システム創成 プロジェクトⅠ
画像認識 演習(第1回)

システム創成情報工学科

演習担当:尾下 真樹、齊藤 剛史、斎藤 寿樹

徳永 旭将、宮野 英次、藤本 晶子

グループでの演習

- ２人１組のグループで演習を行う
  - グループ分けは、あらかじめこちらで決めているので、それに従うこと
  - ２人で一緒に、プログラム作成（演習）、実験、プレゼンテーションなどを行う
  - 成績は、最後のプレゼンテーションで評価される
- ２人１組のグループで隣り合わせに座るように、席を移動する

プロジェクトⅠ　日程(1)

- 1週目 画像認識(1)
  - 3限目　講義(特徴量を使った識別)(佐藤)
  - 4限目　講義(演習説明)(尾下or齊藤or斎藤or徳永or宮野or藤本)
  - 5限目　演習
- 2週目 画像認識(2)
  - 3限目　講義(演習説明)(尾下or齊藤or斎藤or徳永or宮野or藤本)
  - 4～5限目　演習
- 3週目 画像認識(3)
  - 3～5限目　演習
- 計画書提出(3週目5限目まで)

プロジェクトⅠ　日程(2)

- プログラム提出(4週目の前日まで)
- 課題画像収集作業を4週目3限目までに終える。スキャン作業は4週目5限目までに終える。
- 4週目 識別精度
  - 3限目　講義(識別精度)(本田)
  - 4限目　講義(演習説明)(尾下or齊藤or斎藤or徳永or宮野or藤本)
  - 5限目　演習
- 5週目 自由演習
  - 3限目　プレゼン説明(田上or小守)
  - 3～5限目　演習
- 6週目 プレゼンテーション
  - 3～4限目　プレゼンテーション

プロジェクト情報

- 本プロジェクトのウェブページ

  http://www.cg.ces.kyutech.ac.jp/lecture/project/

  – 講義資料や演習に必要なファイルを公開

- Moodleの「システム創成プロジェクト」
  – プレゼンテーションやプログラムの提出に利用
  – 詳細は後日説明

プロジェクトの目的

- 未知の問題を解決できるような応用力を身につける（創成教育）
  - 講義で習った内容をもとに、現実の問題に取り組み、問題発見・解決能力を身につける
  - 誰も正解を知らない問題に取り組む（重要）
- データ収集、実験、考察、プレゼンテーションまでを行い、卒業研究の基礎を身につける

本演習の目的

- 前の講義で学んだ、特徴量を使ってデータを識別する手法について、実際にプログラムを作成して演習を行うことで、より理解を深める。
- 文字画像認識プログラムの開発を通じて、実践的なプログラムの開発方法を学ぶ。
  特に本演習では、実際のソフトウェア開発における、クラス設計の考え方なども学習する。
- 演習で学んだ内容をもとに、各自の課題に取り組む

画像認識の概要

学習用画像

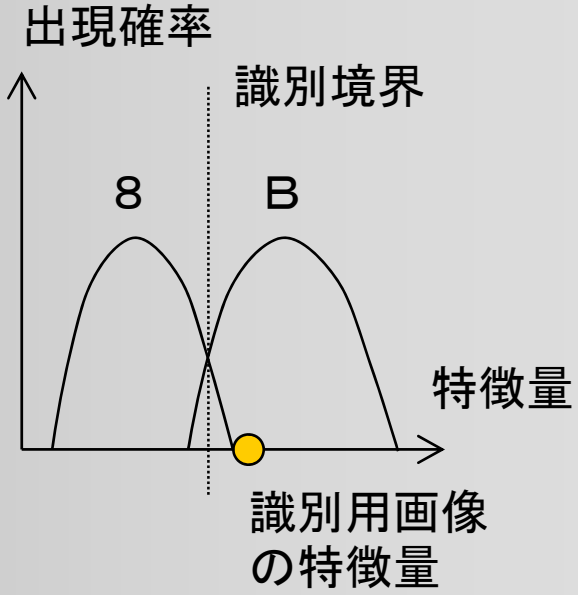
グループ０の画像集合 ″８″

グループ１の画像集合 ″Ｂ″

特徴量計算

学習

学習結果
（識別境界）



識別用画像

特徴量計算

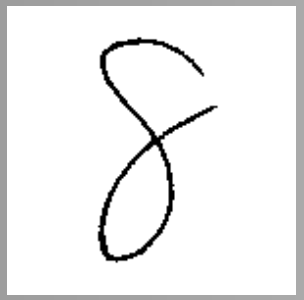
認識

認識結果

８ or Ｂ

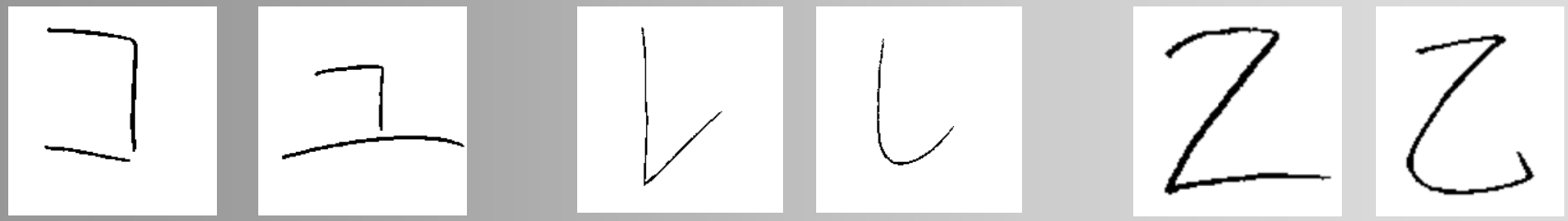
プロジェクト課題(1)

- 準備演習(全員共通)
  - 2種類の画像を識別するプログラムを開発
  - 最初は、こちらで用意した画像データ(8とBの画像データ)を使用する



プロジェクト課題(2)

- 識別する画像を自分達で決める



※ 過去の受講者の課題の例

  - 自分達の決めた2種類の画像を識別できるような特徴量を計算するようプログラムを改良
  - 実際にサンプルを採取(友人等に書いてもらう)して、実験を行い、結果を考察
  - プレゼンテーション

演習手順

- 演習資料を参考に、各自でプログラムを作成する
- これからの説明内容
  - プログラム設計の基本的な考え方を説明
  - 画像認識プログラムの設計を段階的に説明
  - プログラムを実装する上でのポイントを説明
- 演習
  - 演習の詳しい内容は、演習資料を参考に、各自で行う
  - ソースファイルの一部は、講義のページにあるものを利用して良い

資料の内容

- 1～3章　プログラム全体の設計（1次元での識別）
- 4章　プログラム開発手順（1次元での識別）
- 5章　特徴量1の計算処理の作成方法
- 6章　閾値の計算処理の作成方法
- 7章　プログラム全体の設計（2次元での識別）
- 8章　特徴量2の計算処理の作成方法
- 9章　閾値の計算処理の作成方法
- 10章　プログラム開発手順（2次元での識別）
- 11章　プログラム開発手順（各自の画像の識別）

演習の流れ

- 1次元の特徴量を使った識別プログラムの作成
  - 4章の説明に従って作成（5〜6章も参照）
  - 4.2節、4.6節、4.7節、4.8節、6.9節の内容がそれぞれ完成したらTAに報告して確認してもらう。
- 2次元の特徴量を使った識別プログラムに拡張
  - 10章の説明に従って作成（8〜9章も参照）
  - 10.2節、10.3節、10.4節の内容がそれぞれ完成したらTAに報告して確認してもらう。
- 各自の選んだ画像を識別プログラムに拡張
  - 11章の説明に従って作成

グループでの演習の流れ(1)

- 演習資料で説明されているプログラムを、２人で共同で作成する。
  - 作業を分担して、それぞれの端末で作成して、後でプログラムを統合する。
  - ２人で相談しながら、どちらかの端末で作成してもよい。

グループでの演習の流れ(2)

- 複数グループで取り組む課題画像を相談して決める。
  - 課題画像の収集も複数グループで共同で取り組む。
- グループで取り組む特徴量(3種類)を相談して決める。
- 課題画像と作成予定の特徴量(3種類)を計画書に記入し、3週目16:10までに演習担当教員に提出する。

グループでの演習の流れ(3)

- 特徴量(3種類)のプログラムを作成して実験する。
  - 3種類の特徴量から3通りの組み合わせ全てを実験し、結果を考察する。

プロジェクトⅢとの関連

- プロジェクトⅢでも画像関連のテーマを扱う
  - プロジェクトIII
    - 画像データの仕組みや画像処理技術
      - CCDによる画像データの取得、画像データの表現
      - ハフ変換・フーリエ変換などの一般的な画像処理技術
    - 応用として、衛星画像の処理やナンバープレートの識別を行う
  - プロジェクトI
    - 特徴量を使ったデータの分類手法
      - 文字の特徴に注目した画像処理による特徴量の抽出
      - 正規分布モデルに基づくデータの分類
    - 応用として、手書き文字の識別（分類）を行う

※ 画像識別を扱う点では共通しているが、学習する技術は異なる（どちらを先に受講しても問題ない）

プログラムの仕様

プログラムの仕様

- まず、どのようなプログラムを作成するのかという「仕様」を決めることが重要
  - 仕様として、特に、データの流れ（プログラムの入力、出力）が重要になる
    - 何を入力として受け取り、どのような処理を行って、何を出力するのか？

画像認識プログラムの仕様

- 画像認識のテストを行う Java プログラム
  - 入力データ
    - 学習用の画像データ(グループ0・1の画像)

    - 認識用の画像データ(グループ0・1の画像)

認識テスト　○ × ○ ○ ×　　○ × × ○ ○

  - 出力データ
    - 誤認識率を出力 ？？%

画像認識プログラムの概要

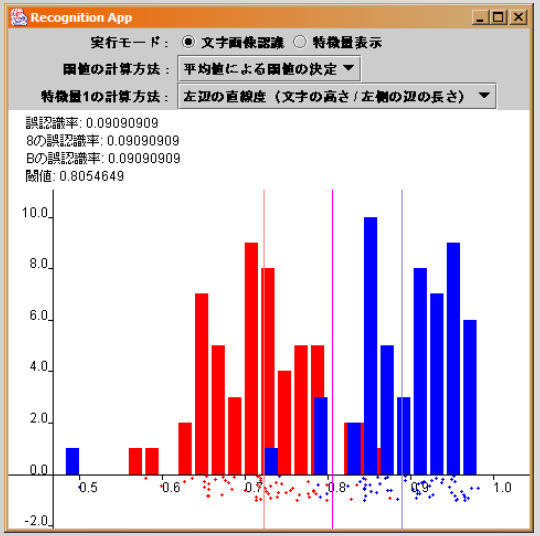
学習用画像

グループ０の画像集合 ″８″

特徴量計算

グループ１の画像集合 ″Ｂ″

学習



学習結果
（識別境界）

評価用画像

グループ０の画像集合 ″８″

特徴量計算

グループ１の画像集合 ″Ｂ″

認識

認識率

※ 複数の画像の認識を行い、認識率を計算

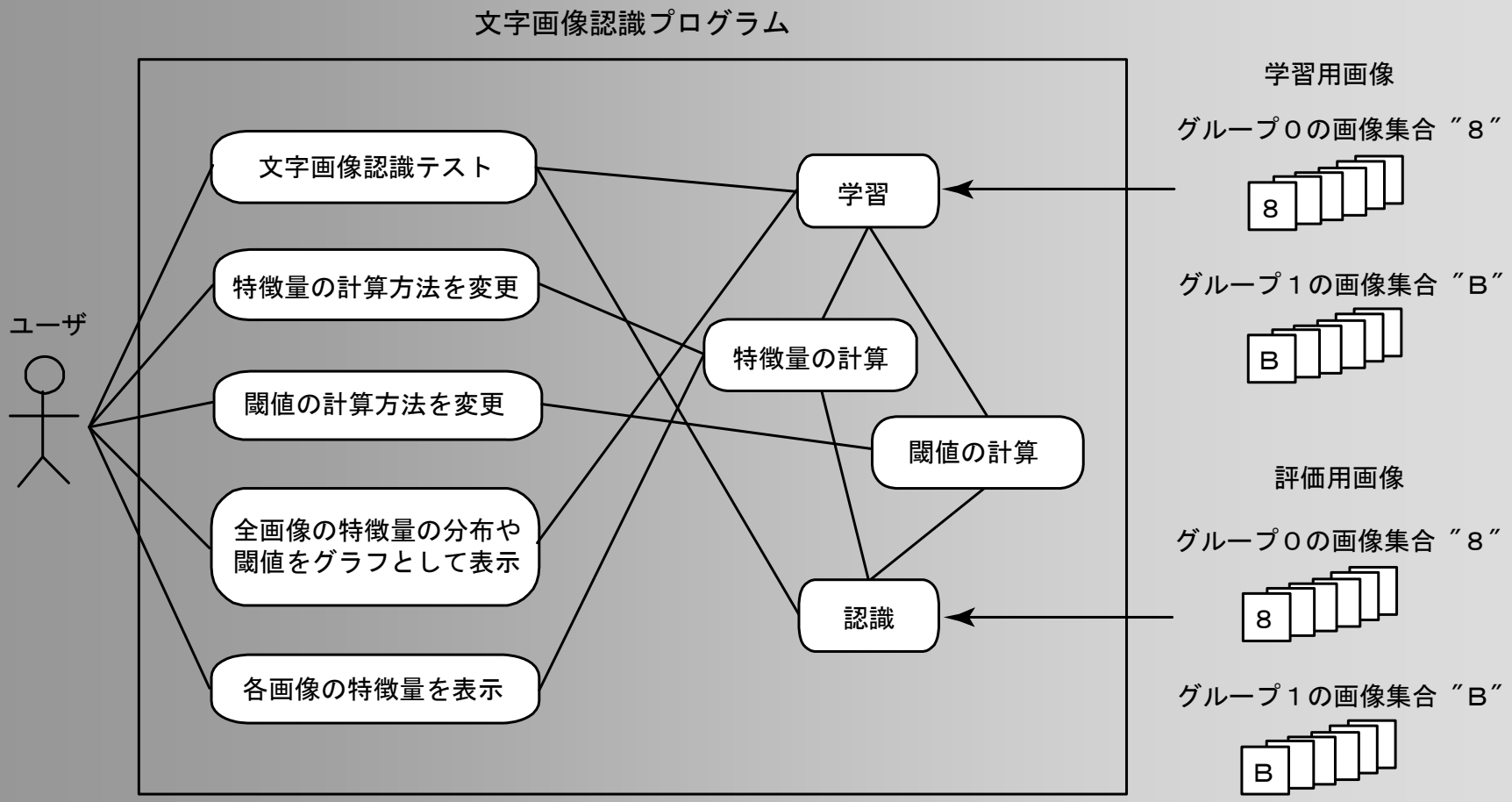
画像認識プログラムの処理の流れ

1. 学習処理
   - 両グループの各学習用画像の特徴量を計算
   - 特徴量の分布から、2つのグループを識別するための閾値（識別境界）を計算
2. 認識処理
   - 評価用画像の特徴量を計算して、学習処理で計算した閾値から、どちらのグループに属するかを判定
   - 各評価用画像ごとに認識処理を繰り返し、どれくらいの割合の画像の認識に失敗したかという、誤認識率を計算する

