第7回　継承とインターフェイス

～オブジェクト指向の真髄に挑戦！～

学習目標

- 継承とインターフェイスを利用したプログラムが書ける
  - 継承を使う利点を説明できる
  - Java で継承を使ったプログラムが書ける
  - ポリモーフィズムを使ったプログラムが書ける

7.1.　継承

7.1.1.　配列リストと連結リストの性能比較

今回は、配列リストと連結リストの追加の性能を比較してみたいと思います。前回までに作ったクラス群を利用して書いた性能を比較するプログラムを例題 7-1 に示します。

リスト 7-1：配列、連結リストの性能比較(Example7_1.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-1：配列、連結リストの性能比較
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example7_1 {
9:
10:       /**
11:       * メイン
12:       * 1 万件の種類をリストに追加して、その時間を計る
13:       */
14:       public static void main(String[] args) {
15:
16:         long startTime;//開始時間を保存する
17:         long endTime;  //終了時間を保存する
18:
19:         //-------配列リストの性能を測定する-------
20:         //商品種類配列リストを生成する
21:         ItemTypeArrayList itemTypeArrayList = new ItemTypeArrayList();
22:
23:         startTime = System.currentTimeMillis();//開始時間を測定する
24:
25:         //商品種類を 1 万件登録する
26:         for(int i=0;i<10000;i++){
27:           itemTypeArrayList.add(new ItemType(1001,"コーラ",120));
28:         }
29:
30:         endTime = System.currentTimeMillis();//終了時間を測定する
31:
32:         System.out.println("かかった時間は"+ (endTime - startTime) + "ミリ秒です");
33:
34:         //-------連結リストの性能を測定する-------
35:         //商品種類連結リストを生成する
36:         ItemTypeLinkedList itemTypeLinkedList = new ItemTypeLinkedList();
37:
38:         startTime = System.currentTimeMillis();//開始時間を測定する
39:
```

```
40:         //商品種類を 1 万件登録する
41:         for(int i=0;i<10000;i++){
42:           itemTypeLinkedList.add(new ItemType(1001,"コーラ",120));
43:         }
44:
45:         endTime = System.currentTimeMillis();//終了時間を測定する
46:
47:         System.out.println("かかった時間は"+ (endTime - startTime) + "ミリ秒です");
48:       }
49:     }
```

例題 7-1：配列、連結リストの性能比較(ItemTypeArrayList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-1：配列、連結リストの性能比較
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品種類配列リストクラス
7:      */
8:      public class ItemTypeArrayList {
9:
10:       private int ARRAY_SIZE = 10000;
11:
12:       private ItemType[] itemTypeArray = new ItemType[ARRAY_SIZE];  //商品種類を保存す
るための配列
13:
14:       /**
15:       * 商品種類を追加する
16:       */
17:       public void add(ItemType addItemType){
18:         //商品種類が入っていない箱を探す
19:         for(int i=0;i<ARRAY_SIZE;i++){
20:           if(itemTypeArray[i] == null){//入っていない
21:             itemTypeArray[i] = addItemType;//書き込む
22:             break;
23:           }
24:         }
25:       }
26:
27:       /**
28:       * 商品種類リストを表示する
29:       */
30:       public void display(){
31:         for(int i=0;i<ARRAY_SIZE;i++){
32:           if(itemTypeArray[i] != null){//商品種類が入っている
33:
System.out.println(itemTypeArray[i].getId()+":"+itemTypeArray[i].getName()+":"+itemTypeA
rray[i].getPrice()+"は販売中です");
34:           }
35:         }
36:       }
```

