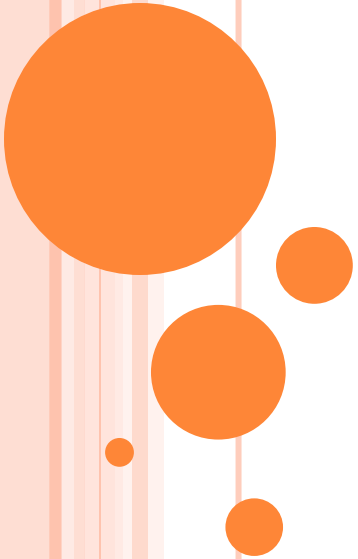


JAVAの活用

オブジェクト指向プログラミング特論

只木進一：総合情報基盤センター

目的

- Javaの様々な機能を使う
  - 便利な機能
  - C++と違うところ
- スマートなプログラム開発
  - ライブラリの活用

PACKAGE

- プログラム全体をモジュールに分ける
- 関連の強いクラスをpackageにまとめる
- ソースファイルのフォルダに対応
  - 階層化できる
- ソースファイル先頭でpackage宣言
  - プロジェクトフォルダ中の相対位置に対応
- 修飾子(public, private)が無いと、package内に公開

IMPORT

- 使用するライブラリクラスの名前解決を支援
  - クラス名が重複しているときには注意が必要
- 特定のクラス：import クラス名
  - 例：import java.util.List;
  - プログラム中のListクラスは、java.util.Listになる
- パッケージ: import パッケージ名
  - 例: import java.awt.*;

修飾子

- アクセス制限：public, private, protected
  - C++と同様な制限
- 定数：final
  - コンストラクタ内で一度だけ値を設定することも可能
- クラスに属する変数、メソッド：static
  - クラスインスタンスを作らなくても存在

例外処理

- 実行時に発生するエラーへの対処を記述する
  - ファイルが開かない
  - 正しく値を変換できない
- メソッドが例外を返す：throws
- メソッドの呼出側がそれを処理する

```
try{
    処理
} catch(例外 ex){
    例外処理
}
```

例外処理：例

```
public class FileIOMain {

    private File inFile;
    private File outFile;
    private final String nl = System.getProperty("line.separator");

    public FileIOMain() throws FileNotFoundException {
        inFile = new File("input.txt");
        if (!inFile.isFile() || !inFile.canRead()) {
            String message = "入力ファイル"
                    + inFile.getAbsolutePath() + "がありません。";
            throw new java.io.FileNotFoundException(message);
        }
        outFile = new File("output.txt");
    }
```

入力ファイルinput.txtが存在しないと
例外FIleNotFoundExceptionをthrow

```
public static void main(String[] args) throws IOException {
        FileIOMain fileIOMain = null;
        try {
            fileIOMain = new FileIOMain();
        } catch (FileNotFoundException ex) {
            Logger.getLogger(
                    FileIOMain.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
        if (fileIOMain != null) {
            int n = fileIOMain.getData();
            System.out.println(n);
        }
    }
}
```

FileIOMainのコンストラクタから例外が出ると、エラーログを生成

型パラメタの利用 (GENERIC)

- クラスが保持するデータの型を明示する
  - コンパイル時に整合性を確認する
  - コレクションライブラリ(リストなど)で重要

- 例:Integerクラスのインスタンスからなる List

```
java.util.List<Integer> list = new java.util.List<>();
.....
Integer a = list.get(0);
```

Boxing

- 原始型：int, double, boolean, charなど
- 対応するクラス：Integer, Double, Boolean, Character
- 自動でキャストできる

```
int n = Integer.valueOf(10);
Integer nInt = n;
```