プログラムに必要な機能

- サンプル画像を読み込んで文字画像認識テストを実行（メイン機能）
- 特徴量の計算方法を変更
- 閾値の計算方法を変更
- 全サンプル画像の特徴量の分布や閾値をグラフで表示（確認用）
- 各サンプル画像の特徴量を表示（確認用）

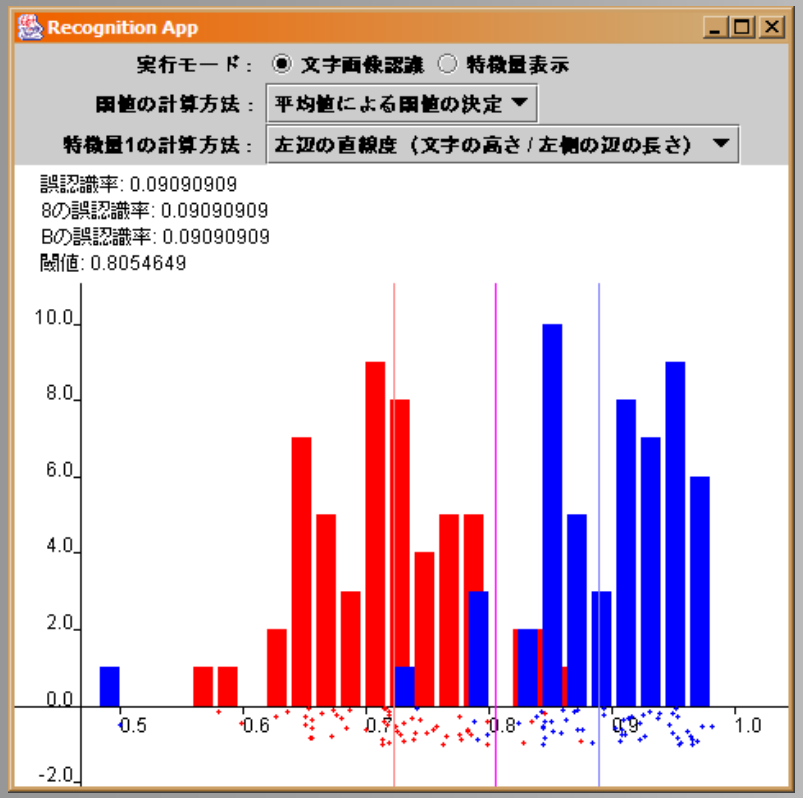
ユースケース図



※ ソフトウェアに実装すべき機能を表した図

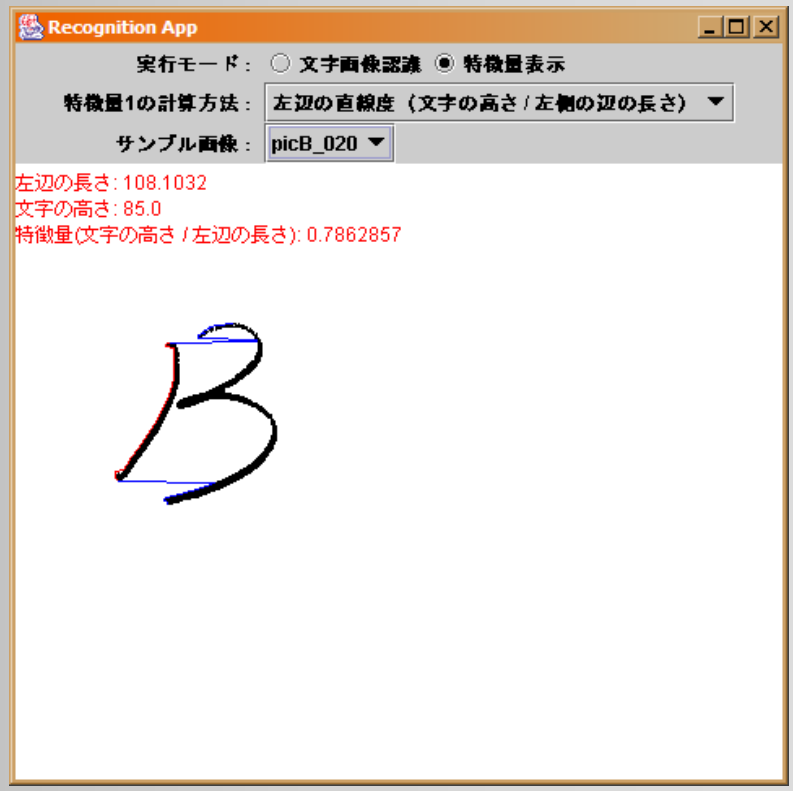
ユーザインターフェース

認識結果の表示



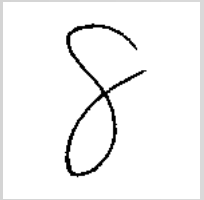
特徴量の分布、閾値、結果の表示
閾値・特徴量の計算方法の変更

各画像の特徴量の表示



画像の選択
特徴量の計算方法の変更

テスト用データ



- テスト用のデータとして、「８」「Ｂ」のそれぞれ55枚の手書き文字をスキャンした画像ファイル（GIF形式、256×256ドット）を用意する
- 最初は、学習用画像と評価用画像に同じものを使用する（手間を減らすため）
  - 後半の演習で、これらを分けた場合の、実験や評価を行う
- 演習では、各自、認識を行う2つの文字を決めて、実際に手書き文字の画像を収集して実験を行う（文字に限らなくても良い）

プログラムの設計

設計の考え方

- 設計
  - クラス定義（各クラスの属性やメソッド）を決める
  - 各オブジェクトがどのように連係して全体の処理を行うかを決める
- 実装
  - 設計に従って実際にプログラムを記述する
  - ある程度規模の大きいプログラムでは、始めにきちんと設計を行うことが必要
  - ソフトウェア開発者には、このような設計ができる能力が期待される

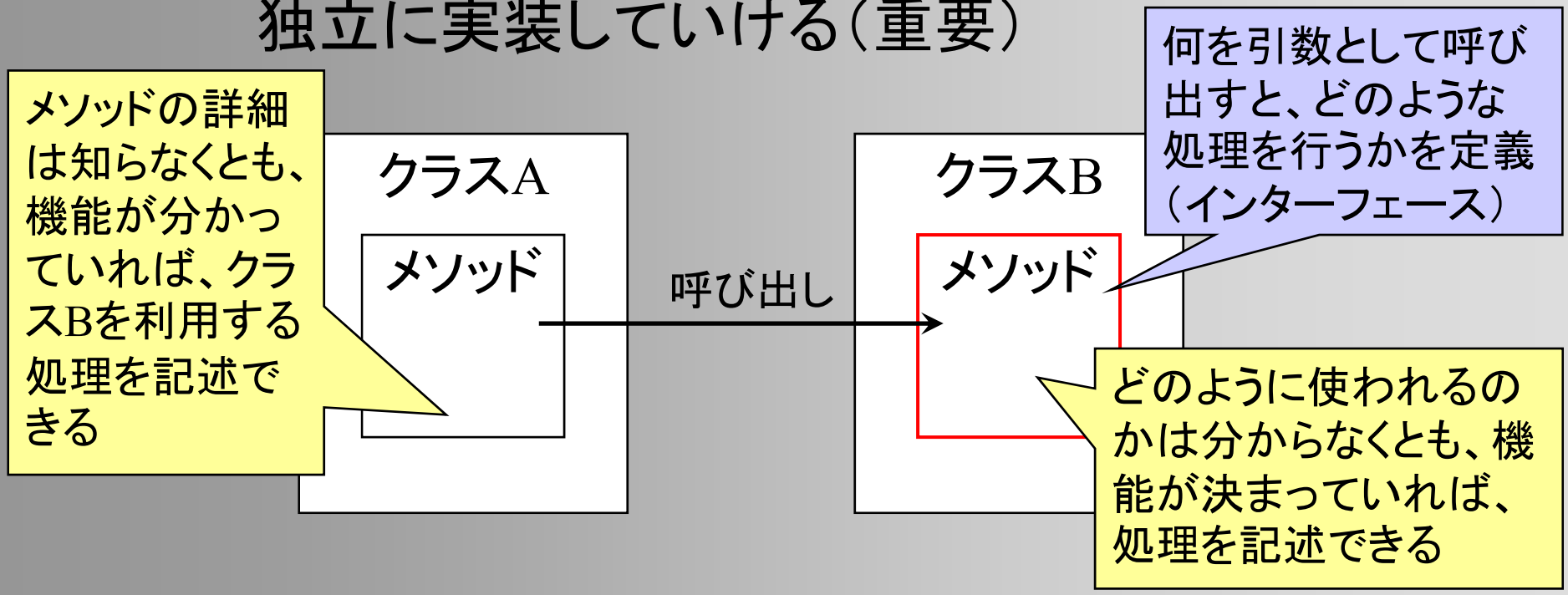
設計の方針

- どのようなクラスを定義するか？
- 基本的には、機能単位でクラスに分ける
- 今回の画像認識プログラムでは、
  - まず、大きく2つのクラスを定義することにする
    - 画像認識クラス
      - 学習・識別の機能を持つクラス
    - テストアプリケーションクラス
      - 画像認識クラスを使って、テストを行うクラス

    ※ 2つを分けておくことで、例えば、画像識別クラスだけを別のアプリケーションで再利用することができる

設計のポイント

- インターフェースをきちんと決めることが重要
  - 各クラスのメソッド（＋引数・戻り値）を決める
  - インターフェースを決めることで、各メソッドを独立に実装していける（重要）