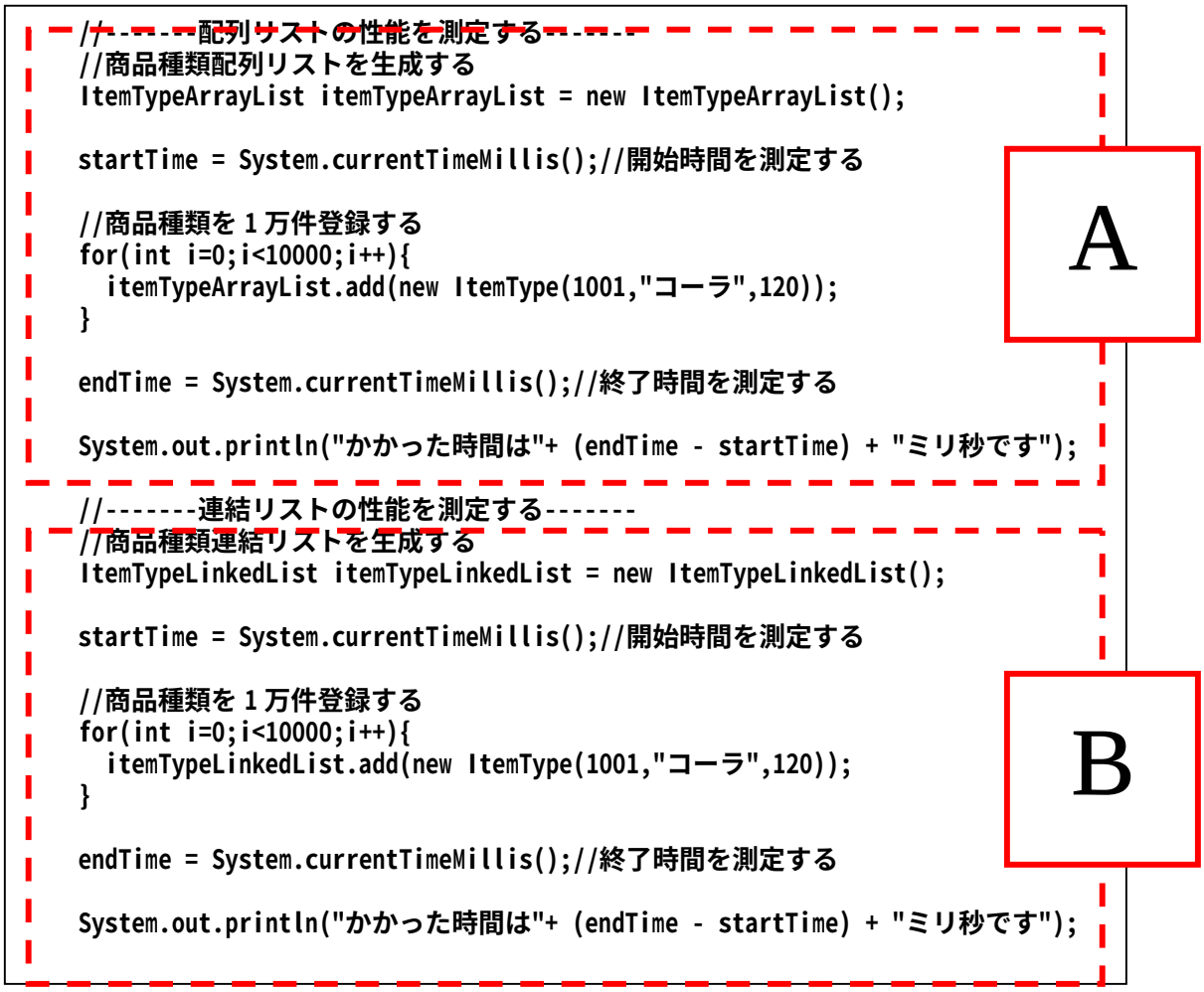
定数は一まとめに

● プログラム解説

System.currentTimeMills();メソッドは、現在の時刻をミリ秒単位で取得できるメソッドです。返り値は long 型で、仕事の前と後に測って引き算をすればかかった時間が得られるというわけです。詳しくは第 8 回を参照のこと。

①.ほとんど同じコードが 2 度かかれている

例題 7-1 は、正しく動作するプログラムですが、Example7_1.java のメインプログラムにほとんど同じコードが 2 度かかれているのが気になるところです。下に Example7_1.java の問題のコードを載せます。



②.どうしたらメソッド化できるか

重複コードは、直ちにメソッド化したいものです。しかし、意味が異なる場合は、メソッド化できません。プログラムの意味を考えてみましょう。

＜考えよう！＞A,B のプログラムの意味を考えてみよう

A の部分のプログラムの意味	B の部分のプログラムの意味

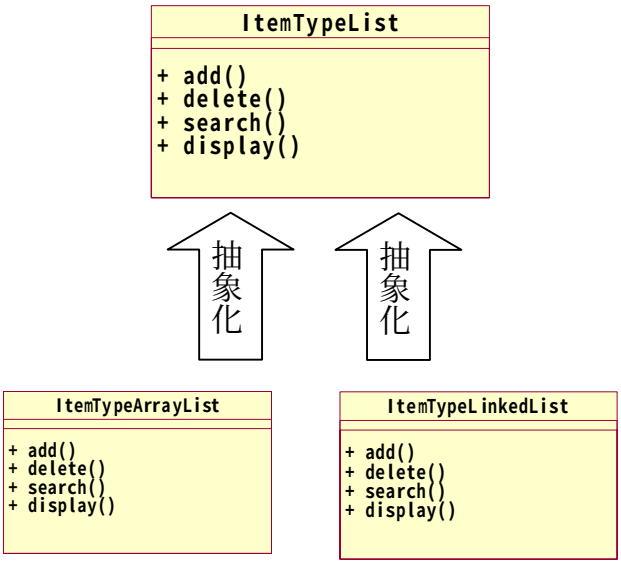
＜考えよう！＞何が異なるためにメソッド化できないのか？

7.1.2.　継承

①.継承

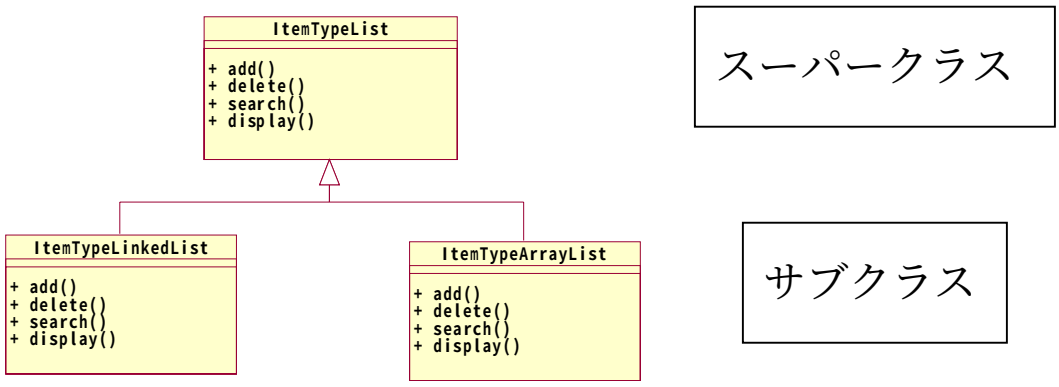
共通の意味を持つクラスを「抽象化」して、共通の意味をもつ新しいクラスを作るのが、「継承」の考え方です。

