# モダンJava再入門ハンズオン
# Java in Education For JUGs

JJUG CCC 2024 Fall
Java女子部

# 今日のセッション

Java in Education で公開されているプレゼンテーション、
「Java in Education For JUGs」をもとにしたハンズオンがメインです

- Java女子部 の紹介
- Java in Education について
- Java in Education For JUGs について
- モダンJava ハンズオン

# ハンズオンの準備について

事前準備

- ハンズオン資料のダウンロード
  - https://github.com/java-women/jjug-ccc-2024-fall-hands-on-for-participants
- Java 23のインストール
  - インストール方法がわからない方は、ハンズオン資料の説明を参照ください

事前準備が終わっていなくて、インターネットに接続できない方は
Java女子部のスタッフにお知らせください！

# Java女子部について

2014年1月発足
女性が運営する女性のためのJavaユーザコミュニティ

定期的にJavaと周辺技術に関する勉強会を開催しています

# 最近のイベント

8/17
浴衣でミニセッション大会

6/2
Git/GitHub勉強会
with PyLadies Tokyo

# Java in Education

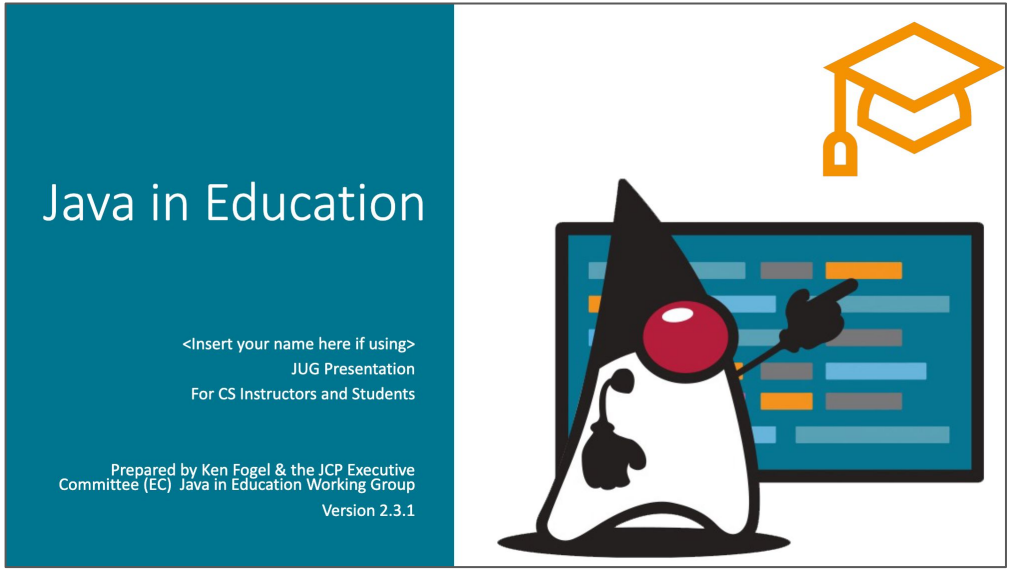
学生や初学者がJavaに親しみやすくするためにJCP(Java Community Process）が行っている取り組み

Javaへのマイナスイメージを払拭したり、モダンなJavaの書き方を広めるような教材（プレゼンテーション、ビデオなど）が公開されている

Java in Education Wiki
https://github.com/jcp-org/Java-in-Education/wiki/Java-in-Education---Wiki-Page

# Java in Education For JUGs

Java in Education で公開されているプレゼンテーション




https://github.com/jcp-org/Java-in-Education/blob/106825a6d073e822aa09a609807366347e011d4a/Java%20in%20Education%20For%20JUGs%202.3.1.pdf

# Javaに関する誤解

🤨 初心者にはとっつきにくい
🤨 Javaって古いじゃん
🤨 簡単なタスクで使いにくい
🤨 オープンソースじゃないよね

→ Javaの学習を簡単にするために改善が進んでいる
→ 安定しているし、ドキュメントも多い
→ 大規模で複雑なアプリケーションに適している
→ Java（OpenJDK）の開発はオープンソースで行われている