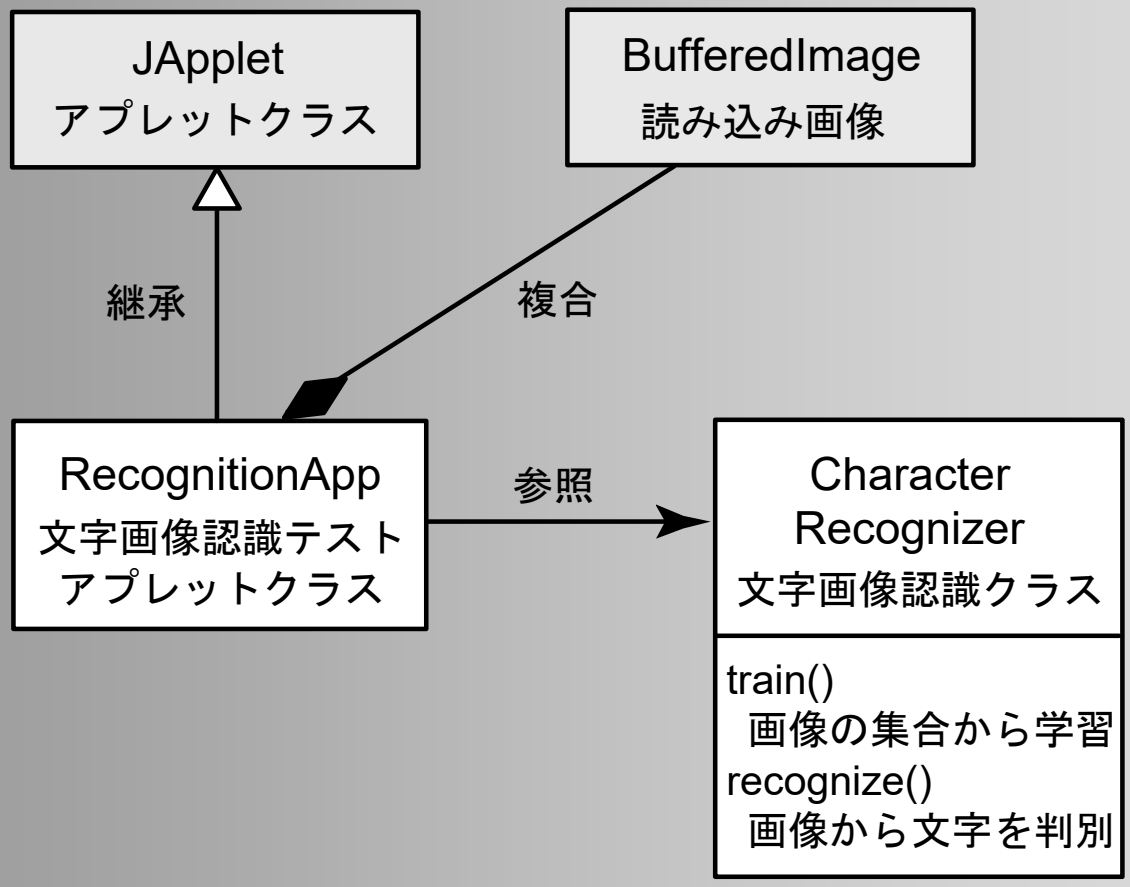
画像認識クラスの設計

- 画像認識クラスに必要なインターフェース
  - 学習メソッド
    - 引数：グループ0の画像配列、グループ1の画像配列

    - 戻り値：なし（識別境界を記録する）
  - 識別メソッド
    - 引数：1つの画像

    - 戻り値：グループ0と1のどちらに属するかを返す

クラス図



※ 上の2つは Java の標準ライブラリのクラス

クラス定義

- 資料3～4ページのソースを参照
  - RecognitionApp, CharacterRecognizer の2つのクラスの定義
  - RecognitionAppから、CharacterRecognizer オブジェクトへの参照を持つ
  - CharacterRecognizer には、2つのメソッドが定義（中身はまだ空）
  - RecognitionAppは、CharacterRecognizer のメソッドを呼び出して全体の処理を実行

```
//
//   文字画像認識クラス
//
class   CharacterRecognizer
{
        // 与えられた２グループの画像データを判別するような特徴量の閾値を計算
        public void   train( BufferedImage[] images0, BufferedImage[] images1 )
        {
                // 各入力画像の特徴量を計算

                // 特徴量の分布をもとに２つのグループを識別するような閾値を計算

        }

        // 学習結果に基づいて与えられた画像を判別
        //（判別した画像の種類として 0 or 1 の値を返す）
        public int   recognize( BufferedImage image )
        {
                // 入力画像の特徴量を計算

                // 閾値に基づいてどちらのグループに属するかを判定

                return   0;
        }
}
```

資料 4ページ

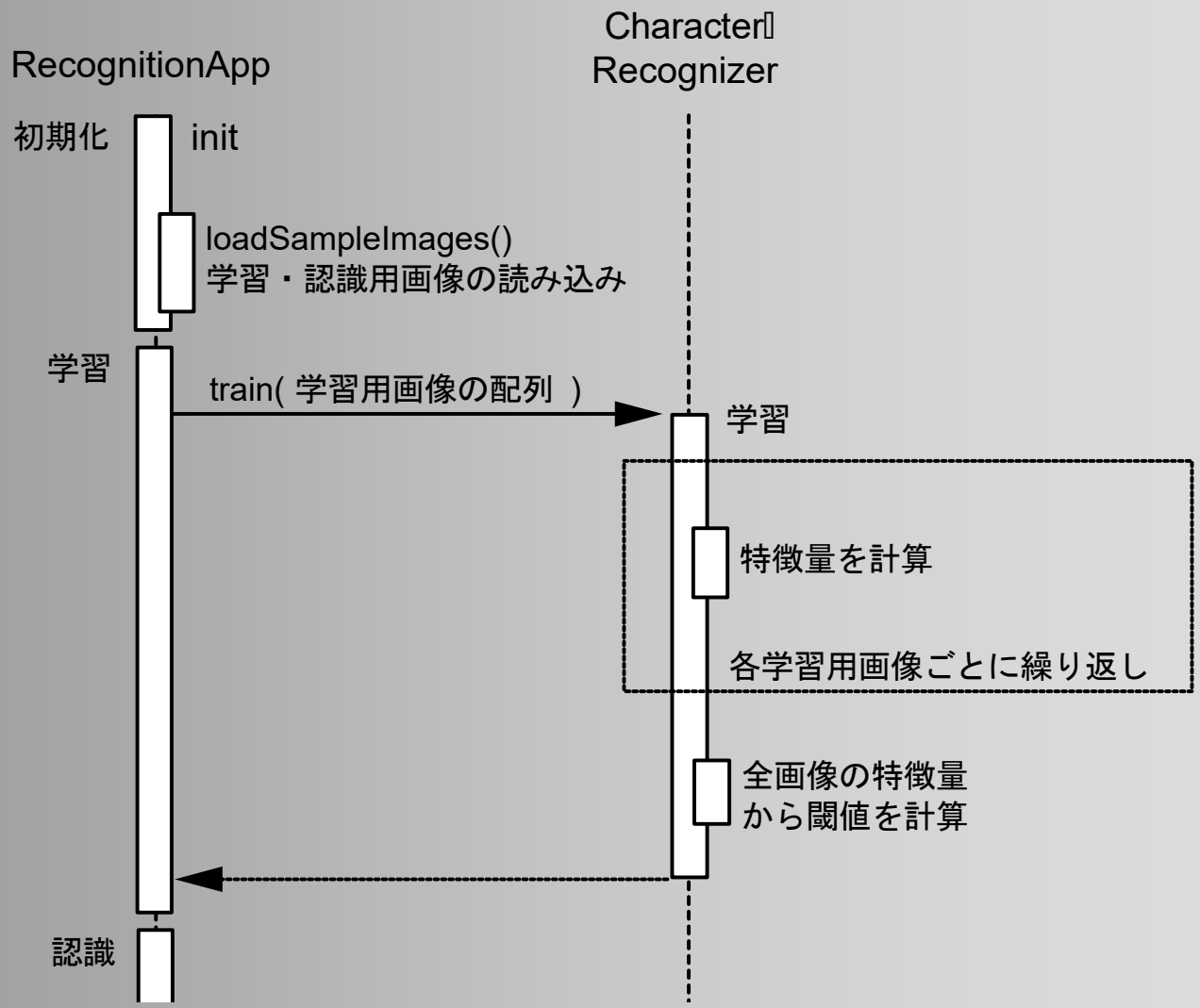
```
//
//   文字画像認識テスト アプレット
//
public class RecognitionApp extends JApplet
{
        // 文字画像認識モジュール
        protected CharacterRecognizer   recognizer;

        // サンプル画像
        protected BufferedImage   sample_images0[];
        protected BufferedImage   sample_images1[];

        // 初期化処理
        public void   init()
        {
                // 文字画像判別モジュールの生成
                recognizer = new CharacterRecognizer();

                // 全サンプル画像の読み込み
                loadSampleImages();

                // サンプル画像を使った文字画像認識のテスト
                recognitionTest();
        }

        // 描画処理
        public void   paint( Graphics g )
        {
                // 誤認識率を画面に描画する
        }

        // サンプル画像の読み込み
        public void   loadSampleImages()
        {
                // サンプル画像を読み込む処理をここに記述
                // sample_images0, sample_images1 にサンプル画像の配列を格納する
        }

        // サンプル画像を使った文字画像認識のテスト
        public void   recognitionTest()
        {
                // サンプル画像をもとに文字画像認識オブジェクトの学習を実行
                recognizer.train( sample_images0, sample_images1 );

                // サンプル画像を使って誤認識率を調べる
                // sample_images0, sample_images1 の各画像につき
                // recognizer.recognize( image ) メソッドを呼び出して
                // 正しく判定できるかどうかを調べる
        }
}
```

資料 3〜4ページ

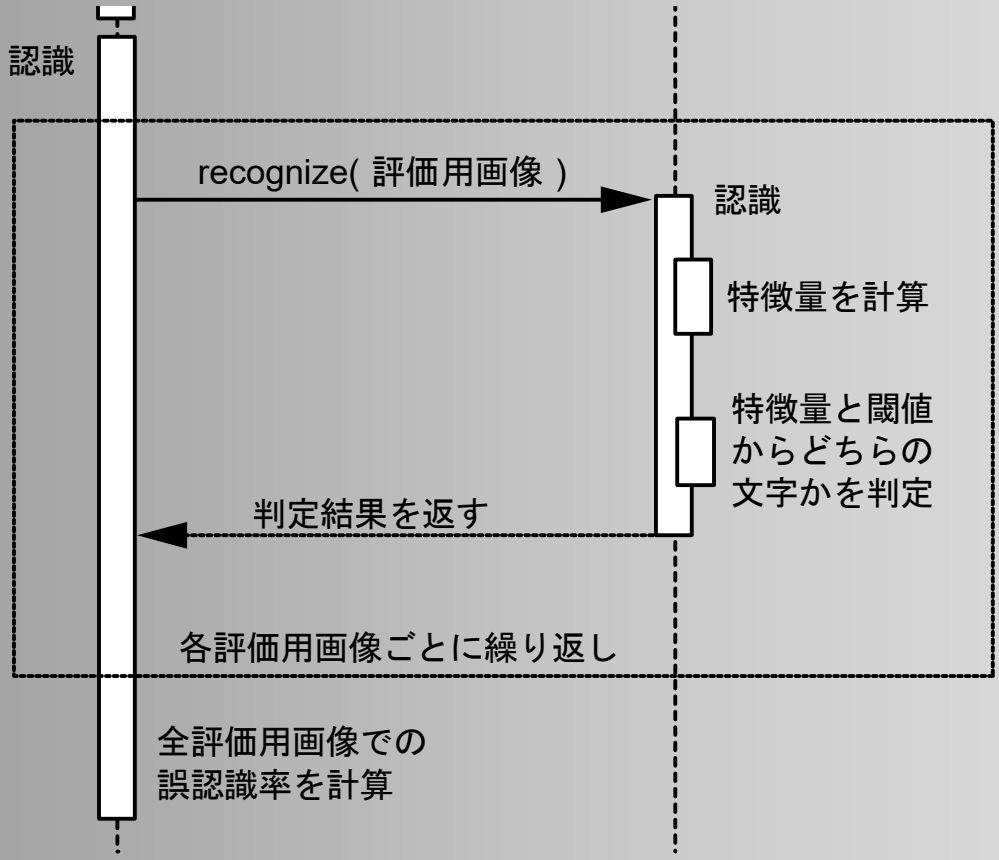
処理の流れの確認

- アプリケーションクラス
  - 学習・識別に使用する画像データをファイルから読み込む
  - 学習処理
  - 認識処理

シーケンス図

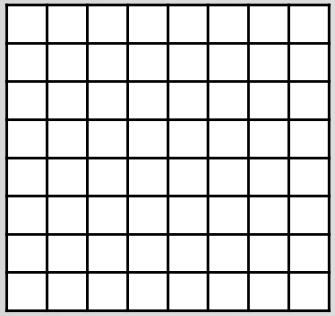

RecognitionApp
Character
Recognizer
初期化
init
loadSampleImages()
学習・認識用画像の読み込み
学習
train(学習用画像の配列)
学習
特徴量を計算
各学習用画像ごとに繰り返し
全画像の特徴量
から閾値を計算
認識

シーケンス図（続き）


認識
recognize(評価用画像)
認識
特徴量を計算
特徴量と閾値
からどちらの
文字かを判定
判定結果を返す
各評価用画像ごとに繰り返し
全評価用画像での
誤認識率を計算

画像クラス

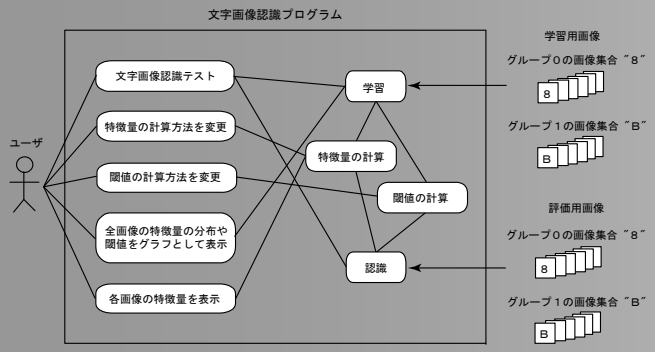
- Javaの画像クラスを使用
  - java.awt.Imageクラス
    - 内部のピクセルデータにアクセスできない
  - java.awt.image.BufferedImage クラス