Java ではそのような場合には「ItemTypeLinkedList クラスと ItemTypeArrayList クラスは、ItemTypeList クラスを継承している」と言います。



②.継承をクラス図で表現する

クラスの継承関係をクラス図で表現する場合、以下のようになります。



クラス間に上記のような矢印を書きます。こうすると

ItemTypeLinkedList クラスは ItemTypeList クラスを継承している。
ItemTypeArrayList クラスは ItemTypeList クラスを継承している。

ということをクラス図で表現できます。

また、継承されたクラスと継承したクラスには、スーパークラスとサブクラスという呼び名がつきます。上記のクラス図を例にすれば、以下のような関係になります。

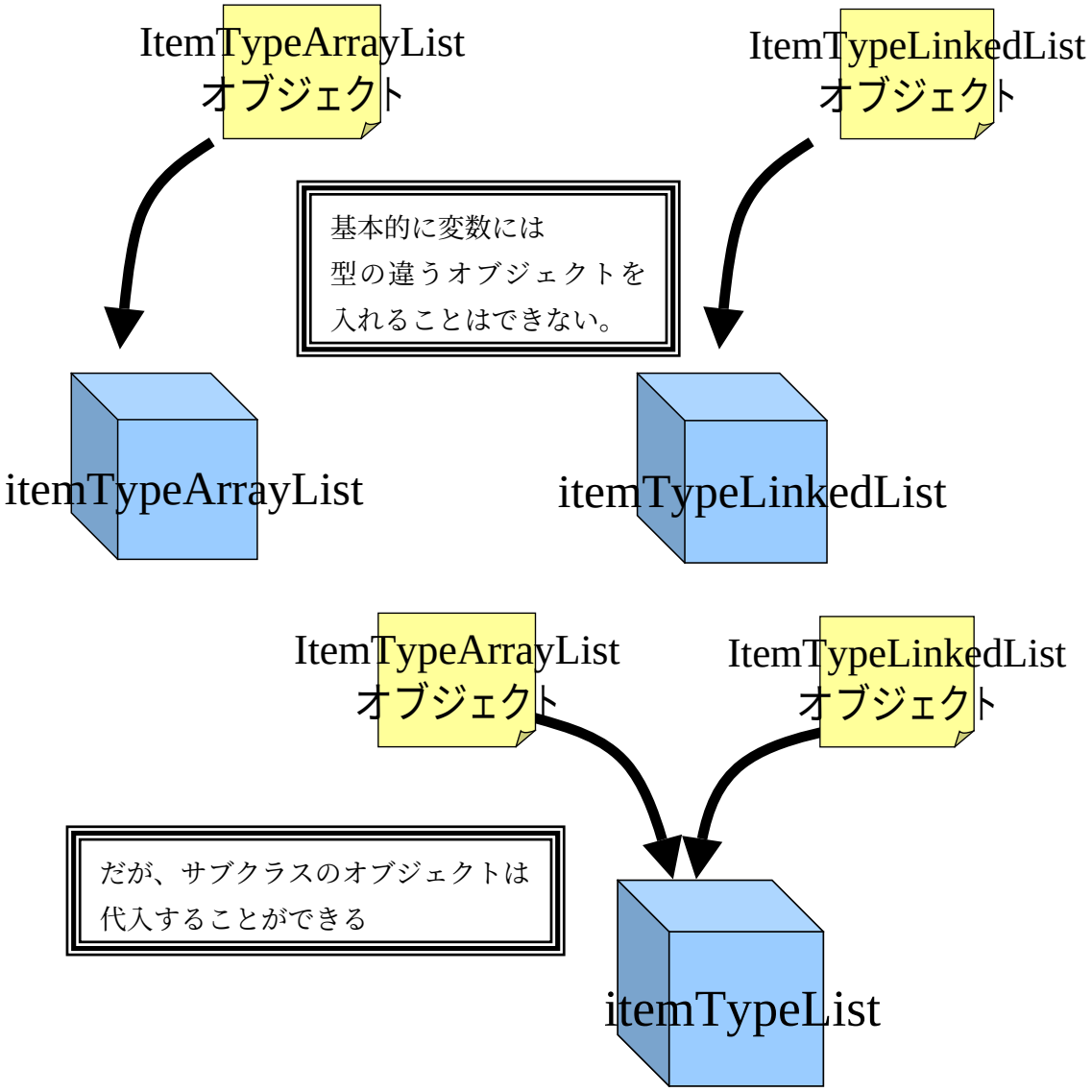
ItemTypeList クラスのサブクラスが ItemTypeArrayList クラスと ItemTypeLinkedList クラス。
ItemTypeArrayList クラスと ItemTypeLinkedList クラスのスーパークラスが ItemTypeList クラ