# 教育現場でのプログラミング言語

教育現場ではJavaScript、Pythonが人気

- JavaScript
  - 環境を用意するのが簡単
- Python
  - トレンドのビッグデータ、AI/MLと関係が深い

Javaでも機械学習やCライブラリとの連携を簡単にするためのAPIが作られている

# 学生や教員への働きかけ

Javaを学生に教える理由

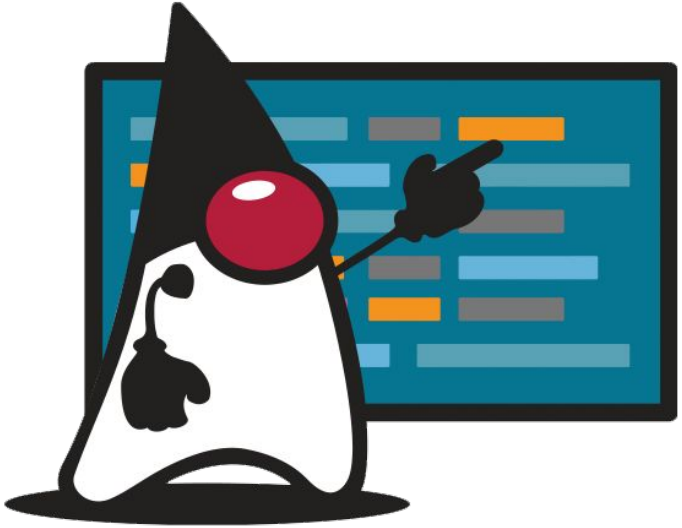
- 金融機関や大規模企業で Java が使われている
- 今後のキャリアで様々な言語を扱うための準備として適している
- プログラミングの意味を学生に明確に理解させられる

学生や教員にJUGに参加してもらおう！

- 学生や教員に JUG に参加するよう勧める
- キャンパスでイベントを開く
- 学生向けの Java ハッカソンを実施する　など

# モダンJava ハンズオン

お手元に資料をご用意ください！
説明パート→ハンズオンパートの順で進みます



Java 女子部

# Launch Multi-File Source-Code Programs Implicitly Declared Classes and Instance Main Methods (複数ファイルのソースコード・プログラムの起動と暗黙的に宣言されたクラスとインスタンスのメイン・メソッド)

# 複数ファイルのソースコード・プログラムの起動

**今までのJavaの実行方法**

- 実行するには 2 ステップ必要
  - コンパイル

```
javac
```

  - 実行

```
java -jar
```

# 複数ファイルのソースコード・プログラムの起動

**これからのJavaの実行方法**おまけ

学習用の機能です。

- ワンステップで実行できるようになる。
  - コンパイルと実行

    ```
    java
    ```

  - ファイルに main を持つ public クラスがある場合、コンパイルして実行。
  - 同じフォルダまたはサブフォルダに複数のクラスファイルを持つことができる。
  - jar ファイルを含めることもできる。
  - IDE をマスターしなくてもよくなる。

# 複数ファイルのソースコード・プログラムの起動

- Javaの実行に関するJEP
  - Java11の`JEP 330 Launch Single-File Source-Code Programs`
    - 明示的なコンパイル操作を省略し、.java ファイルを java コマンドで直接実行できるようになった。

Prog.java

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

```
$ cd part1/JEP330
$ java Prog.java
```

```
Hello!
```

# Launch Multi-File Source-Code Programs

- Java22の`JEP 458 Launch Multi-File Source-Code Programs`
  - 複数のソースコードから構成される Java プログラムのすべてのファイルを事前にコンパイルしなくとも、java コマンドで直接実行すると実行時に必要に応じてコンパイルし、実行できるようになった。

Prog.java

```
class Prog {
    public static void main(String[] args) {
        Helper.run();
    }
}
```

Helper.java

```
class Helper {
  static void run() {
      System.out.println("Hello!");
  }
}
```

```
$ cd part1/JEP458
$ java Prog.java
```

```
Hello!
```

# Preview Features

- 正式リリース前の Java 言語の新機能はすぐに使うことはできない。
  - 追加された新機能はプレビュー機能として利用可能となる。
  - プレビュー機能を利用する場合はオプションの付与が必要。
    - コマンドライン、または IDE でオプションを指定する必要がある。

