

クラスの活用

モデリングとシミュレーション

2020年度

1

目的

- 基本的クラスの利用
 - 講義で使用するライブラリ
- 抽象クラスの利用
 - インターフェースの利用

Javaの豊富なライブラリ

- ▶ プログラミングで共通的に必要なライブラリが、言語とともに配布されている。
- ▶ 良く使うライブラリを紹介し、利用方法を学ぶ
- ▶ マニュアル
 - ▶ <https://www.oracle.com/jp/java/technologies/javase/documentation/api-jsp.html>

原始型と対応するクラス

- ▶ クラスでない型
 - ▶ int、double、boolean、char、など
 - ▶ ポインタが存在しないことに注意
- ▶ 対応するクラス
 - ▶ Integer、Double、Boolean、Character、など
- ▶ マニュアルを確認

例 : Integer

- <https://docs.oracle.com/javase/jp/11/docs/api/java.base/java/lang/Integer.html>

原始型と対応するクラス

- ▶ Boxing : 原始型からクラスへの代入

```
int x = 1;  
Integer y = x;
```

- ▶ Unboxing : クラスから原始型への代入

```
Integer y = 1;  
int x = y;
```

文字列クラスString

- ▶ C/C++のように、文字の配列ではない
 - ▶ ダブルクォーテーションを使う
 - ▶ 比較、部分文字列などのメソッド
 - ▶ 生成方法

```
String str = "文字列";
```
- ▶ マニュアルを確認

抽象クラス

- Abstract Class
 - メソッドの一部が実装されていない
 - abstract キーワードのついたメソッド
 - インスタンスを生成できない
- 継承クラスで抽象メソッドを実装
- 一つのクラスしか継承(extend)できない

抽象クラス

- ▶ Interface
 - ▶ 少数のメソッドのみを持つ
 - ▶ 他のクラスから呼ばれる方法を定義
- ▶ 複数のinterfaceを実装(implement)できる
 - ▶ 未実装のメソッドを実装

抽象クラス リストを例に

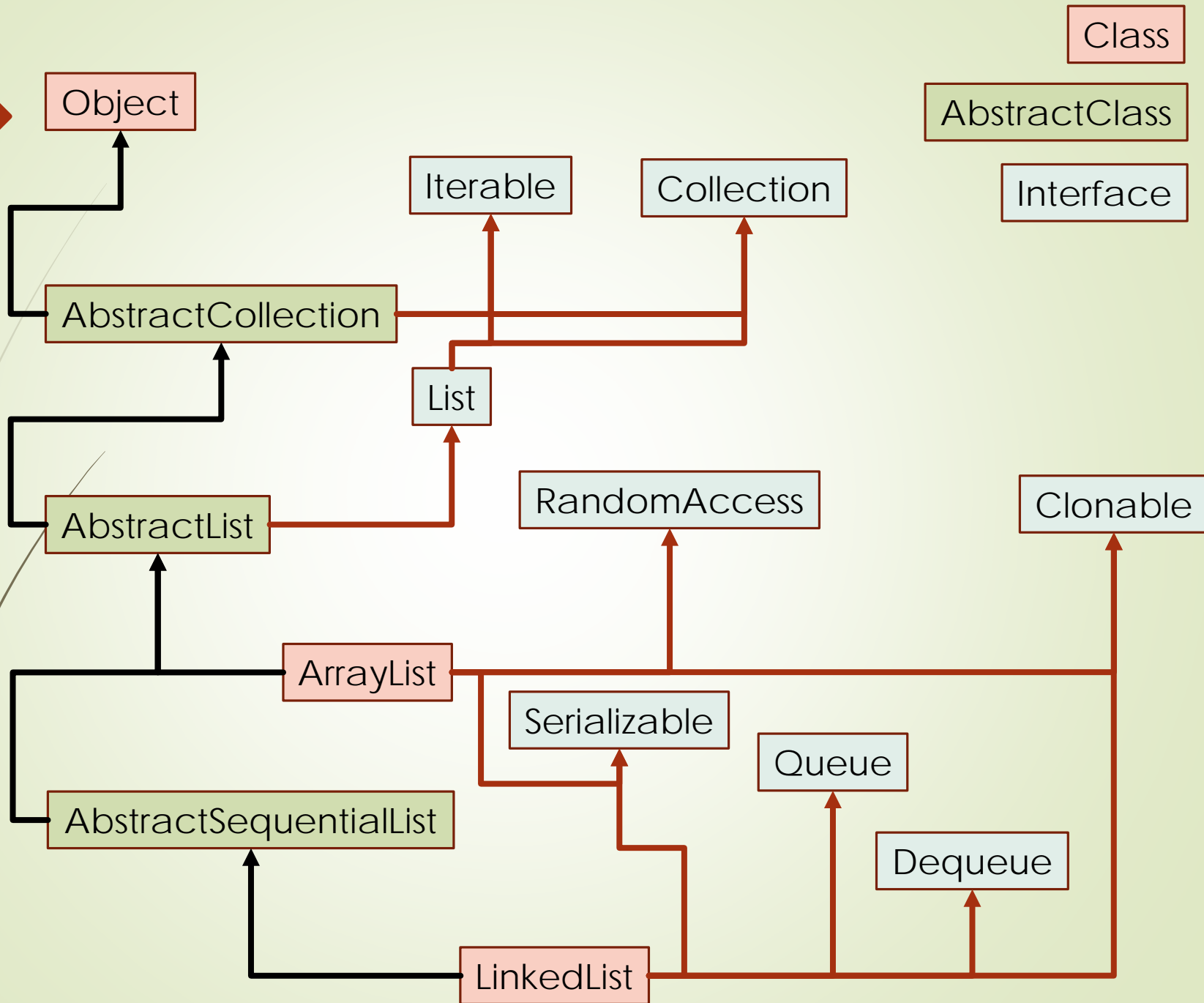
- ▶ `java.util.List`
 - ▶ リストを操作するメソッドを定義するインターフェース
 - ▶ リストを操作する方法のみを定義
 - ▶ 要素の追加・削除、リストの大きさ
- ▶ マニュアルを確認