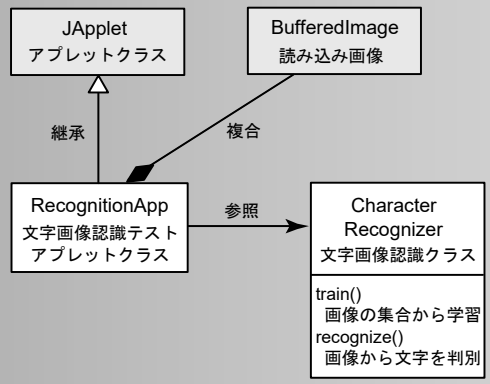


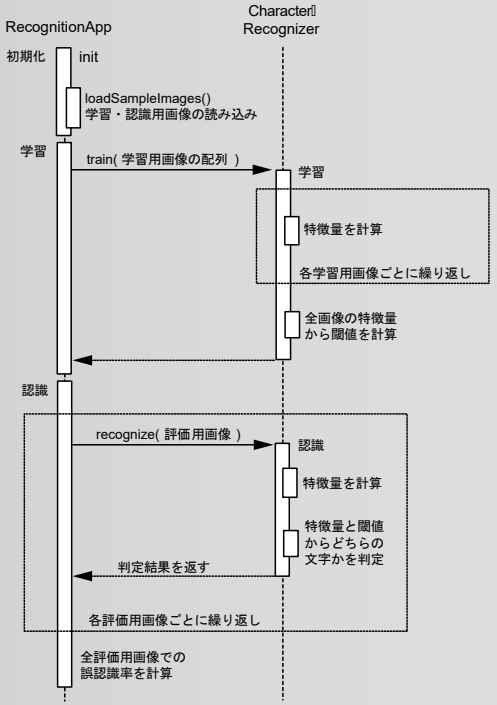
- 画像ファイルの読み込み
  - Applet.getImage() メソッド
    - gif, jpg, PNG のみ（BMPは読み込めない）
  - Image I/O ライブラリ
    - プラグインを追加すると BMP も読み込める
    - CLの端末にはプラグインを追加済み

UMLについて

- Unified Modeling Language（UML）
- オブジェクト指向でのソフトウェア設計を図で記述するときの描き方
  - 最近、広く使われつつある
  - UMLを知っておくと、コミュニケーションが円滑に進められる





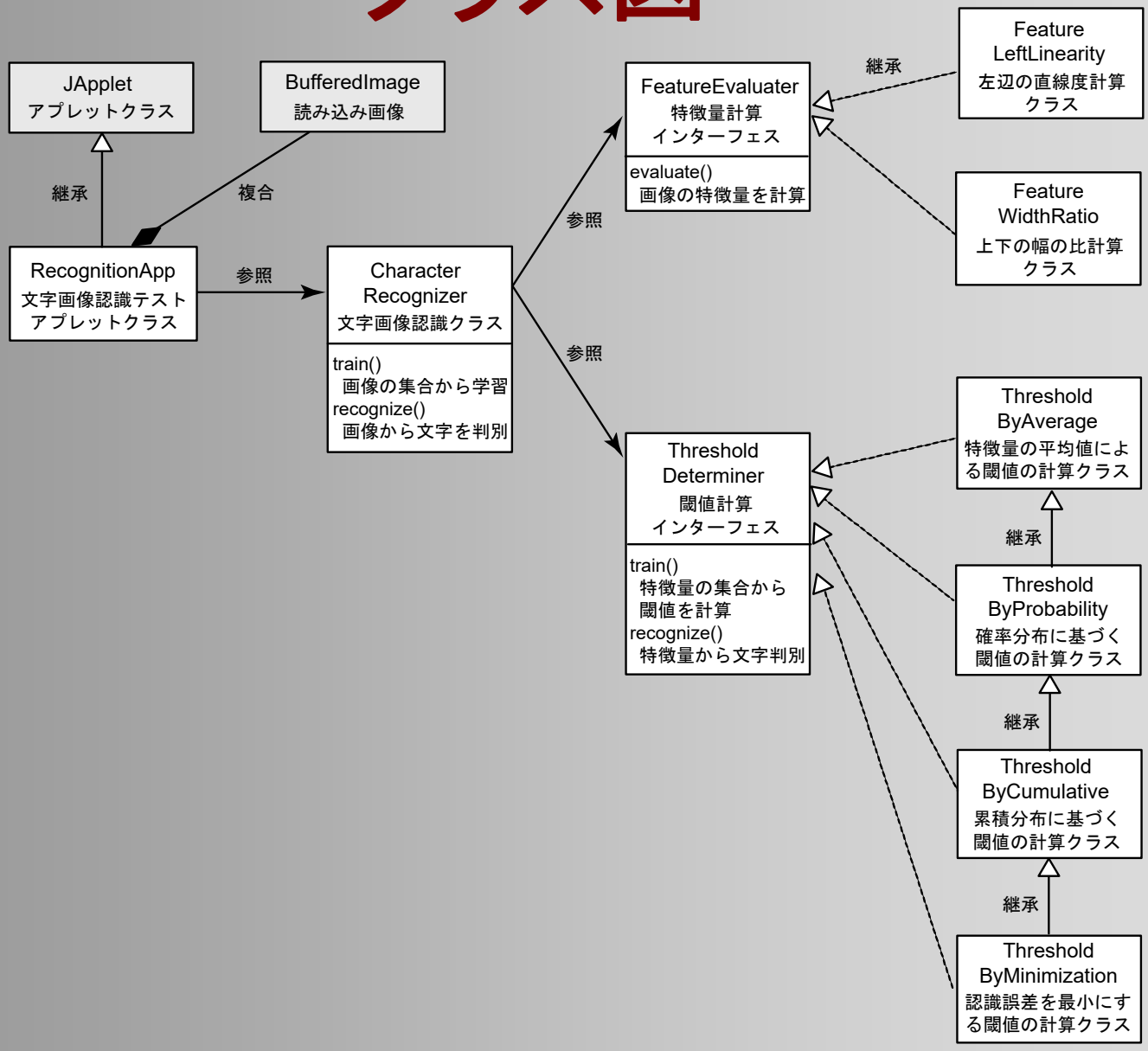


１次元の特徴量による認識

画像識別クラスの詳しい設計

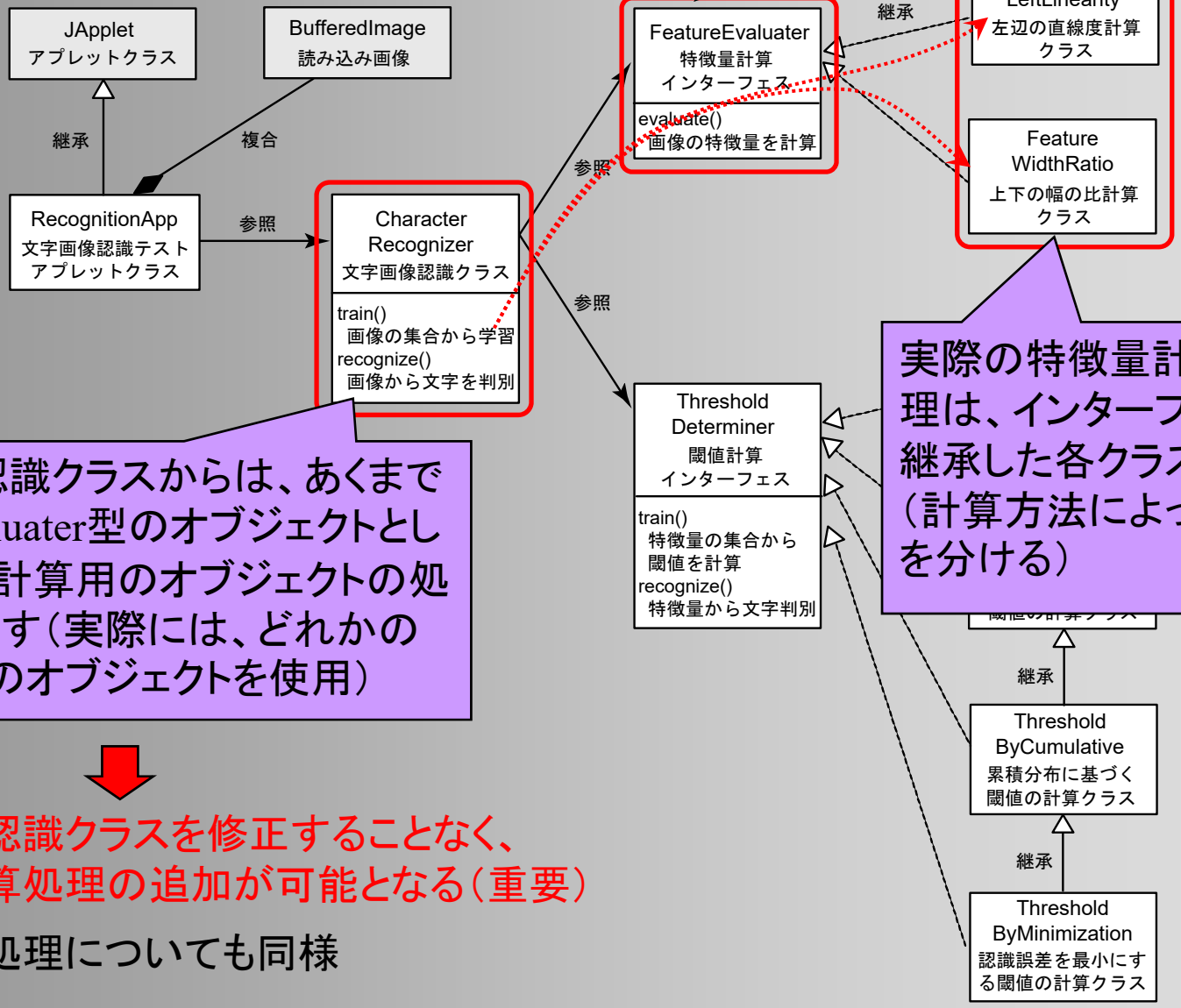
- 本プログラムでは、特徴量の計算方法や、閾値の計算方法を切り替えれるようにしたい
- 各計算処理を別のクラスに分離する
  - 各計算処理メソッドを持つ基底クラス（インターフェース）を定義する
  - 基底クラスを継承した、各計算処理に対応したクラスを定義する
  - 画像識別クラスでは、基底クラスのオブジェクトへの参照を持つ

クラス図


JApplet
アプレットクラス
BufferedImage
読み込み画像
継承
複合
RecognitionApp
文字画像認識テスト
アプレットクラス
参照
Character
Recognizer
文字画像認識クラス
train()
画像の集合から学習
recognize()
画像から文字を判別
参照
参照
FeatureEvaluater
特徴量計算
インターフェス
evaluate()
画像の特徴量を計算
継承
Feature
LeftLinearity
左辺の直線度計算
クラス
Feature
WidthRatio
上下の幅の比計算
クラス
Threshold
Determiner
閾値計算
インターフェス
train()
特徴量の集合から
閾値を計算
recognize()
特徴量から文字判別
Threshold
ByAverage
特徴量の平均値による閾値の計算クラス
継承
Threshold
ByProbability
確率分布に基づく
閾値の計算クラス
継承
Threshold
ByCumulative
累積分布に基づく
閾値の計算クラス
継承
Threshold
ByMinimization
認識誤差を最小にする閾値の計算クラス

クラス図

特徴量を計算する処理のインターフェースを定義（実際の処理はまだなし）



実際の特徴量計算の処理は、インターフェースを継承した各クラスで実装（計算方法によってクラスを分ける）

文字画像認識クラスからは、あくまでFeatureEvaluater型のオブジェクトとして、特徴量計算用のオブジェクトの処理を呼び出す（実際には、どれかのサブクラスのオブジェクトを使用）

文字画像認識クラスを修正することなく、特徴量計算処理の追加が可能となる（重要）

閾値計算処理についても同様

クラス定義

- 資料6～8ページのソースを参照
  - FeatureEvaluater, ThresholdDeterminer インターフェース
    - これらはメソッド定義のみで中身は持たない
  - CharacterRecognizer クラスは、FeatureEvaluater, ThresholdDeterminer オブジェクトへの参照
    - 2つのインターフェースのメソッドを呼び出して処理を実行
  - 実際には、FeatureEvaluater, ThresholdDeterminer を継承したいずれかのクラスのメソッドが呼び出される
    - どのクラスのメソッドが呼び出されるかは、外部から自由に設定できる
    - FeatureEvaluater, ThresholdDeterminer を継承したクラスであれば、どのクラスでも使用可能

```
//
//   文字画像の特徴量検出クラスのインターフェース
//
interface   FeatureEvaluater
{
        // 特徴量の名前を返す
        public String   getFeatureName();

        // 文字画像から１次元の特徴量を計算する
        public float    evaluate( BufferedImage image );

        // 最後に行った特徴量計算の結果を描画する
        public void    paintImageFeature( Graphics g );
}


//
//   閾値決定クラスのインターフェース
//
interface   ThresholdDeterminer
{
        // 閾値の決定方法の名前を返す
        public String   getThresholdName();

        // 両グループの特徴量から閾値を決定
        public void    determine( float[] features0, float[] features1 );

        // 与えられた特徴量からどちらの文字かを判定
        public int    recognize( float feature );

        // 特徴空間のデータをグラフに描画（グラフオブジェクトに図形データを設定）
        public void    drawGraph( GraphViewer gv );
}
```

資料 8ページ

```
//
//   文字画像の特徴量検出クラスのインターフェース
//
interface   FeatureEvaluater
{
        // 特徴量の名前を返す
        public String   getFeatureName();

        // 文字画像から１次元の特徴量を計算する
        public float    evaluate( BufferedImage image );

        // 最後に行った特徴量計算の結果を描画する
        public void     paintImageFeature( Graphics g );
}


                                    ーフェース
//
interface   ThresholdDeterminer
{
        // 閾値の決定方法の名前を返す
        public String   getThresholdName();

        // 両グループの特徴量から閾値を決定
        public void     determine( float[] features0, float[] features1 );

        // 与えられた特徴量からどちらの文字かを判定
        public int      recognize( float feature );

        // 特徴空間のデータをグラフに描画（グラフオブジェクトに図形データを設定）
        public void     drawGraph( GraphViewer gv );
}
```

