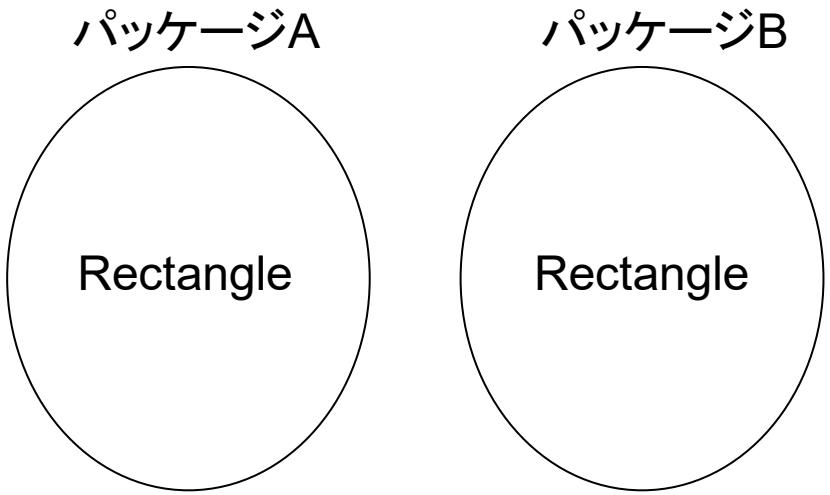
Javaプログラミング（8）
パッケージ，コレクション

成蹊大学理工学部
情報科学科

パッケージ

- パッケージ：名前の衝突を防ぐためのグループ分けの仕組み
- クラスの名前が同じでも，パッケージが異なっていれば異なるクラスとして扱われる



同じRectangleクラスという名前でも，パッケージが異なっていれば名前の衝突エラーは起きない．

パッケージの宣言

パッケージの宣言方法
package パッケージ名

```
package myShapes;
public class Rectangle {
   . . .
}
```

```
package myShapes;
public class Triangle {
. . .
}
```

```
package myShapes;
public class Circle {
     Rectangle rt = new Rectangle();
. . .
}
```

パッケージの宣言(Cont.)

- ソースファイルのはじめ（クラスやインタフェースの宣言よりも前）に書くこと.
- パッケージは”.”でつなぐことにより，ディレクトリ構造のような階層を持たせることができる.