# Preview Features

- コマンドライン
  - コンパイル

```
javac --source 23 --enable-preview Main.java
```

  - 実行

```
java --enable-preview Main
```

# Preview Features

- JShellの起動

```
jshell --enable-preview
```

# 暗黙的に宣言されたクラスとインスタンスのメイン・メソッド

これまでのHelloWorld（**装飾が多すぎる！**）

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

# 暗黙的に宣言されたクラスとインスタンスのメイン・メソッド

学習用の機能です。

## あたらしいHelloWorld（簡単になった？）

```
void main() {
    System.out.println("HelloWorld");
}
```

# 暗黙的に宣言されたクラスとインスタンスのメイン・メソッド

**より簡単に書けるようになるための工夫が追加された。**

- すべての暗黙化されたクラスで利用可能となった。

```
public static void println(Object obj);
public static void print(Object obj);
public static String readln(String prompt);
```

# 暗黙的に宣言されたクラスとインスタンスのメイン・メソッド

**もっと簡単になったHelloWorld**

```
void main() {

    println("HelloWorld");
}
```

# 暗黙的に宣言されたクラスとインスタンスのメイン・メソッド

**これまでのコンソール操作（コンソールを用いた対話）**

```
try {
    BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
    String line = reader.readLine();
    ...
} catch (IOException ioe) {
    ...
}
```

# 暗黙的に宣言されたクラスとインスタンスのメイン・メソッド

**簡単になったコンソール操作**

```
void main() {
        String name = readln("Name: ");
        print("Pleased to meet you, ");
        println(name);
}
```

# ハンズオン part1

3分

https://github.com/java-women/jjug-ccc-2024-fall-hands-on-for-participants/blob/main/part1.md

JShell
var
Text Blocks

# JShell

- Java9 から導入された。
- Windows ユーザはコマンドプロンプト、Mac/Linux ユーザはターミナルで実行することができる。
- 即座にソースコードの動作確認ができる。

（後で実際手を動かしていくよ！）

# var

- Java10から使用可能。
- 型の宣言がvarでできるようになった。
- varを使うことでソースコードが短くなる。

# var

**今までの書き方**

```
List<String> list = new ArrayList<>();
```

**varを使った書き方**

```
var list = new ArrayList<String>();
```

# var

- var利用時の注意
  - 初期化が必要

```
// コンパイルエラーになる
var obj;
```

  - nullは代入できない

```
// コンパイルエラーになる
var obj = null;
```

# var

- 右辺を見て、型が容易に分からない場合は、可読性が落ちる恐れがある。

**例）**

```
// File#getAbsolutePath()
var path = file.getAbsolutePath();
```

# Text Blocks

- 3つの引用符で文字列をくくることでText Blocksを表現できる。
- 改行なども含めて文字列として入力したものをそのまま使用できる。
- HTML、XML、JSONを含む文字列には特に便利。

# Text Blocks

## 今までの書き方

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

# Text Blocks

## Text Blocksを使った新しい書き方

```
String html = """
                <html>
                    <body>
                        <p>Hello, world</p>
                    </body>
                </html>
                """;
```

# ハンズオン part2

3分

**ハンズオンパートの資料**

コピペするのに使ってください！

https://github.com/java-women/jjug-ccc-2024-fall-hands-on-for-participants/blob/main/part2.md

# ハンズオン part2

3分

**JShellを使ってみよう！**

- 起動（今回はPreview Featuresで起動）

```
$ jshell --enable-preview
```

- 終了

```
jshell> /exit
```

# ハンズオン part2

3分

**JShellでHello Worldを出力してみよう！**

```
jshell> System.out.println("Hello World")
Hello World
```

# ハンズオン part2

おまけ

JShellでは$2のように値を格納する変数が暗黙的に作成される

（$の後の数値は可変）

```
jshell> "Hello Javajo!"
$2 ==> "Hello Javajo!"
jshell> $2
$2 ==> "Hello Javajo!"
```