メソッドの定義
ここでは定義のみ
実装は派生クラスで行う

interface ＝ メソッドの定義のみを行う特殊な class

資料 8ページ

```
//
//   文字画像認識クラス
//
class   CharacterRecognizer
{
        // 特徴量の評価用オブジェクト
        protected FeatureEvaluater      feature_evaluater;

        // 閾値の決定用オブジェクト
        protected ThresholdDeterminer   threshold_determinat

        // 入力画像の特徴量
        protected float   features0[];
        protected float   features1[];


        // 特徴量の評価用オブジェクトを設定
        public void   setFeatureEvaluater( FeatureEvaluater fe )
        {
                feature_evaluater = fe;
        }

        // 閾値の計算用オブジェクトを設定
        public void   setThresholdDeterminer( ThresholdDeterminer td )
        {
                threshold_determiner = td;
        }

        // 特徴量の評価用オブジェクトを取得
        public FeatureEvaluater   getFeatureEvaluater()
        {
                return   feature_evaluater;
        }

        // 閾値の計算用オブジェクトを取得
        public ThresholdDeterminer   getThresholdDeterminer()
        {
                return   threshold_determiner;
        }
```

特徴量を計算するためのオブジェクト
FeatureEvaluater型のオブジェクトとして扱う
（実際には、FeatureEvaluater型を派生（implements）したクラスのオブジェクトが設定される）

計算用オブジェクトの設定メソッド

資料 7ページ

```
        // 与えられた２グループの画像データを判別するような特徴量の閾値を計算
        public void   train( BufferedImage[] images0, BufferedImage[] images1 )
        {
                // 各入力画像の特徴量を計算
                features0 = new float[ images0.length ];
                features1 = new float[ images1.length ];
                for ( int i=0; i<images0.length; i++ )
                        features0[ i ] = feature_evaluater.evaluate( images0[ i ] );
                for ( int i=0; i<images1.length; i++ )
                        features1[ i ] = feature_evaluater.evaluate( images1[ i ] );

                // 特徴量の分布から２つのグループを識別するような閾値を決定
                threshold_determiner.determine( features0, features1 );
        }

        // 学習結果に基づいて与えられた画像を判別
        //（判別した画像の種類として 0 or 1 の値を返す）
        public int   recognize( BufferedImage image )
        {
                // 入力画像の特徴量を計算
                float   feature = feature_evaluater.evaluate( image );

                // 閾値に基づいてどちらのグループに属するかを判定
                return   threshold_determiner.recognize( feature );
        }
}
```

資料 7ページ

```
        // 与えられた２グループの画像データを判
        public void   train( BufferedImage[] images
        {
                // 各入力画像の特徴量を計算
                features0 = new float[ images0.le
                features1 = new float[ images1.le
                for ( int i=0; i<images0.length; i++ )
                        features0[ i ] = feature_evaluater.evaluate( images0[ i ] );
                for ( int i=0; i<images1.length; i++ )
                        features1[ i ] = feature_evaluater.evaluate( images1[ i ] );

                // 特徴量の分布から２つのグループを識別するような閾値を決定
                threshold_determiner.determine( features0, features1 );
        }

        // 学習結果に基づいて与えられた画像を判別
        // （判別した画像の種類として 0 or 1 の値を返す）
        public int   recognize( BufferedImage image )
        {
                // 入力画像の特徴量を計算
                float   feature = feature_evaluater.evaluate( image );

                // 閾値に基づいてどちらのグループに属するかを判定
                return   threshold_determiner.recognize( feature );
        }
}
```

FeatureEvaluater型のオブジェクトのメソッドを呼び出して、特徴量を計算（実際には、FeatureEvaluater型を派生（implements）したクラスのメソッドが呼ばれる）

資料 7ページ

```
//
//   文字画像の左辺の直線度を特徴量として計算するクラス
//
class   FeatureLeftLinerity implements FeatureEvaluater
{
        // FeatureEvaluater で定義されたメソッドを実装
}


//
//   閾値決定クラスのインターフェース
//
class   ThreshouldByAverage implements ThresholdDeterminer
{
        // ThresholdDeterminer で定義されたメソッドを実装
}
```

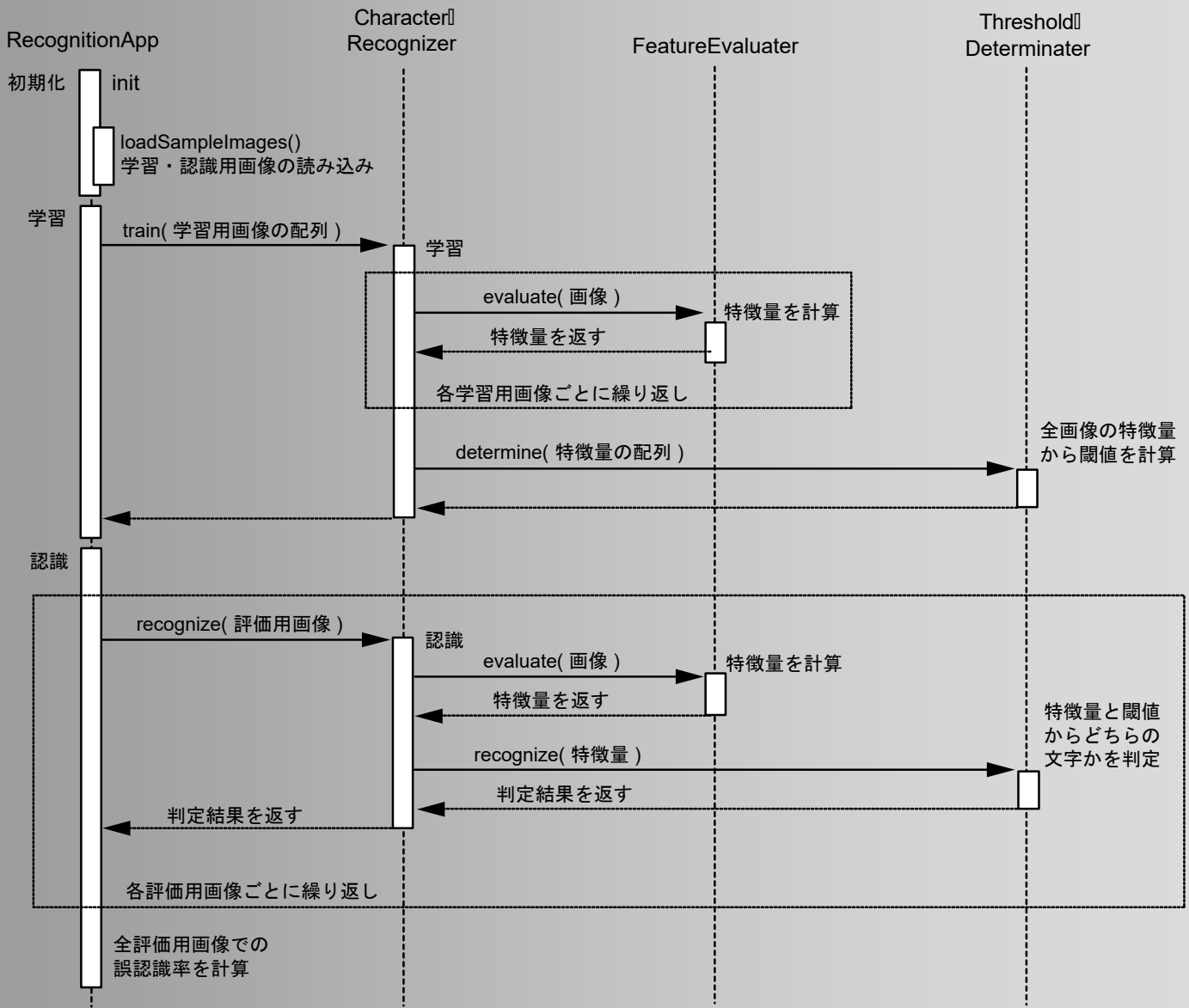
実際の処理は、FeatureEvaluater型を派生（implements）したクラスのメソッドに記述する

資料 8ページ

```
//
//   文字画像認識テスト アプレット
//
public class RecognitionApp extends JApplet
{
        // 初期化処理
        public void   init()
        {
                // 文字画像判別モジュールの生成
                recognizer = new CharacterRecognizer();

                // 特徴量計算モジュール、閾値計算モジュールを設定
                recognizer.setFeatureEvaluater( new FeatureLeftLinerity() );
                recognizer.setThresholdDeterminer(new ThreshouldByAverage() );

                // 後は変更なし
        }
}
```

資料 6ページ

シーケンス図


RecognitionApp
Character Recognizer
FeatureEvaluater
Threshold Determinater
初期化
init
loadSampleImages()
学習・認識用画像の読み込み
学習
train(学習用画像の配列)
学習
evaluate(画像)
特徴量を計算
特徴量を返す
各学習用画像ごとに繰り返し
全画像の特徴量から閾値を計算
determine(特徴量の配列)
認識
recognize(評価用画像)
認識
evaluate(画像)
特徴量を計算
特徴量を返す
特徴量と閾値からどちらの文字かを判定
recognize(特徴量)
判定結果を返す
判定結果を返す
各評価用画像ごとに繰り返し
全評価用画像での誤認識率を計算

継承を使用するメリット

- 文字認識を行うクラスと、個別の特徴量や閾値を計算するクラスが分離できるので、別々に実装できる
- 文字認識クラス（CharacterRecognizer）を変更することなく、特徴量や閾値の計算方法を拡張できる

継承を使用しない例

- 資料9ページのプログラムを参照
  - 全て CharacterRecognizer のみで実現する例
    - どの計算方法を使うかを変数で記録しておく
    - 設定されている計算方法に応じた処理を実行
  - 計算方法が増えるたびに、CharacterRecognizer を書き換える必要があり、どんどん複雑になっていってしまう

外部インターフェースを使用しない CharacterRecognizer クラスの設計の例

```
//
//   文字画像認識クラス
//
class   CharacterRecognizer
{
        // 特徴量の評価処理にどの計算方法を使うかの番号
        protected int   feature_evaluater_no;

        // 閾値の決定処理にどの計算方法を使うかの番号
        protected int   threshold_determinater_no;

        // 文字画像から 1 次元の特徴量を計算する
        public float   evaluate( BufferedImage image )
        {
                // feature_evaluater_no により計算方法を分ける
                if ( feature_evaluater_no == 1 )
                {
                        // 方法 1
                }
                else if ( feature_evaluater_no == 2 )
                {
                        // 方法 2
                }
                else if ( ...
        }

        // 最後に行った特徴量計算の結果を描画する
        public void   paintImageFeature( Graphics g )
        {
                // 上と同様
        }

        // 両グループの特徴量から閾値を決定
        public void   determine( float[] features0, float[] features1 )
        {
                // threshold_determinater_no により計算方法を分
                if ( feature_evaluater_no == 1 )
                {
                        // 方法 1
                }
                else if ( feature_evaluater_no == 2 )
                {
                        // 方法 2
                }
                else if ( ...
        }

        // 与えられた特徴量からどちらの文字かを判定
        public int   recognize( float feature )
        {
                // 上と同様
        }

        // 特徴空間のデータをグラフに描画（グラフオブジェクトに図形データを設定）
        public void   drawGraph( GraphViewer gv )
        {
                // 上と同様
        }
}
```

```
// 特徴量の評価処理にどの計算方法を使うかの番号
protected int   feature_evaluater_no;

// 閾値の決定処理にどの計算方法を使うかの番号
protected int   threshold_determinater_no;

// 文字画像から 1 次元の特徴量を計算する
public float   evaluate( BufferedImage image )
{
        // feature_evaluater_no により計算方法を分ける
        if ( feature_evaluater_no == 1 )
        {
                // 方法 1
        }
        else if ( feature_evaluater_no == 2 )
        {
                // 方法 2
        }
        else if ( ...
}
```

資料 9ページ

継承の利用

- 抽象的なインターフェースを定義・継承
  - 呼び出し側は、そのインターフェースに従って処理を実行する
    - 呼び出し側は、呼び出される側の詳しい実装を知る必要はない
  - 呼び出される側は、インターフェースを継承し、適切な方法で処理の中身を実装する
- ポリモーフィズム（多態性）
  - オブジェクト指向言語の最も重要な機能

インターフェースについて

- 複数の意味で使われているので注意
- 広い意味でのインターフェース
  - クラスのメソッド定義など
  - クラスの機能がどのように呼び出されるかを規定
- 狭い意味でのインターフェース
  - Javaのインターフェース
  - メソッド定義のみで、実装を持たない基底クラス
- ユーザーインターフェース