（例）Javaで用意されているパッケージ

java.io, java.util

Javaのパッケージ一覧

java.applet, java.awt, java.awt.color, java.awt.datatransfer, java.awt.dnd, java.awt.event, java.awt.font, java.awt.geom, java.awt.im, java.awt.im.spi, java.awt.image, java.awt.image.renderable, java.awt.print, java.beans, java.beans.beancontext, java.io, java.lang, java.lang.annotation, java.lang.instrument, java.lang.management, java.lang.ref, java.lang.reflect, java.math, java.net, java.nio, java.nio.channels, java.nio.channels.spi, java.nio.charset, java.nio.charset.spi, java.rmi, java.rmi.activation, java.rmi.dgc, java.rmi.registry, java.rmi.server, java.security, java.security.acl, java.security.cert, java.security.interfaces, java.security.spec, java.sql, java.text, java.util, java.util.concurrent, java.util.concurrent.atomic, java.util.concurrent.locks, java.util.jar, java.util.logging, java.util.prefs, java.util.regex, java.util.zip, javax.accessibility, javax.activity, javax.crypto, javax.crypto.interfaces, javax.crypto.spec, javax.imageio, javax.imageio.event, javax.imageio.metadata, javax.imageio.plugins.bmp, javax.imageio.plugins.jpeg, javax.imageio.spi, javax.imageio.stream, javax.management, javax.management.loading, javax.management.modelmbean, javax.management.monitor, javax.management.openmbean, javax.management.relation, javax.management.remote, javax.management.remote.rmi, javax.management.timer, javax.naming, javax.naming.directory, javax.naming.event, javax.naming.ldap, javax.naming.spi, javax.net, javax.net.ssl, javax.print, javax.print.attribute, javax.print.attribute.standard, javax.print.event, javax.rmi, javax.rmi.CORBA, javax.rmi.ssl, javax.security.auth, javax.security.auth.callback, javax.security.auth.kerberos, javax.security.auth.login, javax.security.auth.spi, javax.security.auth.x500, javax.security.cert, javax.security.sasl, javax.sound.midi, javax.sound.midi.spi, javax.sound.sampled, javax.sound.sampled.spi, javax.sql, javax.sql.rowset, javax.sql.rowset.serial, javax.sql.rowset.spi, javax.swing, javax.swing.border, javax.swing.colorchooser, javax.swing.event, javax.swing.filechooser, javax.swing.plaf, javax.swing.plaf.basic, javax.swing.plaf.metal, javax.swing.plaf.multi, javax.swing.plaf.synth, javax.swing.table, javax.swing.text, javax.swing.text.html, javax.swing.text.html.parser, javax.swing.text.rtf, javax.swing.tree, javax.swing.undo, javax.transaction, javax.transaction.xa, javax.xml, javax.xml.datatype, javax.xml.namespace, javax.xml.parsers, javax.xml.transform, javax.xml.transform.dom, javax.xml.transform.sax, javax.xml.transform.stream, javax.xml.validation, javax.xml.xpath, org.ietf.jgss, org.omg.CORBA, org.omg.CORBA_2_3, org.omg.CORBA_2_3.portable, org.omg.CORBA.DynAnyPackage, org.omg.CORBA.ORBPackage, org.omg.CORBA.portable, org.omg.CORBA.TypeCodePackage, org.omg.CosNaming, org.omg.CosNaming.NamingContextExtPackage, org.omg.CosNaming.NamingContextPackage, org.omg.Dynamic, org.omg.DynamicAny, org.omg.DynamicAny.DynAnyFactoryPackage, org.omg.DynamicAny.DynAnyPackage, org.omg.IOP, org.omg.IOP.CodecFactoryPackage, org.omg.IOP.CodecPackage, org.omg.Messaging, org.omg.PortableInterceptor, org.omg.PortableInterceptor.ORBInitInfoPackage, org.omg.PortableServer, org.omg.PortableServer.CurrentPackage, org.omg.PortableServer.POAManagerPackage, org.omg.PortableServer.POAPackage, org.omg.PortableServer.portable, org.omg.PortableServer.ServantLocatorPackage, org.omg.SendingContext, org.omg.stub.java.rmi, org.w3c.dom, org.w3c.dom.bootstrap, org.w3c.dom.events, org.w3c.dom.ls, org.xml.sax, org.xml.sax.ext, org.xml.sax.helpers

パッケージの利用

パッケージの利用方法
①ソース内で個別に指定　　パッケージ名.クラス名;
②ソースファイル先頭で指定　import パッケージ名.*;

①の例　オブジェクトを作成する際にパッケージ名を指定

```
java.util.Random r = new java.util.Random();
  //java.utilがパッケージ名，Randomがクラス名
```

②の例　ファイルの先頭でパッケージ名を指定

```
import java.util.*;
Random r = new Random(); //Randomオブジェクトを作成
 // java.utilの部分を書かなくてよい
```

パッケージの利用(Cont.)

- import文はいくつも並べて書くことができる.
- import文の “*” は，このパッケージ内のクラスとインタフェースに合致という意味.
- パッケージは”.”でつなぐことにより，ディレクトリ構造のような階層を持たせることができる.

  （例）

  import java.awt.*;　// java.awtパッケージをimport

  import java.awt.image.*; //java.awtのサブパッケージである java.awt.imageパッケージをimport
- 通常，java.langパッケージが自動的に取り込まれている．従って，import java.lang.*;とは書かなくてよい.
- パッケージはクラス階層とは無関係．従って，あちこちの階層のクラスをまとめて1つのパッケージにすることもできる.