7.1.3.　Java で継承を使ったプログラムを書く

①.スーパークラスの変数への代入

これまでは、変数の代入は「同じ型から同じ型への代入」しかできませんでした。例えば ItemTypeArrayList クラスの変数には、ItemTypeArrayList のオブジェクトしか入れることができませんでした。

ところが、継承関係にあるオブジェクトはこの原則を少し外れます。サブクラスのオブジェクトは、スーパークラスの変数に代入することができるのです。



だから以下のプログラムはエラーではありません。

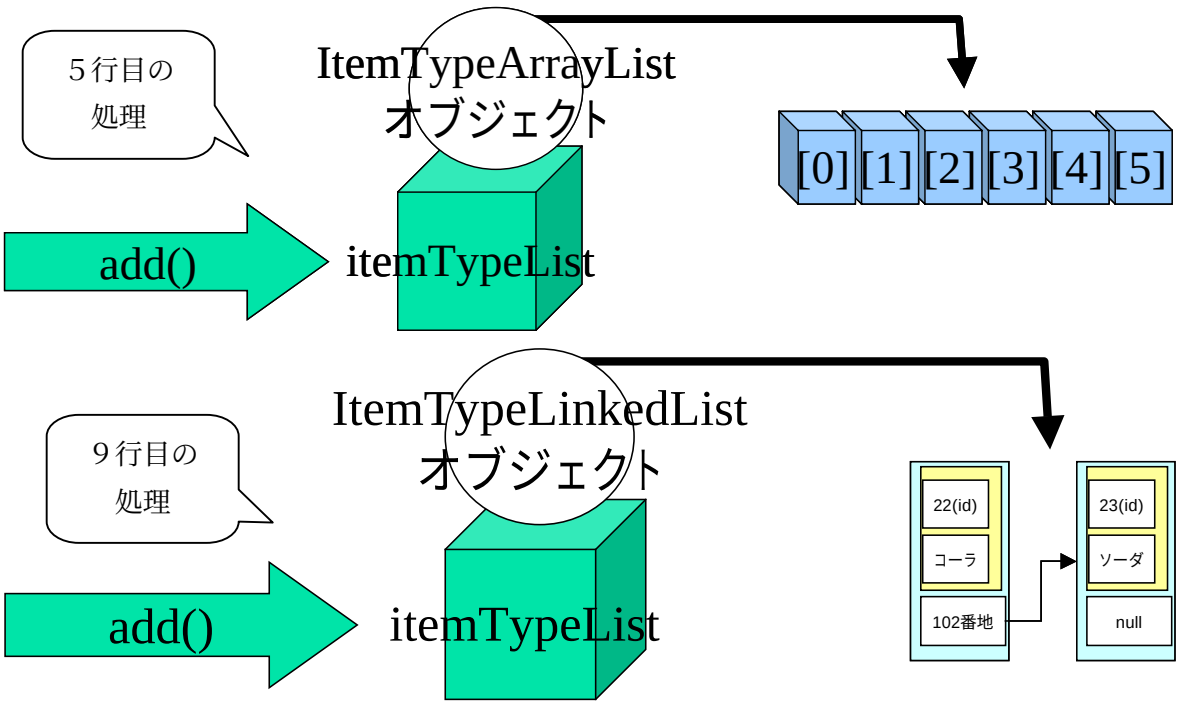
```
ItemTypeList itemTypeList;//スーパークラスの変数を用意する。

// サブクラスのオブジェクトを、スーパークラスの変数に代入する。
itemTypeList = new ItemTypeLinkedList();
itemTypeList = new ItemTypeArrayList();
```

よって、以下のようなプログラムが可能になります。

```
1:      ItemTypeList itemTypeList;//スーパークラスの変数を用意する
2:
3:      //---配列版---
4:      itemTypeList = new ItemTypeArrayList();//配列版オブジェクトを代入
5:      itemTypeList.add(new ItemType(1001,"コーラ")); //配列版オブジェクトに追加されます
6:
7:      //---連結リスト版---
8:      itemTypeList = new ItemTypeLinkedList();//連結リスト版オブジェクトを代入
9:      itemTypeList.add(new ItemType(1001,"コーラ")); //連結リスト版オブジェクトに追加されます
```

５行目と９行目は全く同じコードになっていますが、実際にメソッドが実行されている対象は変わっています。５行目の時点で itemTypeList に入っているのは ItemTypeArrayList オブジェクトなので、それに対して add メソッドが呼び出されています。一方９行目では ItemTypeLinkedList オブジェクトが入っているので、それに対して add メソッドが呼び出されることになるのです。