デザインパターン

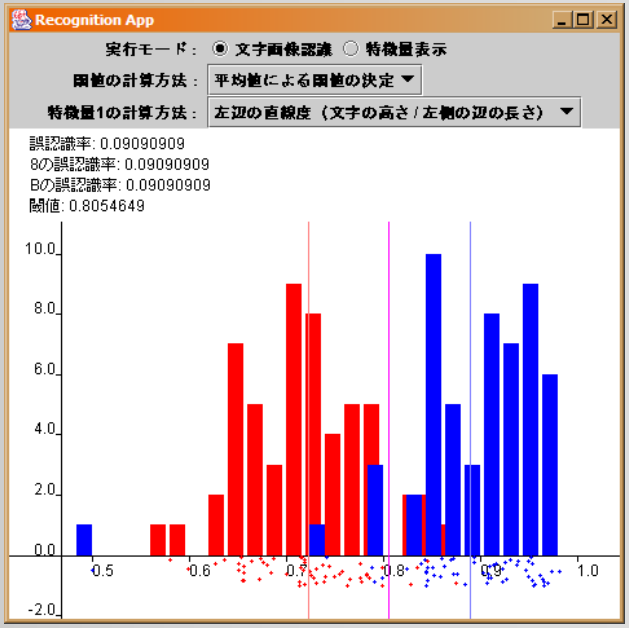
- デザインパターン
  - オブジェクト指向による開発では、複数のクラス同士が連携して大きな機能を実現する
  - よく使われる組み合わせのパターンをまとめたものがデザインパターン（23個のパターンが代表的）
  - デザインパターンを知っていると、設計を整理する参考になったり、他の設計者とのコミュニケーションがうまくいったりする
- 今回の例は、ストラテジ パターンと呼ばれる
  - 複数の選択肢がある処理を、インターフェースを用いて別クラスに分離するパターン

2次元の特徴量への拡張

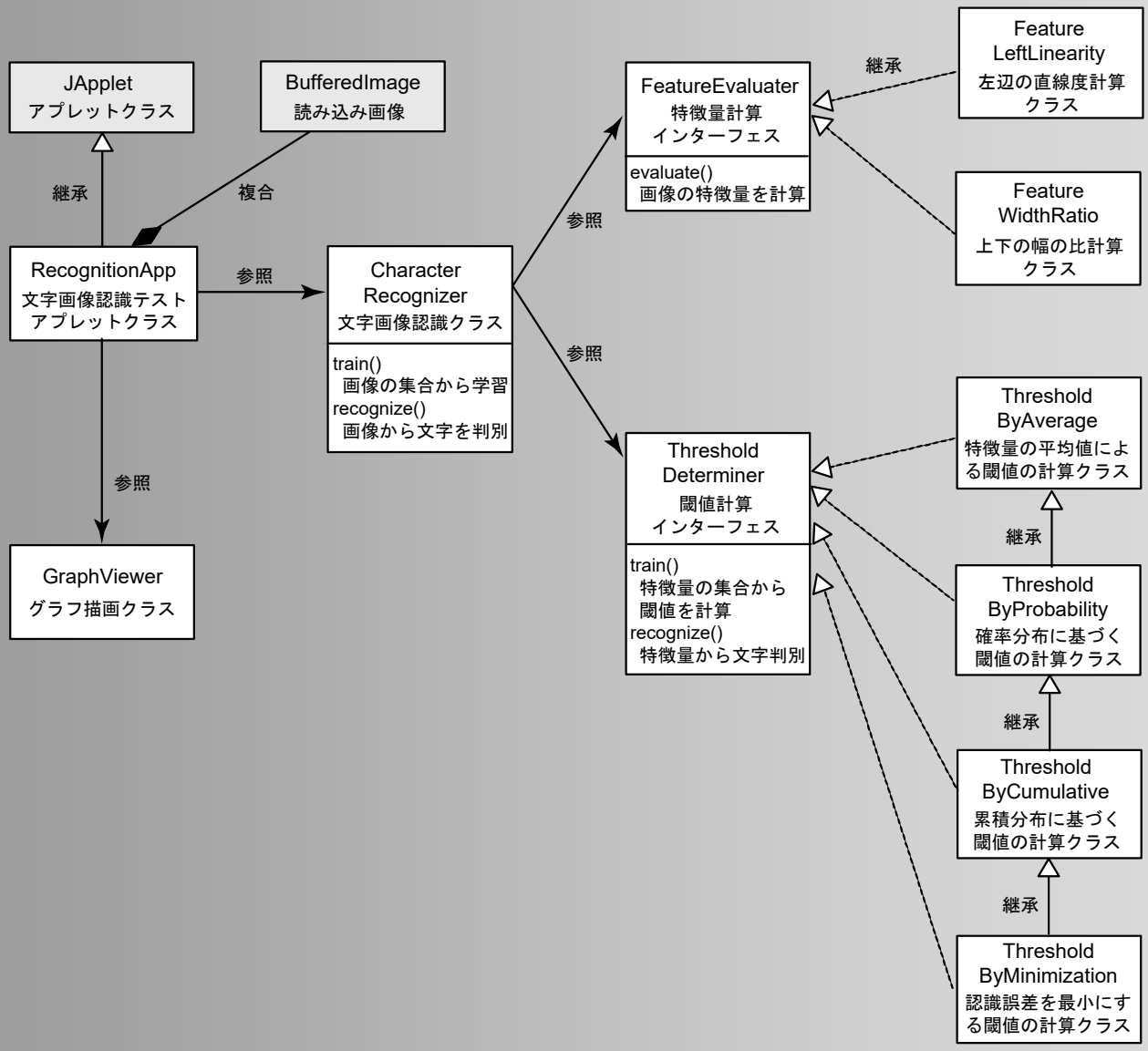
- ここまで説明したクラス設計は、1つの特徴量のみを使った画像認識
- 後で、2つの特徴量を使った画像認識に拡張する
- 変更の必要があるところ
  - 特徴量の計算はそのままで問題ない
  - 閾値の計算に、2つの特徴量を入力できるように変更する必要がある

グラフの描画

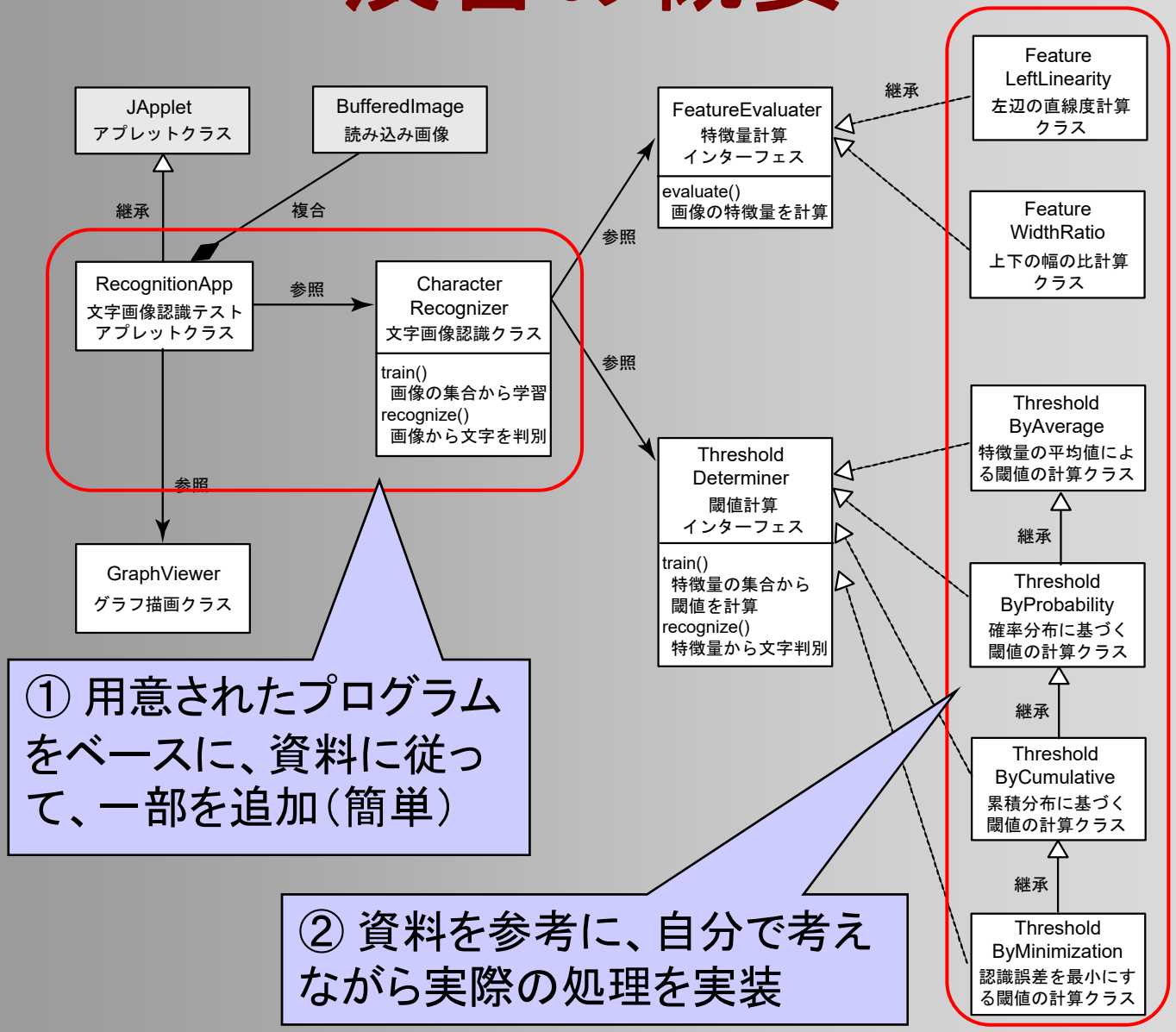
- グラフを描く機能を持ったクラスを別に定義
  - 散布図、棒グラフ、折れ線、直線などのグラフに最低限必要な図形を描画
  - 描画範囲を自動的に決定
- 用意してあるクラスを使用して良い



設計のまとめ


JApplet
アプレットクラス
BufferedImage
読み込み画像
継承
複合
RecognitionApp
文字画像認識テスト
アプレットクラス
参照
Character
Recognizer
文字画像認識クラス
train()
画像の集合から学習
recognize()
画像から文字を判別
参照
GraphViewer
グラフ描画クラス
参照
参照
FeatureEvaluater
特徴量計算
インターフェス
evaluate()
画像の特徴量を計算
継承
Feature
LeftLinearity
左辺の直線度計算
クラス
Feature
WidthRatio
上下の幅の比計算
クラス
Threshold
Determiner
閾値計算
インターフェス
train()
特徴量の集合から
閾値を計算
recognize()
特徴量から文字判別
Threshold
ByAverage
特徴量の平均値による閾値の計算クラス
継承
Threshold
ByProbability
確率分布に基づく
閾値の計算クラス
継承
Threshold
ByCumulative
累積分布に基づく
閾値の計算クラス
継承
Threshold
ByMinimization
認識誤差を最小にする閾値の計算クラス

演習の概要


JApplet
アプレットクラス
BufferedImage
読み込み画像
継承
複合
RecognitionApp
文字画像認識テスト
アプレットクラス
参照
Character
Recognizer
文字画像認識クラス
train()
画像の集合から学習
recognize()
画像から文字を判別
参照
GraphViewer
グラフ描画クラス
FeatureEvaluater
特徴量計算
インターフェス
evaluate()
画像の特徴量を計算
参照
参照
Threshold
Determiner
閾値計算
インターフェス
train()
特徴量の集合から
閾値を計算
recognize()
特徴量から文字判別
継承
Feature
LeftLinearity
左辺の直線度計算
クラス
Feature
WidthRatio
上下の幅の比計算
クラス
Threshold
ByAverage
特徴量の平均値による閾値の計算クラス
継承
Threshold
ByProbability
確率分布に基づく
閾値の計算クラス
継承
Threshold
ByCumulative
累積分布に基づく
閾値の計算クラス
継承
Threshold
ByMinimization
認識誤差を最小にする閾値の計算クラス
① 用意されたプログラムをベースに、資料に従って、一部を追加（簡単）
② 資料を参考に、自分で考えながら実際の処理を実装