パッケージとアクセス制御

クラスのメソッドとフィールドに対するアクセス制御		
メソッド, フィールドの修飾子	publicがついているクラスにある場合	publicがついていないクラスにある場合
private	クラス内でのみ使える	
なし	同じパッケージからのみ使える	
protected	同じパッケージとサブクラスからのみ使える	同じパッケージのみから使える
public	他のパッケージから使える	同じパッケージのみから使える

アクセス制御関係の修飾子のまとめ

- アクセス制御関係の修飾子
  - public：他のパッケージからのアクセスを許可する
  - protected：自パッケージおよびサブクラスからのアクセスを許可する
  - 修飾子なし：自パッケージからのみアクセスを許可する
  - private：自クラス内だけからアクセスを許可する
- クラスがpublicでなければ，メソッドやフィールドをpublicにしても他のパッケージからは使えない
- インタフェースのメソッドとフィールドは常にpublicになる

アクセス制御修飾子の使用例

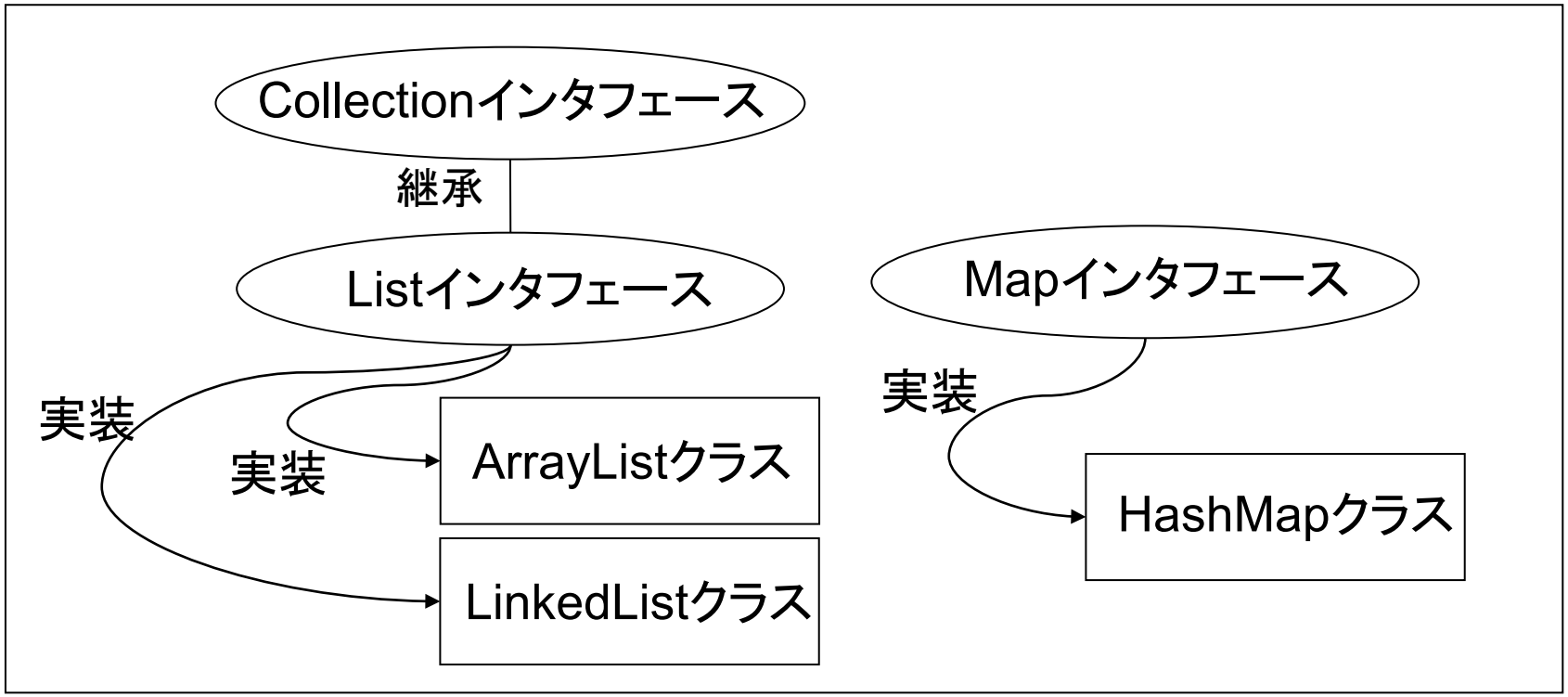
```
package myTools;
public class MyWriter { //myToolsパッケージ外からも使えるクラス
  private int mediaId;
      //MyWriterクラス内からのみアクセス可能なフィールド
  public void writeFile() {
      // myToolsパッケージ外からも使えるメソッド
  }
  protected void loadFile (String filename) {
      // myToolsパッケージ内とサブクラスからのみ使えるメソッド
  }
}
class MyReader{
  //myToolsパッケージ内からのみ使えるクラス
  void readFile() {
    // myToolsパッケージ内からのみ使えるメソッド
  }
}
```