②.Java で継承を使ったプログラムを書く

例題 7-2：継承を使う(Example7_2.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-2：継承を使う
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example7_2 {
9:
10:       /**
11:       * メイン
12:       * 1 万件の種類をリストに追加して、その時間を計る
13:       */
14:       public static void main(String[] args) {
15:
16:         //配列リストの性能を測定する
17:         performanceTest(new ItemTypeArrayList());
18:
19:         //連結リストの性能を測定する
20:         performanceTest(new ItemTypeLinkedList());
21:
22:       }
23:
24:       /**
25:       * 商品種類リストの性能を測る
26:       */
27:       private static void performanceTest(ItemTypeList itemTypeList){
28:         long startTime;//開始時間を保存する
29:         long endTime;  //終了時間を保存する
30:
31:         startTime = System.currentTimeMillis();//開始時間を測定する
32:
33:         //商品種類を 1 万件登録する
34:         for(int i=0;i<10000;i++){
35:           itemTypeList.add(new ItemType(1001,"コーラ",120));
36:         }
37:
38:         endTime = System.currentTimeMillis();//終了時間を測定する
39:
40:         System.out.println("かかった時間は"+ (endTime - startTime) + "ミリ秒です");
41:       }
42:
43:     }
```

例題 7-2：継承を使う(ItemTypeList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-2：継承を使う
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品リストクラス
7:      */
8:      public class ItemTypeList {
9:
10:       /**
11:       * 商品種類を登録する
12:       */
13:       public void add(ItemType addItemType){
14:       }
15:
16:       /**
17:       * 商品種類リストを表示する
18:       */
19:       public void display(){
20:       }
21:
22:     }
```

例題 7-2：継承を使う(ItemTypeArrayList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-2：継承を使う
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品種類配列リストクラス
7:      */
8:      public class ItemTypeArrayList extends ItemTypeList{
9:
10:       private int ARRAY_SIZE = 10000;
11:
12:       private ItemType[] itemTypeArray = new ItemType[ARRAY_SIZE];  //商品種類を保存す
るための配列
13:
14:       /**
15:       * 商品種類を追加する
16:       */
17:       public void add(ItemType addItemType){
18:         //商品種類が入っていない箱を探す
19:         for(int i=0;i<ARRAY_SIZE;i++){
```

```
20:           if(itemTypeArray[i] == null){//入っていない
21:             itemTypeArray[i] = addItemType;//書き込む
22:             break;
23:           }
24:         }
25:       }
26:
27:       /**
28:       * 商品種類リストを表示する
29:       */
30:       public void display(){
31:         for(int i=0;i<ARRAY_SIZE;i++){
32:           if(itemTypeArray[i] != null){//商品種類が入っている
33:
System.out.println(itemTypeArray[i].getId()+":"+itemTypeArray[i].getName()+":"+itemTypeA
rray[i].getPrice()+"は販売中です");
34:           }
35:         }
36:       }
37:
38:     }
```

例題 7-2：継承を使う(ItemTypeLinkedList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-2：継承を使う
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品種類連結リストクラス
7:      */
8:      public class ItemTypeLinkedList extends ItemTypeList{
9:
10:       private LinkObject first;//始点
11:       private LinkObject last;//終点
12:
13:       /**
14:       * 商品種類を追加する
15:       */
16:       public void add(ItemType addItemType){
17:         //追加する連結オブジェクトを生成する
18:         LinkObject addLink = new LinkObject();
19:         addLink.data = addItemType;
20:
21:         if(first == null){//連結リストが空のとき
22:           first = addLink;
23:           last = addLink;
24:         }else{//連結リストが空でないとき
25:           last.next = addLink;
```

```
26:           last = addLink;
27:         }
28:       }
29:
30:       /**
31:       * 商品種類リストを表示する
32:       */
33:       public void display(){
34:         LinkObject current = first;//今たどっている連結オブジェクト
35:         while(current != null){
36:
System.out.println(current.data.getId()+":"+current.data.getName()+":"+current.data.getP
rice()+"は販売中です");
37:           current = current.next;
38:         }
39:       }
40:
41:     }
```

③．抽象クラスとインターフェイス

(1)メソッド名を間違えると

先ほど継承した ItemTypeArrayList.java のコードの中で、もしもメソッドの名前を dispray と書いてしまったとします。

```
/**
* 商品種類を表示するメソッド
*/
public void dispray(){
  for(int i=0;i<10;i++){
    if(itemTypeArray[i] != null){//商品が入っている
      System.out.println(itemTypeArray[i].getName()+"は販売中です");
    }
  }
}
```

