適用領域

エンタープライズシステム
業務システム、ERP、CRM等の大規模システム開発

Webアプリケーション
Spring Boot、JSF等を使用したWebサービス開発

Androidアプリ
Android Studio、Kotlinと連携したモバイル開発

マイクロサービス
Docker、Kubernetes環境でのクラウドネイティブ開発

バージョン情報
現在の最新版はJava 21 LTS（2023年）
企業では Java 8、11、17 LTS が広く使用

オブジェクト指向概念

4つの基本原則

カプセル化 (Encapsulation)
データとメソッドを一つのクラスにまとめ、外部からのアクセスを制御

継承 (Inheritance)
既存クラスの機能を拡張して新しいクラスを作成

ポリモーフィズム (Polymorphism)
同じインターフェースで異なる動作を実現

抽象化 (Abstraction)
複雑な実装を隠し、必要な機能のみを公開

クラスとオブジェクトの関係

クラスは設計図、**オブジェクト**は実際の製品です。
例：「Car」クラスから「myToyota」「yourHonda」オブジェクトを作成

実世界のモデリング例

自動車管理システム

基底クラス：Vehicle（車両）
派生クラス：Car、Truck、Motorcycle
共通メソッド：start()、stop()、getInfo()

社員管理システム

基底クラス：Employee（社員）
派生クラス：Manager、Developer、Designer
共通属性：name、id、salary

図形描画システム

基底クラス：Shape（図形）
派生クラス：Circle、Rectangle、Triangle
抽象メソッド：calculateArea()、draw()

オブジェクト指向の利点

コードの再利用性向上
保守性とスケーラビリティ
チーム開発での分業効率化
現実世界の直感的なモデリング

環境準備

セットアップ手順

1. **JDKのインストール**
   Oracle JDK 11以降またはOpenJDKをダウンロード・インストール
2. **環境変数の設定**
   JAVA_HOME、PATHを設定してコマンドラインから実行可能にする
3. **IDEの選択・設定**
   統合開発環境をインストールして初期設定を完了
4. **Hello World実行**
   最初のJavaプログラムを作成して動作確認

バージョン確認コマンド

```
java -version
javac -version
```

推奨ツール

- **OpenJDK 17 LTS**
  無償で商用利用可能なJDK
- **IntelliJ IDEA**
  高機能IDE（Community版無料）
- **Eclipse IDE**
  オープンソースの統合開発環境
- **VS Code**
  軽量エディタ＋Java拡張

開発環境のポイント

LTS（Long Term Support）版の選択推奨
企業環境では特定バージョンの指定あり
Maven/Gradleビルドツールも併用
Git連携設定も忘れずに

プログラム構造

基本構成要素

- **パッケージ宣言**
  クラスの所属する名前空間を指定
- **import文**
  他のクラスやパッケージを使用するための宣言
- **クラス定義**
  データとメソッドをまとめるオブジェクトの設計図
- **メソッド**
  特定の処理を実行する関数（mainメソッドは実行開始点）

```
// 基本的なJavaプログラム構造

package
com.example.myapp;
import
java.util.*;

public class
HelloWorld {

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

命名規則

クラス名

PascalCase（先頭大文字）
例：StudentManager、UserService

メソッド名・変数名

camelCase（2番目から大文字）
例：getUserName()、studentAge

パッケージ名

すべて小文字、ドット区切り
例：com.company.project.util

定数

すべて大文字、アンダースコア区切り
例：MAX_VALUE、DEFAULT_SIZE

mainメソッドの特徴

public: 外部から呼び出し可能
static: オブジェクト生成不要
void: 戻り値なし
String[] args: コマンドライン引数

JVM動作原理

コンパイルと実行プロセス

1 **Javaソースコード (.java)**

人間が読める形式で記述されたプログラムコード

↓

javac コンパイラ

2 **バイトコード (.class)**

JVM向けの中間コード、プラットフォーム独立

↓

JVM実行エンジン

3 **ネイティブコード実行**

各OS向けの機械語に変換されて実行

JVMコンポーネント

クラスローダー

必要なクラスファイルを動的に読み込み

メモリ領域

ヒープ・スタック・メソッド領域で管理

実行エンジン

インタープリター・JITコンパイラーで実行

ガベージコレクター

不要なオブジェクトを自動的に削除

JVMの利点

- プラットフォーム独立性
- 自動メモリ管理
- セキュリティ機能
- パフォーマンス最適化

基本文法

制御構文

条件分岐 (if文)

```
if (age >= 18) {
  System.out.println("成人");
}
```

繰り返し (for文)

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

選択 (switch文)

```
switch (day) {
  case 1: name = "月"; break;
  default: name = "他";
}
```

データ型システム

プリミティブ型

int: 整数 (-2億 ~ +2億)
double: 小数点数
boolean: true/false
char: 文字 ('A')

参照型

String: 文字列
Array: 配列 (int[], String[])
Object: オブジェクト型

コレクション

List: 順序ありリスト
Set: 重複なし集合
Map: キー-値ペア

変数宣言のポイント

型名 変数名 = 初期値;
例: int age = 25;
String name = "田中";

クラス設計

応用機能

インターフェース
クラスが実装すべきメソッドの仕様を定義

抽象クラス
一部の機能を実装し、一部をサブクラスに委ねる

アクセス修飾子
public, private, protectedでアクセス範囲を制御

staticとfinal
クラスレベルメンバーと不変値の定義

内部クラスとラムダ式
コードの簡潔化とモジュラー化

設計パターン例

Singletonパターン
インスタンスを一つだけ作成するデザインパターン

Factoryパターン
オブジェクト作成を専用クラスに委託する方式

Observerパターン
状態変化を他のオブジェクトに通知する仕組み

Builderパターン
複雑なオブジェクトを段階的に構築する方式

設計のポイント
単一責任の原則
開放・閉鎖の原則
インターフェース分離の原則
依存性逆転の原則

データ取扱

ファイルI/O処理

テキストファイル読み書き

BufferedReader, PrintWriterを使用した効率的なファイル操作

Stream API活用

フィルタリング、マッピング、集約処理の関数型アプローチ

リソース管理

try-with-resources文で自動クローズ処理を実現

try-with-resourcesの例