パッケージの保存場所

- パッケージ名をディレクトリ名として使い，対応するディレクトリに保存することが多い．
  （例）パッケージ名：myTools
  - ファイル保存先ディレクトリ：myTools
  - ファイルの保存：myTools¥myWriter.java
- 従って，コマンドラインでコンパイルするときには，
  > javac myTools¥myWriter.java
- 実行するときには，
  > java myTools.myWriter

コレクション

- コレクションフレームワーク：オブジェクトの集合を扱うための仕組み
- Javaではインタフェースを用いて設計されている
- java.utilパッケージにまとめられている.



コレクションフレームワーク成り立ちの例

Javaコレクションフレームワーク

Javaコレクションフレームワークのクラスとインスタンスの関係（一部）

- java.lang.**Object**
  - java.util.**AbstractCollection**<E> (implements java.util.Collection<E>)
    - java.util.**AbstractList**<E> (implements java.util.List<E>)
      - java.util.**AbstractSequentialList**<E>
        - java.util.**LinkedList**<E> (implements java.lang.Cloneable, java.util.Deque<E>, java.util.List<E>, java.io.Serializable)
      - java.util.**ArrayList**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)
      - java.util.**Vector**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)
        - java.util.**Stack**<E>
    - java.util.**AbstractQueue**<E> (implements java.util.Queue<E>)
      - java.util.**PriorityQueue**<E> (implements java.io.Serializable)
    - java.util.**AbstractSet**<E> (implements java.util.Set<E>)
      - java.util.**EnumSet**<E> (implements java.lang.Cloneable, java.io.Serializable)
      - java.util.**HashSet**<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)
        - java.util.**LinkedHashSet**<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)
      - java.util.**TreeSet**<E> (implements java.lang.Cloneable, java.util.NavigableSet<E>, java.io.Serializable)
    - java.util.**ArrayDeque**<E> (implements java.lang.Cloneable, java.util.Deque<E>, java.io.Serializable)
  - java.util.**AbstractMap**<K,V> (implements java.util.Map<K,V>)
    - java.util.**EnumMap**<K,V> (implements java.lang.Cloneable, java.io.Serializable)
    - java.util.**HashMap**<K,V> (implements java.lang.Cloneable, java.util.Map<K,V>, java.io.Serializable)
      - java.util.**LinkedHashMap**<K,V> (implements java.util.Map<K,V>)
    - java.util.**IdentityHashMap**<K,V> (implements java.lang.Cloneable, java.util.Map<K,V>, java.io.Serializable)
    - java.util.**TreeMap**<K,V> (implements java.lang.Cloneable, java.util.NavigableMap<K,V>, java.io.Serializable)
    - java.util.**WeakHashMap**<K,V> (implements java.util.Map<K,V>)

コレクションフレームワークで実装されているインタフェース

- List<要素の型>: 要素が並んでいるリストを管理するインタフェース
- Queue<要素の型>: キューを管理するインタフェース. スレッドで使われる
- Set<要素の型> : 要素の集合を管理するインタフェース
- Map<キーの型，値の型>: キーと値の対応関係を管理するインタフェース.
- これら以外のインタフェースも実装されている