統合環境 Eclipse の利用

統合環境 Eclipse の利用

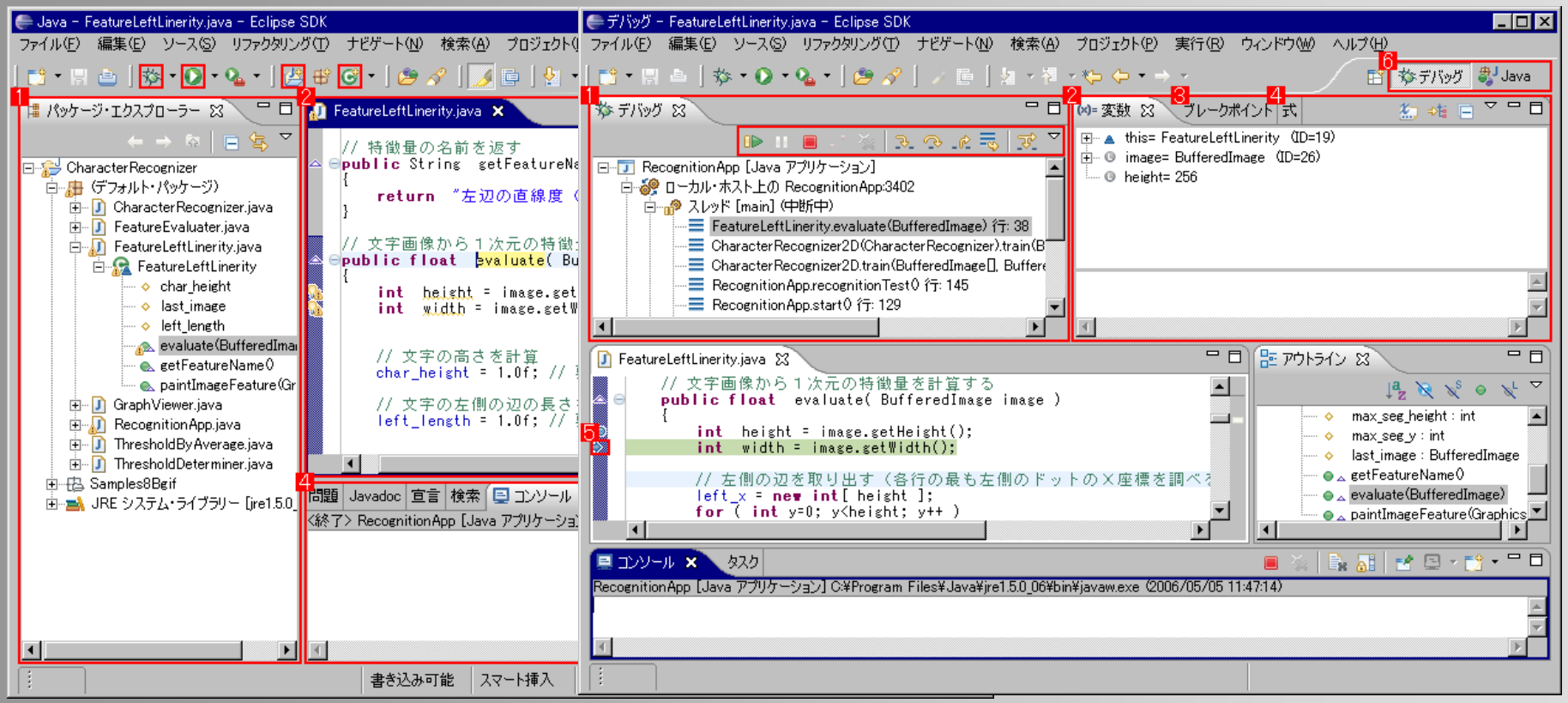
- 本演習では、Java プログラムを作成するための開発環境として、Eclipse を利用
- これまでの講義・演習と同じように、コマンドラインのコンパイラを使っても構わない
- 「Eclipseを使ったJavaプログラミング」
  - http://www.cg.ces.kyutech.ac.jp/lecture/project/eclipse/index.html
  - 講義のページからもリンク有り

統合環境を使うメリット

- これまでの開発環境
  - テキストエディタを使って、Javaプログラムを記述
  - コマンドラインから javac コマンドでコンパイル
  - デバッグを行うときは、プログラム中にデバッグ用のプログラムを記述（変数の値を print 等）
- 統合環境
  - プログラムの記述、コンパイル、デバッグをひとつのソフトウェア上で行える
  - ステップ実行・変数ウォッチなど、デバッグのための便利な機能が利用できる

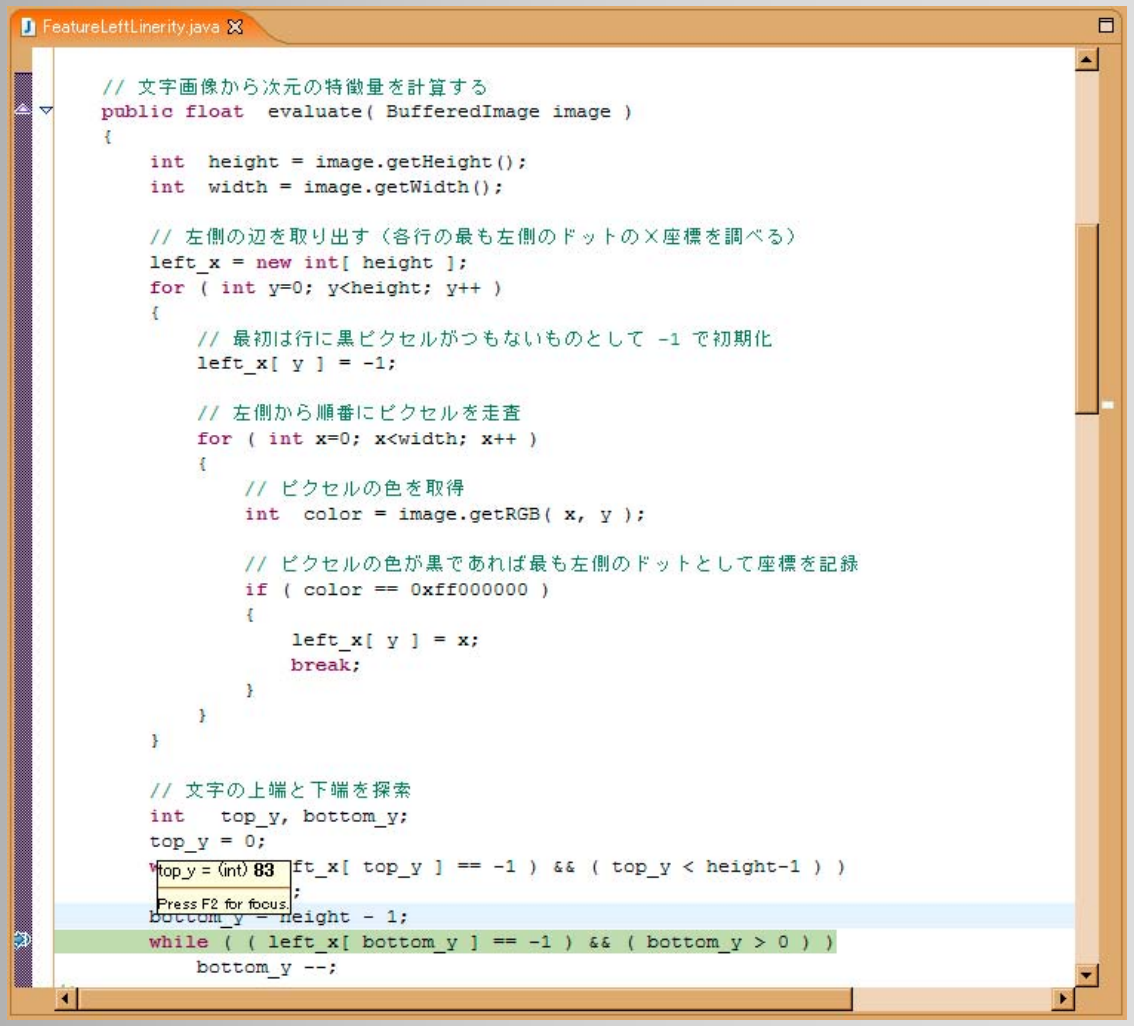
Eclipseの画面イメージ

- エディットビュー、デバッグビュー



Eclipseを使ったデバッグ

- プログラムをステップ実行可能
- 実行中のプログラムの変数の値が見れる



統合環境を使うときの注意

- 一般のソフトウェア開発では、統合環境が使われている
  - 企業などでは、統合環境を使いこなして開発を行うことが期待される
- Javaの機能と統合環境の機能を混同しないようにすることが必要
- 統合環境の全機能を使いこなすのは困難
  - とりあえず、プログラムを作成するための最低限の機能を使えるところから始めれば良い

Eclipseの使用方法

- プロジェクト
  - あるプログラムを構成するソースファイルの情報や、実行のための設定情報をまとめたもの
- Eclipse の使用の流れ
  - Eclipse の起動
  - プロジェクトを作成
    - デフォルトのワークスペースは Z:¥workspace
  - プログラムを記述
    - コンパイルはバックグラウンドで自動的に行われる
  - 実行、デバッグ

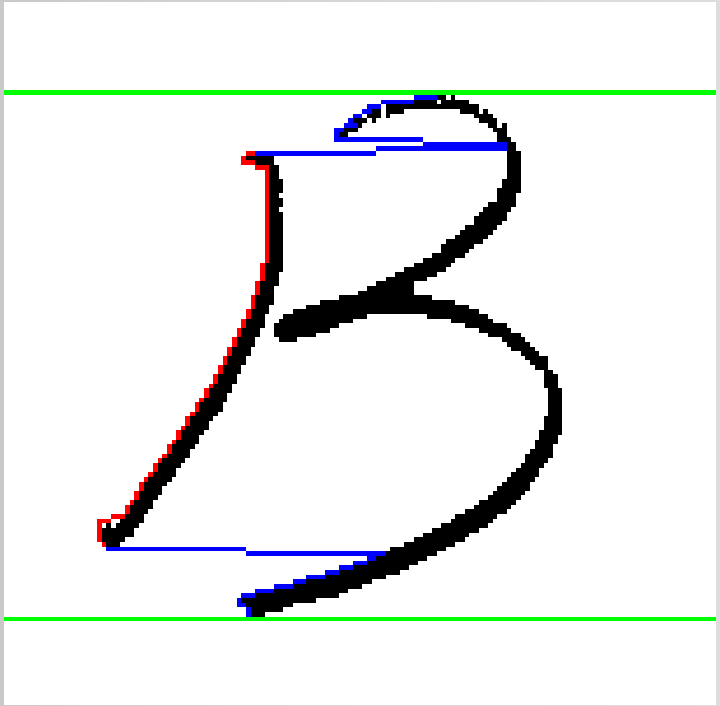
プログラムの実装

本日の演習手順

- プログラム全体の実装
  - サンプルプログラムのコンパイルテスト
  - 初期化処理、ユーザーインターフェースの実装の確認
  - サンプルリストを参考に、画像の読み込み処理 、文字認識のテスト処理の実装
- 1つ目の特徴量の計算
  - 資料の5.1～5.3節まで（できれば5.4節も）
- 1つ目の閾値の計算
  - 資料の6.1節の内容を実装

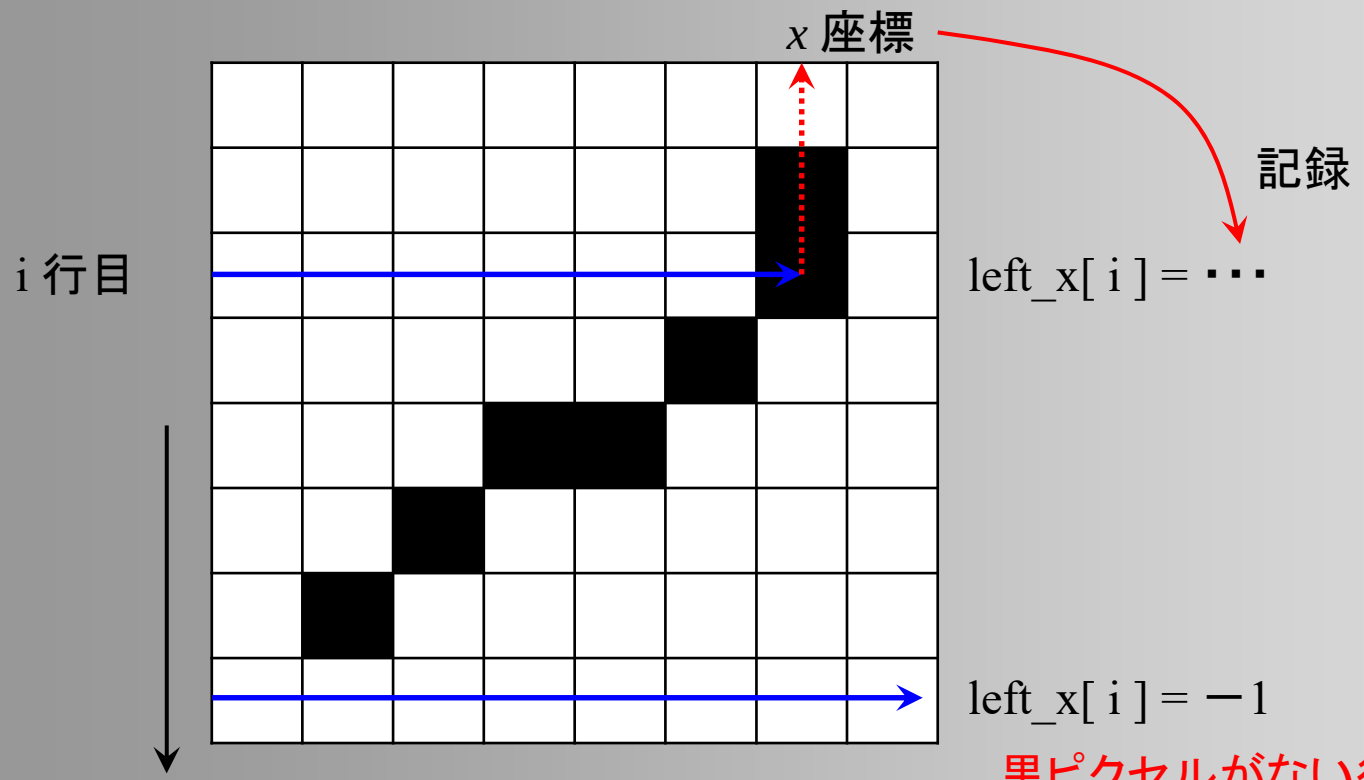
1つ目の特徴量の計算

- 左辺の高さ ／ 左辺の長さ
  - 各行ごとの左辺のX座標を探索して記録
    - 下図の青線＋赤線に相当
  - 左辺の高さを計算
    - 図の緑線の y 座標の差
  - 左辺の長さを計算
    - 左辺の各線分の和