抽象クラス リストを例に

- ▶ `java.util.AbstractList`
 - ▶ `List` インターフェイスを実装
 - ▶ ランダムアクセス的リストの抽象クラス
 - ▶ `get` メソッドが実装されていない
- ▶ マニュアルを確認

リストの実装クラス

- ▶ `java.util.ArrayList`
 - ▶ 検索が高速
 - ▶ `AbstractList`の拡張クラス
- ▶ `java.util.LinkedList`
 - ▶ データの挿入・追加が高速
 - ▶ `AbstractSequentialList`の拡張クラス



リストの利用

➡ 生成方法1

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

➡ 生成方法2

```
List<T> list = Collections.synchronizedList(new ArrayList<>());
```

➡ スレッド間での同期を指定

リストの利用

- 要素追加 : `list.add(T t);`
- 要素取り出し : `T t = list.get(int j);`
- 要素再設定 : `list.set(int j, T t);`
- 要素の数 : `list.size()`

型パラメタ

- クラスが扱う対象を特定せずにパラメタ化できる

```
1. public class Store<T> { Tは特定のクラスではない
2.     List<T> list;
3.     public Store(List<T> list){this.list=list;}
4.     public boolean addData(T t){return list.add(t);}
5. }
```


型パラメタの利点

Listの場合

- ▶ List<Integer> list
- ▶ listに保存できるのは、クラスIntegerのインスタンスのみ
- ▶ listから取り出したものは、クラスIntegerのインスタンス

インターフェースの例

Comparable

- ▶ Comparableは
 - ▶ IntegerやStringなどに実装済み
- ▶ Comparableを前提に整列するメソッドを持つクラスがある

//クラスDataは相互に比較できる

```
public class Data implements Comparable<Data>{
    private int x;

    public Data(int x) {
        this.x = x;
    }

    @Override
    public int compareTo(Data arg0) {
        return this.x - arg0.x;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Data a = new Data(1);
        Data b = new Data(5);
        System.out.println(a.compareTo(b));
    }
}
```

Comparableインターフェース
は比較対象を指定する

型パラメータを持つメソッド generic method

- 汎用的なメソッド（サブルーチン）
 - 対象となる具体的な型を特定せず、その特性（インターフェース）が分かれば処理できる
 - 例えば、整列は比較できるモノであれば整列できる

- 戻り値の前で型パラメタを定義
- ここでは、TはTと比較できるクラスであること
 - TのインスタンスはcompareTo()を持っている→これを使って比較できる

```
public static <T extends Comparable<T>> void sort(List<T> list) {  
    //ソートの実装  
}
```

listの各要素がcompareTo()を持つことを利用

NetBeans 利用tips

- ソースファイルの自動整形
 - ソースファイル編集画面でマウス右ボタン→「format」
- 変数名等の変更
 - ソースファイル編集画面でマウス右ボタン→「refactoring」→「rename」
 - プロジェクト全体に変更が行われる

- ▶ コードの挿入
 - ▶ ソースファイル編集画面でマウス右ボタン→「insert code」
 - ▶ コンストラクタ、設定メソッド、取得メソッドなど
- ▶ メソッドや変数名の自動補完
 - ▶ インスタンスの後にピリオド
- ▶ 行番号に出るバルーン
 - ▶ マウスを合わせてメッセージ表示
 - ▶ マウス左ボタンで、対応策提案