ArrayList

- ArrayList：要素を追加すると，自動的に配列（のようなもの）の長さを追加できるクラス
- java.util.Listインタフェースが実装されている

```
//コレクションを使うには，java.util.*をインポートする
import java.util.*;
public class ArrayListSample {
        public static void main(String[] args) {
                //ArrayListのインスタンスを作成
                ArrayList<String> alist = new ArrayList <String> ();
```

ArrayListの宣言，初期化方法

- ArrayList<String>と書くと，「String型を要素とするArrayList型」を指定することになる．
- <型名>の部分を「型パラメータ」と呼ぶ
- 型パラメータを使って，型の宣言ができる機能のことをジェネリクス(generics)という．
- javaのジェネリクスはJ2SE5.0から導入された．それ以前は，Vectorクラスを用いて，要素を取り出す際，毎回キャストを行っていた．
- 基本型コレクションの作り方
  - intやcharなどの基本型は型パラメータとして使えないという決まりがある．
  - int型のArrayListを作るには、int型のラッパークラスIntegerを使う
  - × ArrayList <int>
  - ○ ArrayList<Integer>

Autoboxing

- 基本データ型はオブジェクトではない
- ラッパークラス：基本データ型を包み込んでオブジェクトとして使えるようにするクラス
  - これにより，メソッドの適用などが可能に
  - コレクションの型パラメータもラッパークラスでなければならない
- Autoboxing:プリミティブ→ラッパークラスの自動変換

  int i = new Integer(10);

プリミティブ型	ラッパークラス
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

ArrayList使用例

```
import java.util.*;
public class ArrayListSample {
        public static void main(String[] args) {
                //ArrayListのインスタンスを作成
                ArrayList<String> alist = new ArrayList <String> ();
                //要素の追加　add
                alist.add("山田");
                alist.add("田中");
                alist.add("長谷川");
                //要素の参照 get
                System.out.println("2番目の人："+alist.get(1));
                //要素の設定 set
                alist.set(1,"鈴木");
                System.out.println("2番目の人："+alist.get(1));
                . . . つづく
```

ArrayList使用例(Cont.)

. . . つづき

```
                //要素の確認 contains
                if (alist.contains("山田")) {
                        alist.remove("山田"); //要素削除 remove
                }
                //要素数を求める size
                System.out.println("現在の人数:"+alist.size());
                for (int i=0;i<alist.size();i++) {
                        System.out.println(i+":"+alist.get(i));
                }
                //指定された要素のリスト中の位置（インデックス）を求める
                System.out.println("長谷川さんは何番
目?:"+alist.indexOf("長谷川"));
        }
}
```

ArrayList使用例（実行結果）

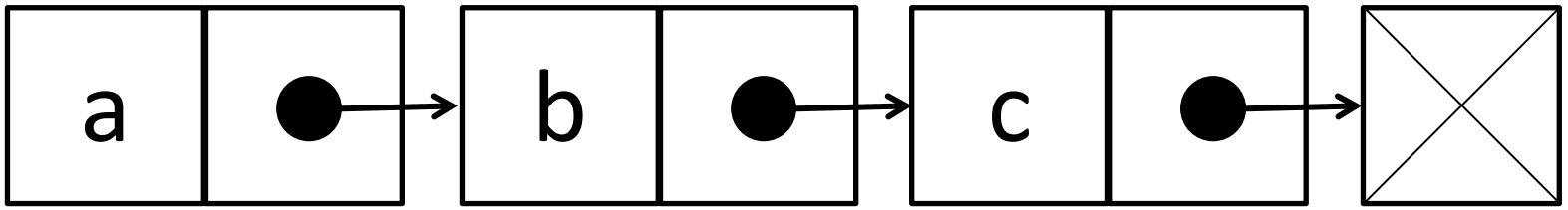
```
2番目の人：田中
2番目の人：鈴木
現在の人数:2
0:鈴木
1:長谷川
長谷川さんは何番目?:1
```