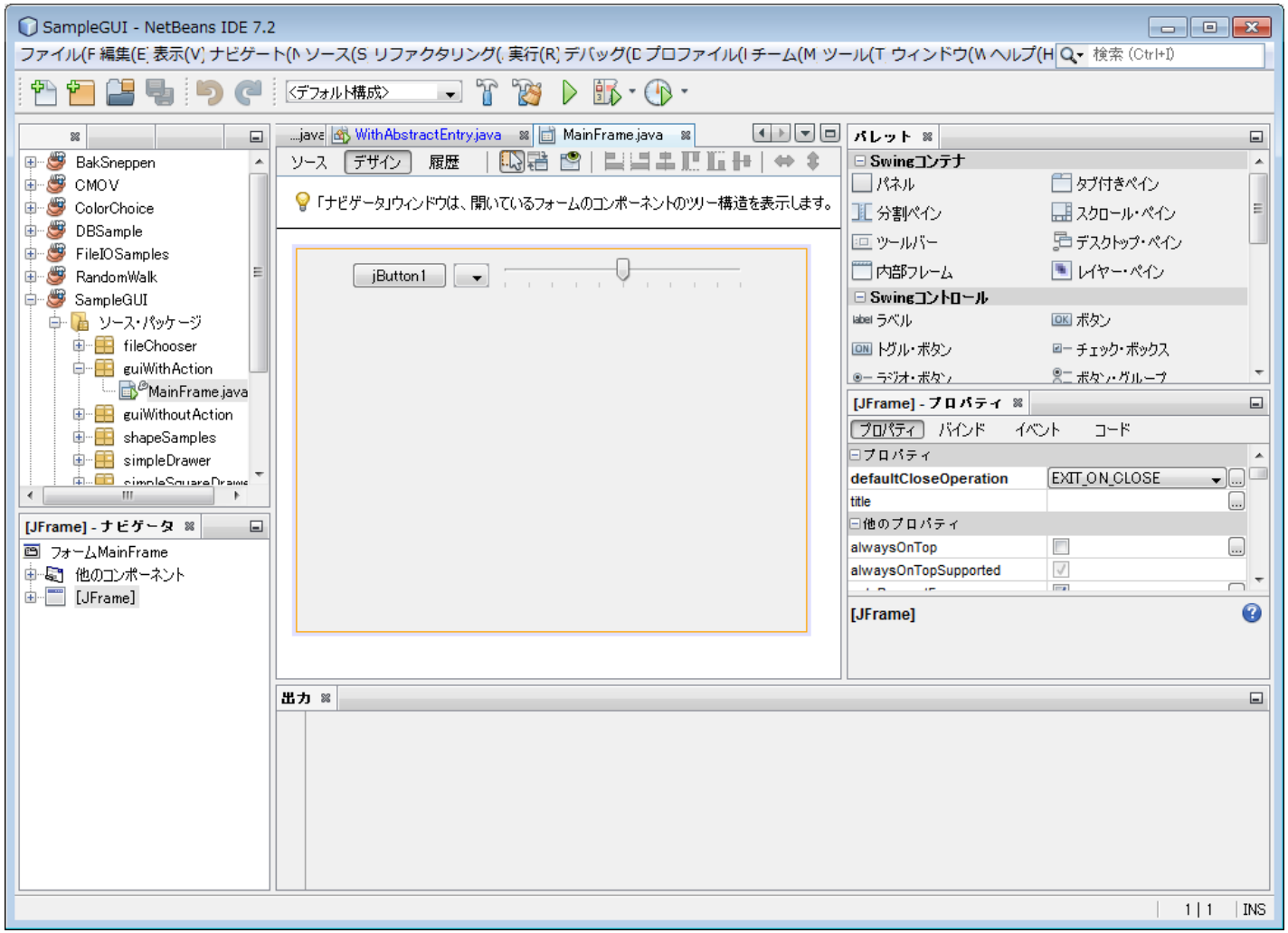
拡張されたFOR

- コレクションの全要素に対するfor

```
List<Integer> list;
…
for(Integer e:list){
    処理
}
```

Graphical User Interface

- javaでは、標準にGUIコンポーネントがある
- 基本的コンポーネント：javax.swing.*
- 描画用クラス
  - java.awt.*;
  - java.awt.geom.*;
  - java.awt.Graphics.*;
  - java.awt.Graphics2D.*;