display メソッドと書くつもりが
dispray メソッドとしてしまった

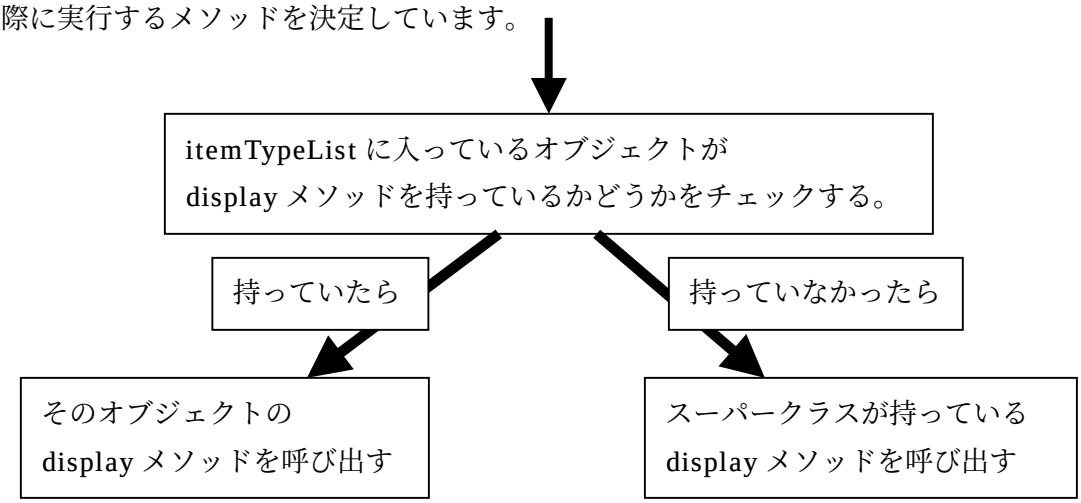
ItemTypeArrayList.java より抜粋

その場合でもコンパイルはちゃんと通りますが、実行すると ItemTypeArrayList クラスの dispray メソッドは呼び出されず、その時は何も起きません。なぜなら dispray メソッドの代りに、ItemTypeList クラスの display メソッドが呼び出されているからです。

(2)メソッドが呼び出される仕組み

```
ItemTypeList itemTypeList = new ItemTypeArrayList();
itemTypeList.display();
```

このプログラムのように display メソッドを呼び出した時、内部的には以下の段階を経て実際に実行するメソッドを決定しています。



だからサブクラスにおいてメソッド名を間違えて宣言すると、気がつかないうちにスーパークラスの display メソッドが呼び出されてしまうのです。

(3)抽象クラス

前ページのようなメソッド名の書き間違いは、デバッグが非常に困難です。これを防ぐためには、「スーパークラスのメソッドを、確実にサブクラスに実装してもらう」という仕組みが必要になります。ItemTypeList クラスの display メソッドを、サブクラスが確実に実装していれば問題がないわけです。

その為の仕組みが「抽象クラス」です。以下に抽象クラスとなった ItemTypeList クラスを示します。

例題 7-3：抽象クラスを使う(ItemTypeList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-3：抽象クラスを使う
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品リストクラス
7:      */
8:      public abstract class ItemTypeList {
9:
10:       /**
11:       * 商品種類を登録する
12:       */
13:       public abstract void add(ItemType addItemType);
14:
15:       /**
16:       * 商品種類リストを表示する
17:       */
18:       public abstract void display();
19:
20:     }
```

クラスとメソッドの宣言に「abstract」と書いてやると、そのクラスは抽象クラスになります。この場合はメソッドの中身を記述しないで、メソッドの宣言だけにします。

抽象クラスを継承したクラスは、抽象クラスのメソッドを実装しないとコンパイルエラーが起きます。だから先ほどのようにサブクラスで dispray メソッドと書き間違えをするとコンパイルエラーが発生します。なぜならサブクラスで display メソッドを実装しなければならないのにそれが見つからないからです。

(4)インターフェイス

Java ではメソッドの定義だけする方法として、インターフェイスがあります。インターフェイスとして定義した ItemTypeList.java を示します。

例題 7-4：インターフェイスを使う(ItemTypeList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-4：インターフェイスを使う
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品リストクラス
7:      */
8:      public interface ItemTypeList {
9:
10:       /**
11:       * 商品種類を登録する
12:       */
13:       public void add(ItemType addItemType);
14:
15:       /**
16:       * 商品種類リストを表示する
17:       */
18:       public void display();
19:
20:     }
```

インターフェイスが持つメソッドは全て定義するだけと決まっているので、抽象クラスと違って「abstract」と書く必要はありません。

このインターフェイスを実装した ItemTypeArrayList クラスを以下に示します。

例題 7-4：インターフェイスを使う(ItemTypeArrayList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-4：インターフェイスを使う
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品種類配列リストクラス
7:      */
8:      public class ItemTypeArrayList implements ItemTypeList{
9:
10:       private int ARRAY_SIZE = 10000;
11:
12:       private ItemType[] itemTypeArray = new ItemType[ARRAY_SIZE];  //商品種類を保存するための配列
```

```
13:
14:       /**
15:       * 商品種類を追加する
16:       */
17:       public void add(ItemType addItemType){
18:         //商品種類が入っていない箱を探す
19:         for(int i=0;i<ARRAY_SIZE;i++){
20:           if(itemTypeArray[i] == null){//入っていない
21:             itemTypeArray[i] = addItemType;//書き込む
22:             break;
23:           }
24:         }
25:       }
26:
27:       /**
28:       * 商品種類リストを表示する
29:       */
30:       public void display(){
31:         for(int i=0;i<ARRAY_SIZE;i++){
32:           if(itemTypeArray[i] != null){//商品種類が入っている
33:
System.out.println(itemTypeArray[i].getId()+":"+itemTypeArray[i].getName()+":"+itemTypeA
rray[i].getPrice()+"は販売中です");
34:           }
35:         }
36:       }
37:
38:     }
```

④.継承 vs インターフェイス

継承とインターフェイスの違いは、以下のようになります。

	抽象クラス	インターフェイス
変数は持てるか	持てる	持てない
実装したメソッドを持てるか	持てる	持てない
多重継承できるか	できない	できる
インスタンス化できるか	できない	できない

このように多少の違いはありますが、基本的にはどちらでも良いと言えます。違いを意識した上で状況に応じて使い分けましょう。

7.2.　if文 vs ポリモーフィズム

7.2.1.　実装クラスの名前を表示する

新しい要求として、「itemTypeManage メソッドの始めにおいて、どの実装クラスなのかを表示するようにして下さい」と求められたとします。どのようにすれば要求を満たすことができるでしょうか。