```
try (FileReader fr = new FileReader("file.txt");
     BufferedReader br = new BufferedReader(fr)) {
  // ファイル読み込み処理
} // 自動クローズ
```

JDBCデータベース連携

データベース接続

DriverManagerで接続管理、Connectionオブジェクト取得

SELECTクエリ実行

PreparedStatementで安全なSQL実行、ResultSetで結果取得

INSERT/UPDATE/DELETE

データ更新操作、トランザクション制御

SQLインジェクション対策

プリペアドステートメントでパラメータ化クエリ

推奨ツール

MyBatis: SQLマッピングフレームワーク
JPA/Hibernate: ORMソリューション
HikariCP: 高速コネクションプール

実践応用 - Webアプリケーション開発

基本技術スタック

サーブレットとJSP

Java Webアプリケーションの基礎技術

- HTTPリクエスト/レスポンス処理
- セッション管理と状態維持
- フォームデータ処理

MVCパターン

Model-View-Controller設計パターン

- Model: ビジネスロジックとデータ
- View: ユーザーインターフェース
- Controller: リクエスト処理制御

Spring Framework

エンタープライズアプリケーション開発フレームワーク

- 依存性注入(DI)と制御の逆転(IoC)
- AOP(アスペクト指向プログラミング)
- Spring Bootでの簡単セットアップ

モダン開発アプローチ

Spring Boot

設定よりも規約を重視した高速開発フレームワーク

RESTful API

RESTアーキテクチャでのWebサービス設計と実装

マイクロサービス

Spring Cloudでの分散システム構築

セキュリティ

Spring Securityでの認証・許可制御

学習ロードマップ

1. サーブレット/JSPの基礎
2. Spring Boot入門
3. データベース連携 (JPA)
4. REST API開発
5. フロントエンド連携

スキルアップロードマップ

基礎習得ステップ (3ヶ月計画)

1 1ヶ月目: 基本文法習得

変数、データ型、演算子
制御構文 (if, for, while)
メソッドの基本
配列とコレクション

2 2ヶ月目: オブジェクト指向

クラスとオブジェクトの作成
コンストラクタ、アクセサメソッド
継承とポリモーフィズム
インターフェースと抽象クラス

3 3ヶ月目: 実践アプリケーション

ファイルI/Oと例外処理
JDBCでのデータベース連携
サーブレット/JSP基礎
シンプルWebアプリ作成

認定資格とキャリアパス

Oracle Java SE 11 認定

ブロンズレベル: Java基礎知識を証明する国際資格

Oracle Java SE 11 シルバー

中級レベル: 実務レベルのJavaスキルを証明

Spring Professional 認定

Spring Frameworkの専門知識を証明する資格

キャリア進路

Webアプリケーション開発者
エンタープライズシステムエンジニア
Androidアプリ開発者
フルスタックエンジニア
システムアーキテクト

学習のコツ

毎日のコーディング習慣
実際のプロジェクト作成
コミュニティ参加
オープンソース貢献

よくある質問 (FAQ)

トラブルシューティング

NullPointerExceptionが発生します

原因: null値のオブジェクトにアクセスしている
対策: nullチェック、Optionalクラスの使用、適切な初期化

メモリ不足エラーが起きます

原因: ヒープメモリの枯渇、メモリリーク
対策: JVMオプション調整(-Xmx)、オブジェクトの適切な解放

ベストプラクティス

コーディング規約

- 一貫した命名規則の遵守
- コメントの適切な記載
- メソッドの簡潔化

セキュリティ

- 入力値のバリデーション
- SQLインジェクション対策
- センシティブデータの暗号化

パフォーマンス

- StringBuilderで文字列連結
- 適切なコレクション選択
- オブジェクトプールの活用

参考資料と次のステップ

公式リソースとドキュメント

公式ドキュメント

Oracle Java Documentation
https://docs.oracle.com/javase/

Java APIリファレンス
クラスとメソッドの詳細仕様

Javaチュートリアル
初心者向け公式ガイド

次のステップ

Spring Boot入門
モダンJava Webアプリケーション開発フレームワーク

サーブレット&JSP実践
基礎Web技術の実践的なシステム構築

JDBCデータベース連携
データベース操作と永続化層の実装

JUnitテスト入門
単体テストとテスト駆動開発の基礎

あなたのJavaスキルを適用できる分野

- エンタープライズシステム開発
- Webアプリケーション・サービス開発
- Androidアプリケーション開発
- マイクロサービスアーキテクチャ
- ビッグデータ処理システム

Java基本学習、おつかれさまでした！
これからも楽しいJavaプログラミングを続けてください