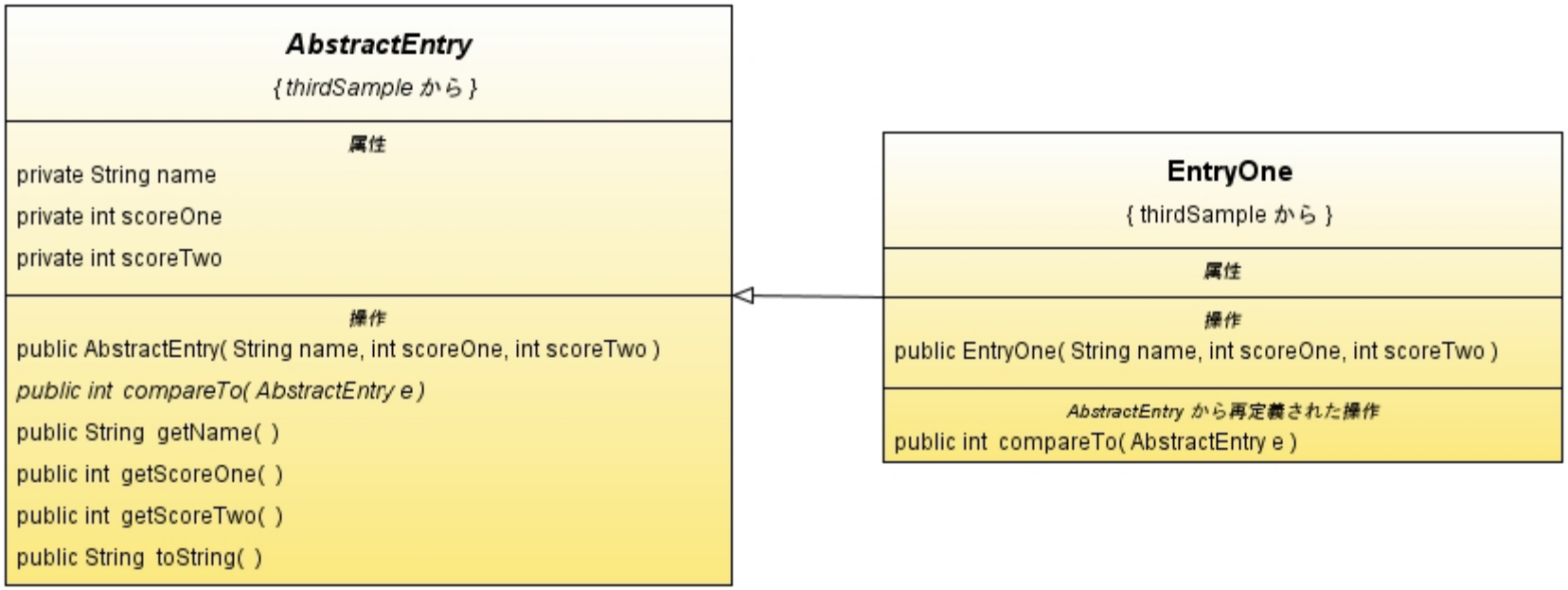

SampleGUI - NetBeans IDE 7.2
ファイル(F 編集(E 表示(V ナビゲート(N ソース(S リファクタリング(実行(R デバッグ(D プロファイル(P チーム(M ツール(T ウィンドウ(W ヘルプ(H
検索 (Ctrl+I)
<デフォルト構成>
BakSneppen
CMOV
ColorChoice
DBSample
FileIOSamples
RandomWalk
SampleGUI
ソース・パッケージ
fileChooser
guiWithAction
MainFrame.java
guiWithoutAction
shapeSamples
simpleDrawer
WithAbstractEntry.java
MainFrame.java
ソース
デザイン
履歴
「ナビゲータ」ウィンドウは、開いているフォームのコンポーネントのツリー構造を表示します。
jButton1
パレット
Swingコンテナ
パネル
タブ付きペイン
分割ペイン
スクロール・ペイン
ツールバー
デスクトップ・ペイン
内部フレーム
レイヤー・ペイン
Swingコントロール
ラベル
ボタン
トグル・ボタン
チェック・ボックス
[JFrame] - プロパティ
プロパティ
バインド
イベント
コード
defaultCloseOperation
EXIT_ON_CLOSE
title
他のプロパティ
alwaysOnTop
alwaysOnTopSupported
[JFrame]
[JFrame] - ナビゲータ
フォームMainFrame
他のコンポーネント
[JFrame]
出力
1 | 1
INS

例題2

- Entryクラスのインスタンスのリストの保持
  - java.util.Listクラスの利用
  - コレクションとテンプレートの実例
- 二分木によるソートをするクラス
  - java.util.TreeSetクラスの利用
  - java.lang.Comparableインターフェイスによる「自然な順序」

例題3

- scoreを二つ持っていて、どちらでソートするか
- AbstractEntryクラス：抽象クラス
  - compareTo()メソッドを実装せず
- AbstractEntryクラスの継承クラスEntryOne


AbstractEntry
{ thirdSample から }
属性
private String name
private int scoreOne
private int scoreTwo
操作
public AbstractEntry(String name, int scoreOne, int scoreTwo)
public int compareTo(AbstractEntry e)
public String getName( )
public int getScoreOne( )
public int getScoreTwo( )
public String toString( )
EntryOne
{ thirdSample から }
属性
操作
public EntryOne(String name, int scoreOne, int scoreTwo)
AbstractEntry から再定義された操作
public int compareTo(AbstractEntry e)