ArrayListの主な機能

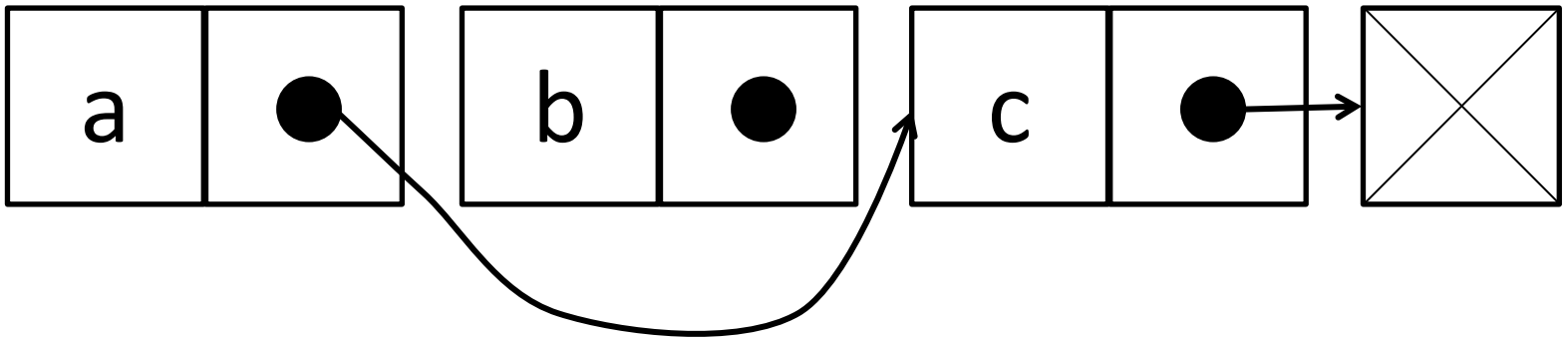
ArrayList の主な機能	
確保	list = new ArrayList<要素の型>();
代入（置き換え）	list.set(添え字, 要素);
追加	list.add(要素);
指定された位置に指定された要素を挿入	list.add(添え字, 要素)
削除	list.remove(オブジェクト);
参照	list.get(添え字);
要素数	list.size();
内容確認	list.contains(オブジェクト);
インデックス検出	list.indexOf(オブジェクト);

LinkedList

- LinkedList
  - 各データが次のデータへのリファレンス（リンク）を保持している一連のデータからなるデータ構造



  - 削除，追加が容易



LinkedList

- LinkedListは要素の並び順について管理している
  - 挿入や削除を頻繁に行う場合は ArrayList よりも LinkedList の方が高速.
- java.utilのListやQueueのインタフェースが実装されている
  - キューの性質を持っている
- ArrayList と同じように使用することができる.
  - add, set, get, remove等のメソッドがすべて同じように使用可

LinekdListによるキューの実装例

```
import java.util.*;
public class LinkedListSample {
        public static void main(String[] args) {
                /*Queueインタフェース型の変数に
                        LinedListクラスのインスタンスを代入*/
                Queue <String> queue = new LinkedList<String>();
                //指定された要素をこのリストの末尾 (最後の要素) に追加
                queue.offer("山田");
                System.out.println(queue);
                queue.offer("田中");
                System.out.println(queue);
                queue.offer("鈴木");
                System.out.println(queue);
                queue.offer("中村");
                System.out.println(queue);
```

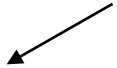
LinekdListによるキューの実装例 (Cont.)

```
. . . つづき
        try{
                while (true) { //無限ループ
                        //先頭要素の参照 element(peekも可)
                        String name = queue.element();
                        System.out.println("element:"+name);
                        //このリストの先頭 (最初の要素) を取得し、削除(pollも可)
                        name = queue.remove();
                        System.out.println("remove:"+name);
                        System.out.println("queue:"+queue);
                }
        } catch (NoSuchElementException e) {
                System.out.println(e);
        }
    }
}
```