1. 左辺のX座標を探索して記録

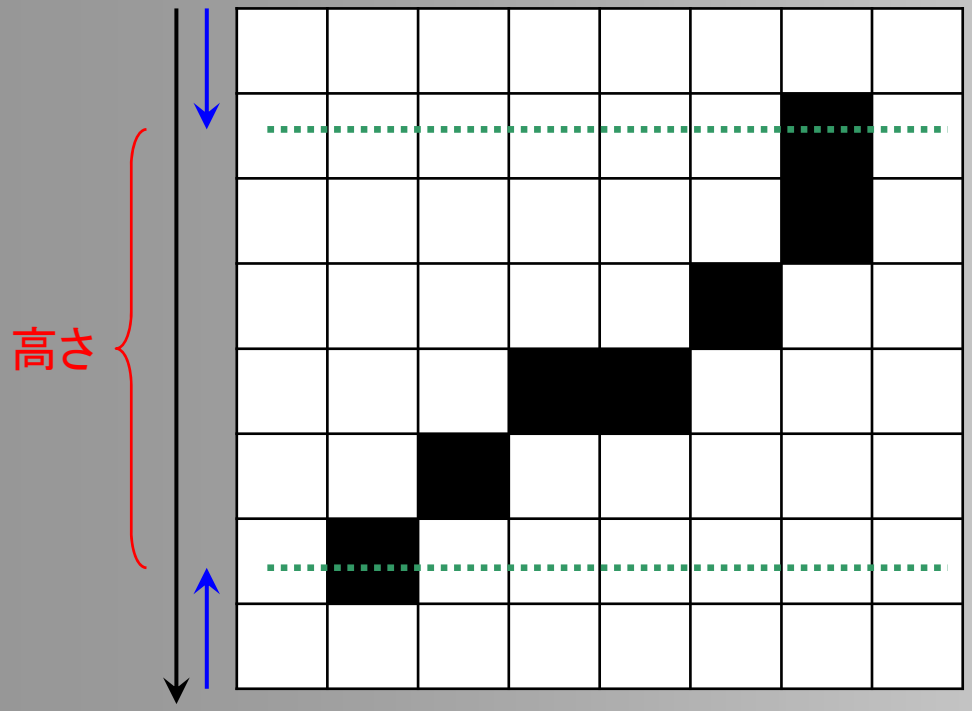
- 最初に黒ピクセルのある x 座標を記録



2. 左辺の高さを計算

- 最初と最後に黒ピクセルがある行の y 座標から高さを計算

height = 2つの y 座標の差から計算



left_x[0] = −1

left_x[1] = ・・・ → 最初に黒ピクセルがある行（y=1）

left_x[2] = ・・・

left_x[3] = ・・・

left_x[4] = ・・・

left_x[5] = ・・・

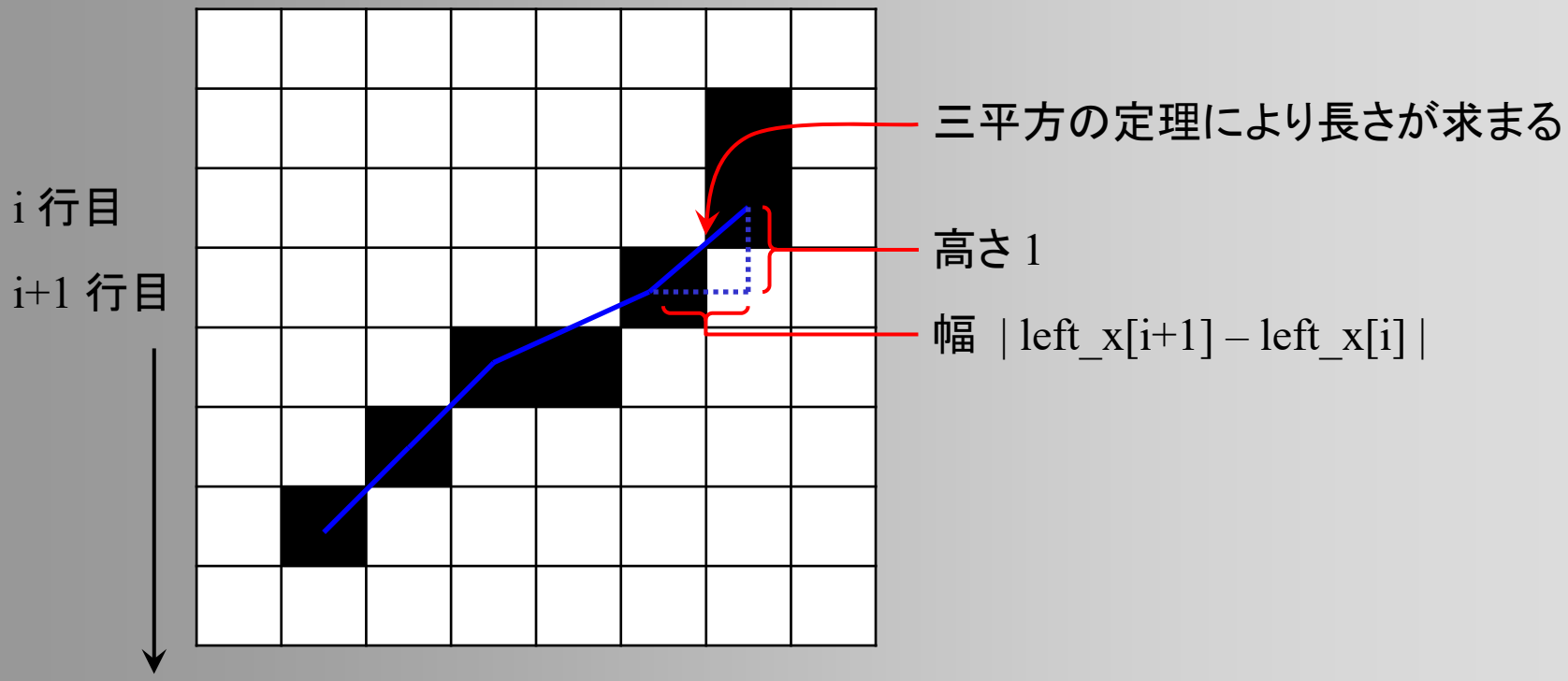
left_x[6] = ・・・ → 最後に黒ピクセルがある行（y=6）

left_x[7] = −1

※ 下から探索したとき、最初に黒ピクセルがある行

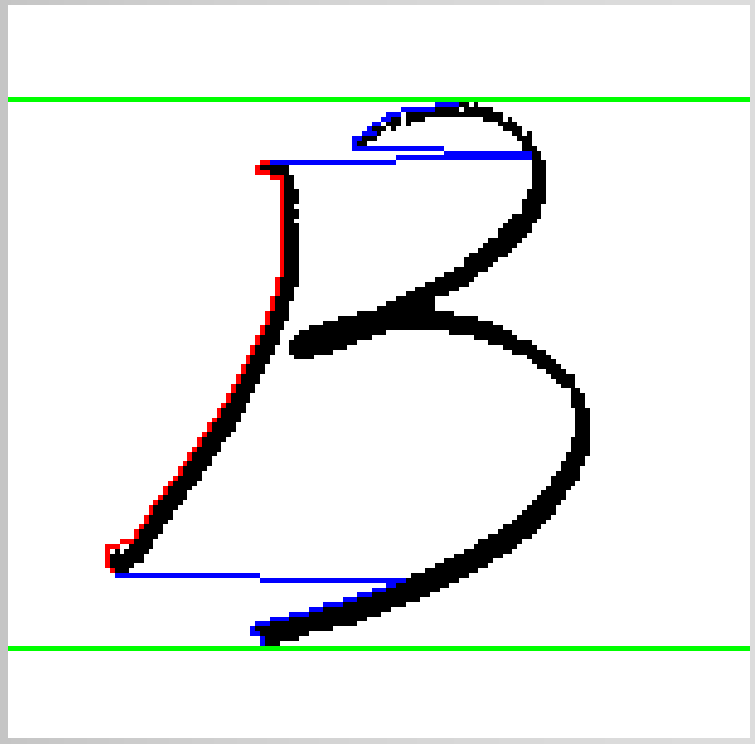
3. 辺の長さを計算

- 折れ線の長さの和により辺の長さを計算



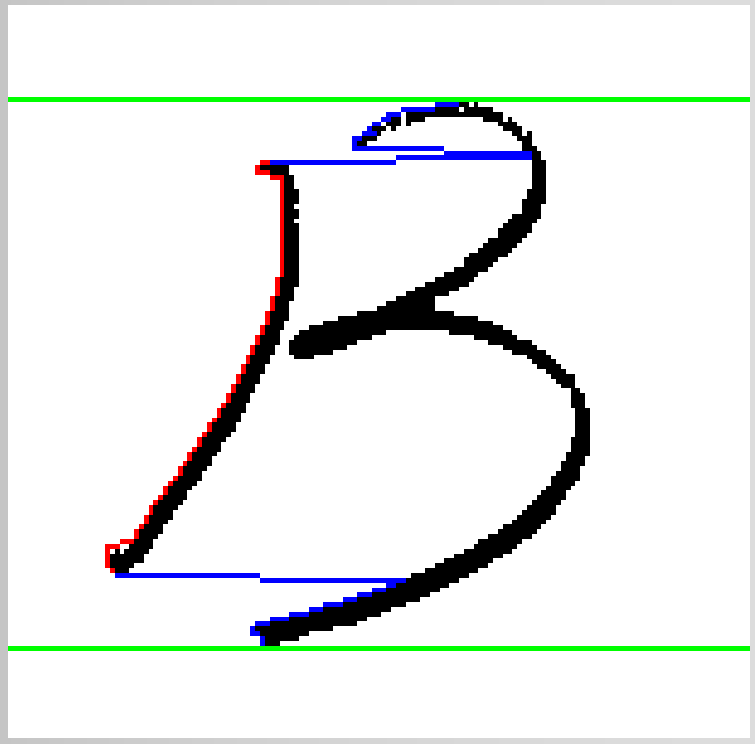
特徴量の計算結果の表示

- 特徴量計算の確認のために、抽出した情報を描画する機能を追加すると、デバッグがやりやすくなる
  - paintImageFeature()
    メソッドに処理を追加



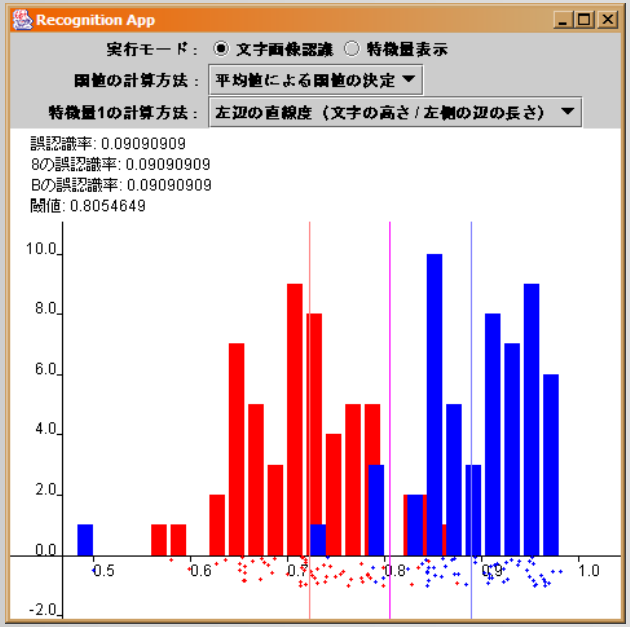
途切れた辺への対策

- いくつかの方法が考えられるので、各自工夫する
  - 左辺を複数の範囲に分け、最も長い範囲のみを抽出
  - あらかじめ左辺のX座標をフィルタリングして平滑化する
  - 単純に、一定以上のギャップは無視する



閾値の計算

- 平均値に基づく閾値
  - ヒストグラムの計算の実装
- 出現確率が等しくなる閾値
- 誤認識率が等しくなる閾値
- 誤認識率が最小になる閾値



閾値の計算の実装

- 平均値に基づく閾値
  - 資料の説明に従って実装（簡単）
- 他の閾値の計算
  - 次回説明

次回予定

- 今回の演習までは終わっているという前提で、残りの内容を簡単に説明
  - 残りの3つの閾値の計算方法
  - 2次元の認識への設計の拡張
  - 2つ目の特徴量の実装
  - （Eclipseを使ったデバッグ）
  - プレゼンテーションの準備方法
- 基本的には、資料を参考に、各自、演習を進める

学科端末室（CL）利用案内

- Computer Laboratory
  - 研究棟 西棟 6階 北側
  - CL I（W604/W605）: 端末33台（授業中以外利用可能）
  - 平日のみ（週末・休日は閉室）
    - 金曜日18:00～月曜日8:00 の間は鍵が閉まる。

CL端末 利用方法

- アカウントは情報科学センタと共通
- Windows 8.1 Pro
  - ソフトウェア環境は情報科学センタやマルチメディア講義室の端末とは異なる。
- 自分のデータは、ホームドライブ（Z:）に置く
  - C: ドライブに置いたファイルは終了時に消える。
  - デスクトップに置いたファイルは残る。
- トラブル等があれば、E526（技術職員室）へ

CL端末室 利用の注意

- 飲食厳禁
- 冷房を入れるときにドア・窓を閉めること
- 最後に出る人は、冷房を止め、窓を閉める
  - 窓を開けたままにすると雨や鳩が部屋に入る

- 利用方法を守れない場合、違反者は利用を禁止する(＝単位が取れなくなる)ことがあるので、十分注意すること