# ハンズオン part2

3分

**JShellで以下のソースコードをvarに書き換えてみよう！**

```
String hello = "Hello World!";
System.out.println(hello);
```

# ハンズオン part2

3分

**答え**

```
jshell> var hello = "Hello World!"
jshell> System.out.println(hello)
```

# ハンズオン part2

3分

**以下を実行してみてエラーになることを確認しよう！**

```
jshell> var obj1
jshell> var obj2 = null
```

# ハンズオン part2

3分

**実行結果**

```
jshell> var obj1
|  エラー:
|  ローカル変数obj1の型を推論できません
|    （初期化子なしで変数に'var'を使用することはできません）
|  var obj1;
|  ^-------^
```

# ハンズオン part2

3分

**実行結果**

```
jshell> var obj2 = null
|  エラー:
|  ローカル変数obj2の型を推論できません
|    (変数初期化子は'null'です)
|  var obj2 = null;
|  ^--------------^
```

# var

- var利用時の注意
  - 初期化が必要

```
// コンパイルエラーになる
var obj;
```

  - nullは代入できない

```
// コンパイルエラーになる
var obj = null;
```

# var

**以下のソースコードを打ってみて、エラー箇所を修正してみよう！**

```
jshell>  class Sample {
    var hoge = "hoge";

    void sample() {
        var fuga = "fuga";
    }
}
```

# var

**実行結果**

```
|  エラー:
|  'var'はここでは許可されません
|      var hoge = "hoge";
|      ^-^
```

# var

ローカル変数のみ使用可能。

```
class Sample {
    // ここではvarが使えない
    String hoge = "hoge";

    void sample() {
        // ここではvarが使える
        var fuga = "fuga";
    }
}
```

# ハンズオン part2

3分

**Text Blocksを変数で定義して出力してみよう！**

```
jshell> String html = """
                      <html>
                         <body>
                            <p>Hello, world</p>
                         </body>
                      </html>
                      """

jshell> System.out.println(html)
```

# ハンズオン part2

3分

入力した通りに整形されていることが確認できる

```
jshell> System.out.println(html)
<html>
    <body>
        <p>Hello, world</p>
    </body>
</html>
```

Records
Switch

# Records

**概要**

Java14~preview,Java16~release

これまでの値を受け渡すオブジェクトに新しい書き方が増えた

現代のプログラミングモデルは、可変オブジェクトではなく、不変オブジェクトに移行しているので、簡単に対応できるようになった。

- Recordは、コンストラクタ、ゲッター、hashCode、equals、toStringを自動生成。（セッターが削除）

# Records

**メリット**

- コードの記述量が減る
- コードの可読性が上がる

**制約**

- Recordクラスは暗黙的にfinalである。
- 他のインスタンスフィールドを持つことはできない
- 他のクラスをextend（継承）することはできない
- Recordはabstractとして宣言できない

# Records

おまけ

## Recordの使用ポイント

- データ転送オブジェクト（DTO）:
  APIレスポンスやデータベース結果のマッピングに最適
- 複数の戻り値:
  メソッドから複数の値を返す際に、アドホックなタプルの代わりに使用可能
- イミュータブルな値オブジェクト:
  日付範囲や座標など、変更不可能な値を表現するのに最適
- パターンマッチング:
  Java 16以降で導入されたパターンマッチング機能と組み合わせて使用

# Records

**これまでの書き方**

とりあえず、なんかめちゃ長いクラスできちゃう！

ハンズオン資料を見てもらったら、書いています。

```
class Person {
  // フィールド宣言
  // コンストラクタ
  // getter
  // equals/hashcode/toStringのoverride
}
```

# Records

**これからの書き方**

長いクラスがこんなにも短く！

```
public record Person(String name, int age, String address) {
    // コンストラクタ、getter、equals、hashCode、toStringが自動生成される
    // オーバーライドも可能
}
```

# Switch

式と文の違い
式：値を計算して返す構造
　　ex. (x > 0) && (x < 10)
文：動作や処理を行う命令
　　値を返さない

ex.System.out.println("Hello!");

**概要**

Java 14～

Switch文からSwitch式へ、値が返せるように

- 感覚的に書ける
- コード量を減らせる

**新しい書き方**

① “：” の代わりに使える “->”が登場。 自動的にbreak相当の動作をする。

② 複数の定数をカンマで区切って指定できる

③ 複数行からなるブロックから値を返すyieldが追加になった

# Switch

**今までの書き方　：　①、②のケース**

```
switch (day) {
    case MONDAY:
    case TUESDAY:
        emotion = "(´Д`)";
        break;
    case FRIDAY:
        emotion = "ヽ(・∀・)ノ";
        break;
}
```