実行結果

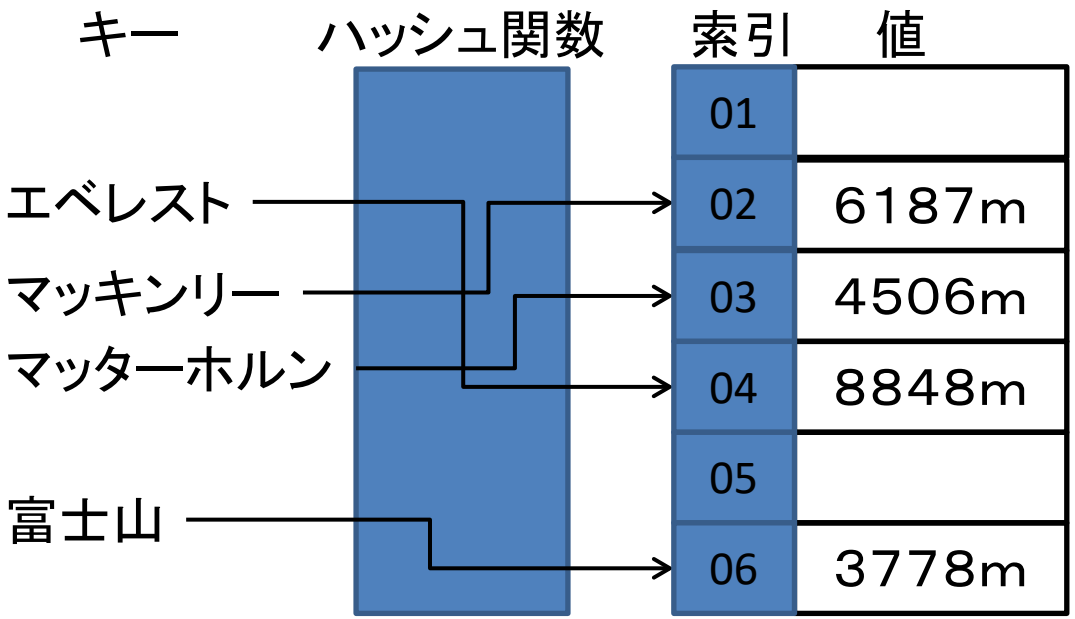
```
[山田]
[山田, 田中]
[山田, 田中, 鈴木]
[山田, 田中, 鈴木, 中村]
element:山田
remove:山田
queue:[田中, 鈴木, 中村]
element:田中
remove:田中
queue:[鈴木, 中村]
element:鈴木
remove:鈴木
queue:[中村]
element:中村
remove:中村
queue:[]
java.util.NoSuchElementException
```

リスト中に要素がなくなり，例外が発生



HashMap

- キーとそれに関連付けられた値との対応関係をインデックスとして登録・保持し，高速に値にアクセスすることができるデータ構造.
- java.util.Mapインタフェースが実装されている



HashMap (具体例)

```
import java.util.*;
public class HashMapSample {
        public static void main(String[] args) {
                //Mapインタフェース型の変数にHashMapを代入
                Map <String, Integer> map = new HashMap<String,
Integer>();
                //キーと値のペアを追加　put
                map.put("山田",80);
                map.put("田中",76);
                map.put("鈴木",89);
                map.put("中村",86);
                //値を参照　get
                System.out.println("山田:"+map.get("山田"));
                System.out.println("鈴木:"+map.get("鈴木"));
        }
}
```

出力結果

```
山田:80
鈴木:89
```

拡張for文(for-each loop)

- J2SE 5.0以上では拡張for文を用いると，リストの各要素へのアクセスを簡単に書くことができる.
- for (要素の型　変数名 : コレクションクラスの変数)
  for (Integer num: list)
  コレクションクラスで作成されたlistオブジェクト内の各要素を変数numに順次代入する．全要素が処理されるまでこれを繰り返す.

拡張for文の例

```
import java.util.ArrayList;

public class ForEachTest {
	public static void main(String[] args) {
		//ArrayListのインスタンスを作成
		ArrayList<String> alist = new ArrayList <String> ();
		//要素の追加　add
		alist.add("山田");
		alist.add("田中");
		alist.add("長谷川");
		// listオブジェクトの各要素をString型の変数stへ代入して
		//プリントアウトし，これを全要素について終了するまで繰り返す
		int i = 0;
		for (String st: alist) {
			i++;
			System.out.println(i+":"+st);
		}
	}
}
```