```
/**
* 商品種類を管理するプログラム
* コーラ、ソーダ、お茶を追加し、商品種類リストを表示する
*/
private static void itemTypeManage(ItemTypeList itemTypeList){

  ここで、「実装クラス名」を表示したい！

  //商品種類を追加する
  itemTypeList.add(new ItemType(1001,"コーラ"));
  itemTypeList.add(new ItemType(1022,"ソーダ"));
  itemTypeList.add(new ItemType(1033,"お茶"));

  //商品種類リストを表示する
  itemTypeList.display();
}
```

7.2.2.　どのように実装するか

①. if 文で実装する

この場合は、メソッドの引数として渡された itemTypeList にどのオブジェクトが入っているかを調べることができれば良さそうです。それを調べるためには「instanceof 演算子」を使います。instanceof 演算子の使い方を示します。

［オブジェクト名（変数名)］instanceof　［クラス名］

これが成立すれば true、不成立ならば false になります。

例えば以下のように書いたとします。

```
ItemTypeList itemTypeList = new ItemTypeArrayList();
if( itemTypeList instanceof ItemTypeArrayList ){
  System.out.println(“yes”);
}else{
  System.out.println(“id”);
}
```

この場合は if 文の中は true となるのでコンソールには「yes」と出力されるでしょう。itemTypeList の中には確かに ItemTypeArrayList オブジェクトが入っているからです。

if 文を使って実装したものが、例題 7-5 です。

例題 7-5：実装クラス名を表示する(Example7_5.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-5：実装クラス名を表示する
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example7_5 {
9:
10:       /**
11:       * メイン
12:       * 1 万件の種類をリストに追加して、その時間を計る
13:       */
14:       public static void main(String[] args) {
15:
```

```
16:         //配列リストの性能を測定する
17:         performanceTest(new ItemTypeArrayList());
18:
19:         //連結リストの性能を測定する
20:         performanceTest(new ItemTypeLinkedList());
21:
22:       }
23:
24:       /**
25:       * 商品種類リストの性能を測る
26:       */
27:       private static void performanceTest(ItemTypeList itemTypeList){
28:
29:         //実装クラスの名前を表示する
30:         if(itemTypeList instanceof ItemTypeArrayList){
31:           System.out.println("実装クラスの名前：ItemTypeArrayList");
32:         }else if(itemTypeList instanceof ItemTypeLinkedList){
33:           System.out.println("実装クラスの名前：ItemTypeLinkedList");
34:         }
35:
36:         long startTime;//開始時間を保存する
37:         long endTime;  //終了時間を保存する
38:
39:         startTime = System.currentTimeMillis();//開始時間を測定する
40:
41:         //商品種類を 1 万件登録する
42:         for(int i=0;i<10000;i++){
43:           itemTypeList.add(new ItemType(1001,"コーラ",120));
44:         }
45:
46:         endTime = System.currentTimeMillis();//終了時間を測定する
47:
48:         System.out.println("かかった時間は"+ (endTime - startTime) + "ミリ秒です");
49:       }
50:
51:     }
```

②.オブジェクトの違いで分岐する

スーパークラスに「実装クラス名を得るためのメソッド」を定義して、サブクラスにそれを実装してもらうというやり方があります。スーパークラスである ItemTypeList クラスを以下に示します。

例題 7-6：ポリモーフィズム(ItemTypeList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 7-6：ポリモーフィズム
4:      * 配列リストと連結リストの追加性能を比較するプログラム
5:      *
6:      * 商品リストクラス
7:      */
8:      public abstract class ItemTypeList {
9:
10:       /**
11:       * 商品種類を登録する
12:       */
13:       public abstract void add(ItemType addItemType);
14:
15:       /**
16:       * 商品種類を表示する
17:       */
18:       public abstract void display();
19:
20:       /**
21:       * このクラスの名前を取得する
22:       */
23:       public abstract String getName();
24:
25:     }
```

例題 7-6：ポリモーフィズム(ItemTypeArrayList#getName()メソッドのみ)

```
/**
* このクラスの名前を取得する
*/
public String getName(){
  return "ItemTypeArrayList";
}
```

例題 7-6：ポリモーフィズム(ItemTypeLinkedList#getName()メソッドのみ)