# Switch

**Java 14以降の書き方　：　①、②、③のケース**

```
String getEmotion(String day) {
    return switch(day) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY -> "(´Д`)";
    case FRIDAY                               -> "ヽ( ・∀・)ノ";
    case SATURDAY, SUNDAY                     -> "ヽ(*´∀`)ノ";
    default -> {
        System.out.println("未知の曜日です: " + day);
        yield "(-_-)";
    }
    };
}
```

# Switch

パターンマッチング：
あるオブジェクトの型や
構造に基づいて条件分岐を
行う機能

**概要**

パターンマッチングが使えるようになった

型チェックとキャストを簡単にし、コードをよりシンプルで安全に

詳細：よこなさんのJEP455の資料見てみてね

sealedとrecordも使ったいい感じの例が出てるよ！


←みんなが求めてた
まとめ資料

# ハンズオン part3

part3.md を参照ください！

# Records

personクラスを作ってみよう

名前と年齢と住所を持った人を表すオブジェクトを書いてみよう！

# Records

**jshellでやってみよう**

```java
public record Person(String name, int age, String address) {}

// インスタンスの自動生成
var shiotsuki = new Person("Shiotsuki",34,"Tokyo");
```

# Records

**jshellでやってみよう**

```
// フィールドへのsetter禁止
shiotsuki.setName("Marika");
// error
|  エラー:
|  シンボルを見つけられません
|    シンボル:   メソッド setName(java.lang.String)
|    場所: タイプPersonの変数 shiotsuki
|  shiotsuki.setName("marika");
|  ^---------------^
```

# Records

**jshellでやってみよう**

```
// フィールドのgetter生成
shiotsuki.age();
// result
$3 ==> 34

// toString()メソッドの自動実装
System.out.println(shiotsuki);
// result
$4 ==> Person[name=Shiotsuki, age=34, address=Tokyo]
```

# Switch

**jshellでやってみよう**

```
String getEmotion(String day) {
    return switch(day) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY -> "(´Д`)";
    case FRIDAY                               -> "ヽ(・∀・)ノ";
    case SATURDAY, SUNDAY                     -> "ヽ(*´∀`)ノ";
    default -> {
        System.out.println("未知の曜日です: " + day);
        yield "(-_-)";
    }
    };
}
```

おまけ

# Virtuous Virtual Threads

# Virtuous Virtual Threads

おまけ

- Java プラットフォームに導入された仮想スレッドのこと

- 仮想スレッド（Virtual Threads）は軽量スレッド
    - 高スループットの同時実行アプリケーションの作成、保守、監視の労力を大幅に削減する

- JDK 21から

# Virtuous Virtual Threads

おまけ

```
public class VirtualThreadClass extends Thread { . . }
 public void perform() {
      for (int i = 0; i < 5; ++i) {
        Thread.ofVirtual().name("Thread # " + i).
          start(new VirtualThreadClass());
}
```

# Virtuous Virtual Threads

おまけ

**今までの書き方、プラットフォーム・スレッドとは**

- OSスレッドの薄いラッパー
- 基盤となるOSスレッド上でJavaコードを実行する
- プラットフォーム・スレッドの存続期間中ずっとOSスレッドを保持する
- 使用可能なプラットフォーム・スレッドの数はOSスレッドの数に制限される

# Virtuous Virtual Threads

おまけ

**新しい書き方、仮想スレッド（Virtual Threads）とは**

- プラットフォーム・スレッドと同様に、仮想スレッドもjava.lang.Threadのインスタンス
- 特定のOSスレッドに関連付けられないにもかかわらず、OSスレッド上でコードを実行する
- 仮想スレッドで実行されているコードがブロッキングI/O操作をコールすると、Javaランタイムは、再開できるまで仮想スレッドを一時停止する
- 一時停止された仮想スレッドに関連付けられていたOSスレッドは解放され、他の仮想スレッドの操作を実行できる

# クロージング

資料に出てくる各機能がどのバージョンから入ったか整理してみる→よこなたすく

# クロージング

今日やったことはこれ！

困ったら、こちらまで！