```
/**
* このクラスの名前を取得する
*/
public String getName(){
  return "ItemTypeLinkedList";
}
```

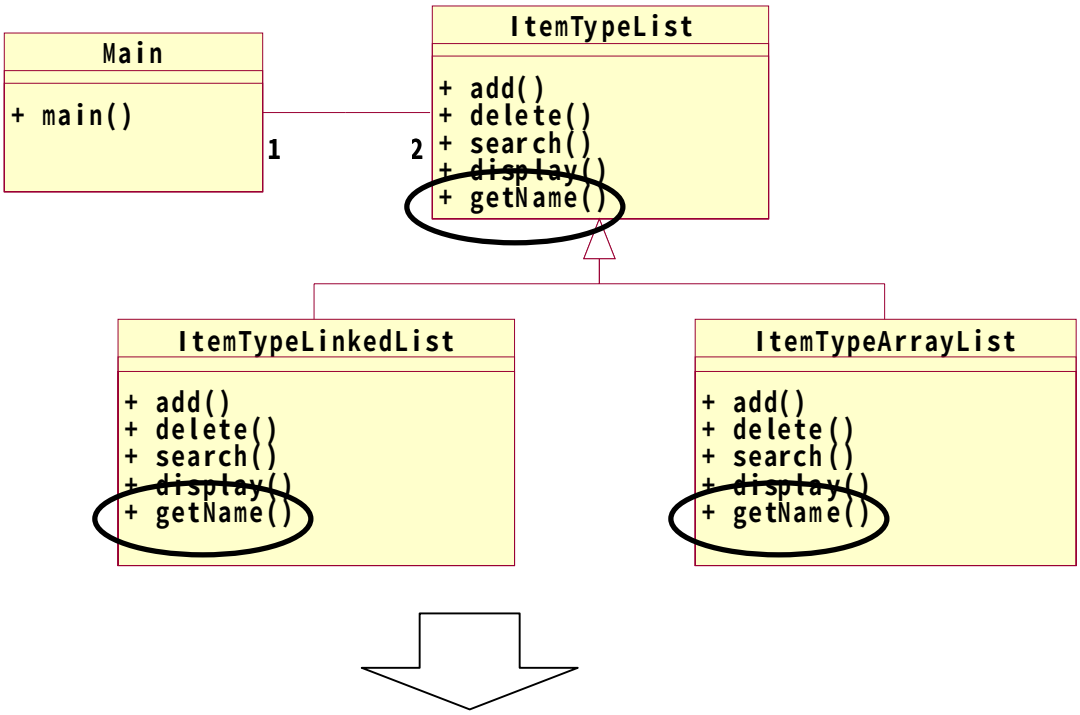
例題 7-6：ポリモーフィズム(Example7_6.java)

```
1:    /**
2:    * オブジェクト指向哲学　入門編
3:    * 例題 7-6：ポリモーフィズム
4:    * 配列リストと連結リストの追加性能を比較するプログラム
5:    *
6:    * メインクラス
7:    */
8:    public class Example7_6 {
9:
10:     /**
11:     * メイン
12:     * 1 万件の種類をリストに追加して、その時間を計る
13:     */
14:     public static void main(String[] args) {
15:       //配列リストの性能を測定する
16:       performanceTest(new ItemTypeArrayList());
17:       //連結リストの性能を測定する
18:       performanceTest(new ItemTypeLinkedList());
19:     }
20:
21:     /**
22:     * 商品種類リストの性能を測る
23:     */
24:     private static void performanceTest(ItemTypeList itemTypeList){
25:
26:       //実装クラスの名前を表示する
27:       System.out.println("実装クラスの名前："+itemTypeList.getName());
28:
29:       long startTime;//開始時間を保存する
30:       long endTime;  //終了時間を保存する
31:       startTime = System.currentTimeMillis();//開始時間を測定する
32:
33:       //商品種類を 1 万件登録する
34:       for(int i=0;i<10000;i++){
35:         itemTypeList.add(new ItemType(1001,"コーラ",120));
36:       }
37:
```

入っているオブジェクトによって
適切なクラスの名前が返ってくる

```
38:         endTime = System.currentTimeMillis();//終了時間を測定する
39:
40:         System.out.println("かかった時間は"+ (endTime - startTime) + "ミリ秒です");
41:       }
42:
43:     }
```

最後に、このような変更を行った際のクラス図を以下に示します。それぞれのクラスに getName メソッドが加わっているのが分かります。



このようなしくみを

「ポリモーフィズム」といいます。

7.2.3.　if 文 vs ポリモーフィズム

要求を満たすための手法として、2 通りのを紹介しました。さて、どちらがよりよい方法でしょうか。

＜議論しよう！＞if 文とポリモーフィズムの利点･欠点

●if 文を用いた場合

●ポリモーフィズムを用いた場合

練習問題

＜記述問題＞

☆記述問題 7-1

継承が適用できない場合はどのような時か述べよ。

☆記述問題 7-2

ポリモーフィズムを適用したらよいと思う場面を考えよ。

＜プログラム問題＞

☆プログラム問題 7-1

プログラム問題 6-1 で作ったプログラムの main()メソッドの重複部分をメソッド化せよ。その際、ItemTypeArrayList、ItemTypeLinkedList クラスを抽象化した ItemTypeList クラスを作れ。ItemTypeList クラスは抽象クラスとして実装せよ。なおプログラム仕様は 6-1 と同様とする。

クラス図は以下のようになる。