第11章　集合类

第11章 集合类

- Iterable接口
- Collection接口
- List接口
- Set接口
- Map接口
- 泛型
- 集合类实例：书籍管理

第11章 集合类

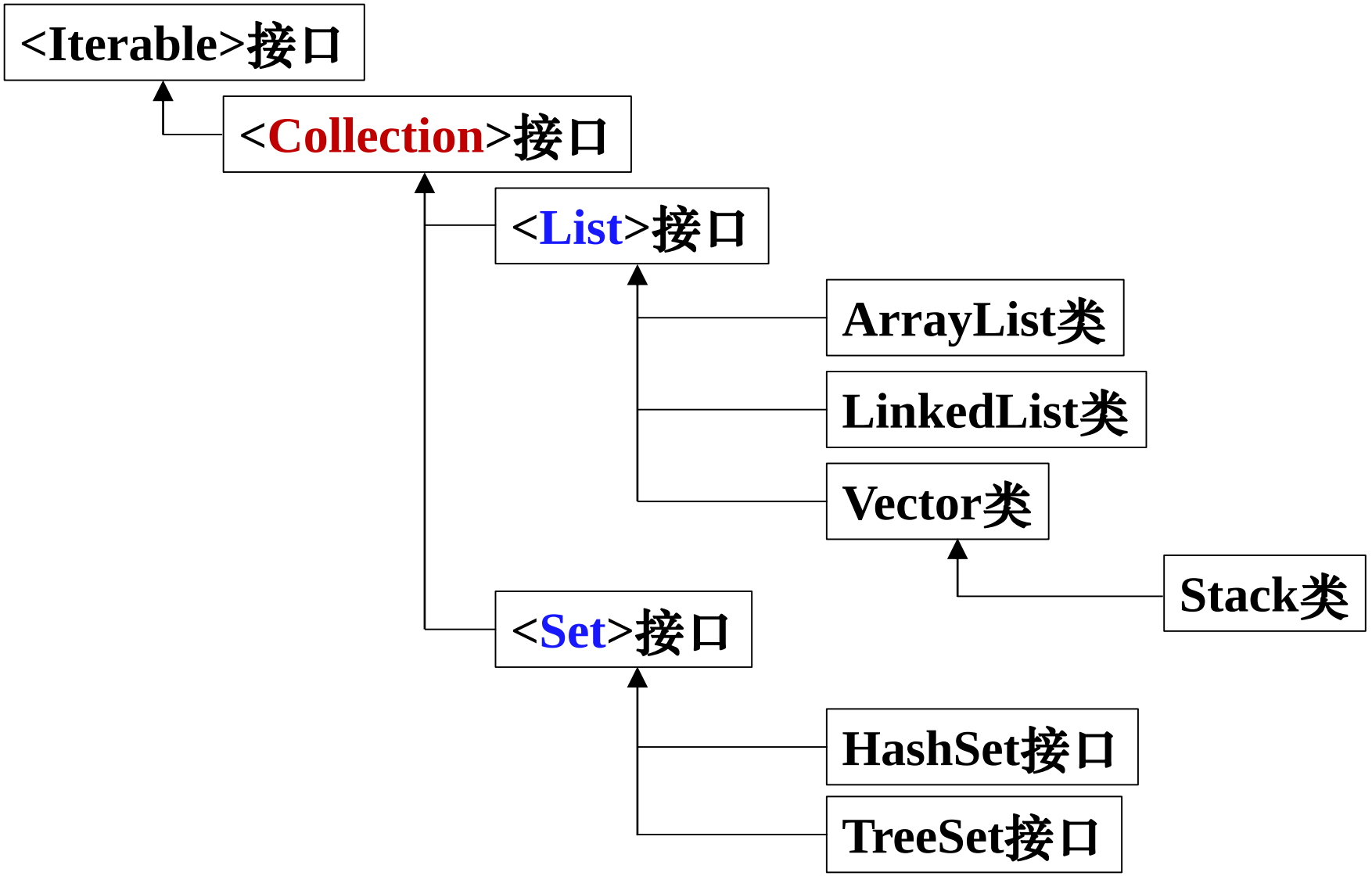
- 数组是相同类型数据的集合。数组创建后，数组的大小固定，无法动态改变。

- Java语言在java.util包中提供了一套集合类型，集合类可以容纳多个变量。

- 与数组不同，集合类型自动扩展。

- 集合类中只能容纳对象，不能容纳基本数据类型数据。

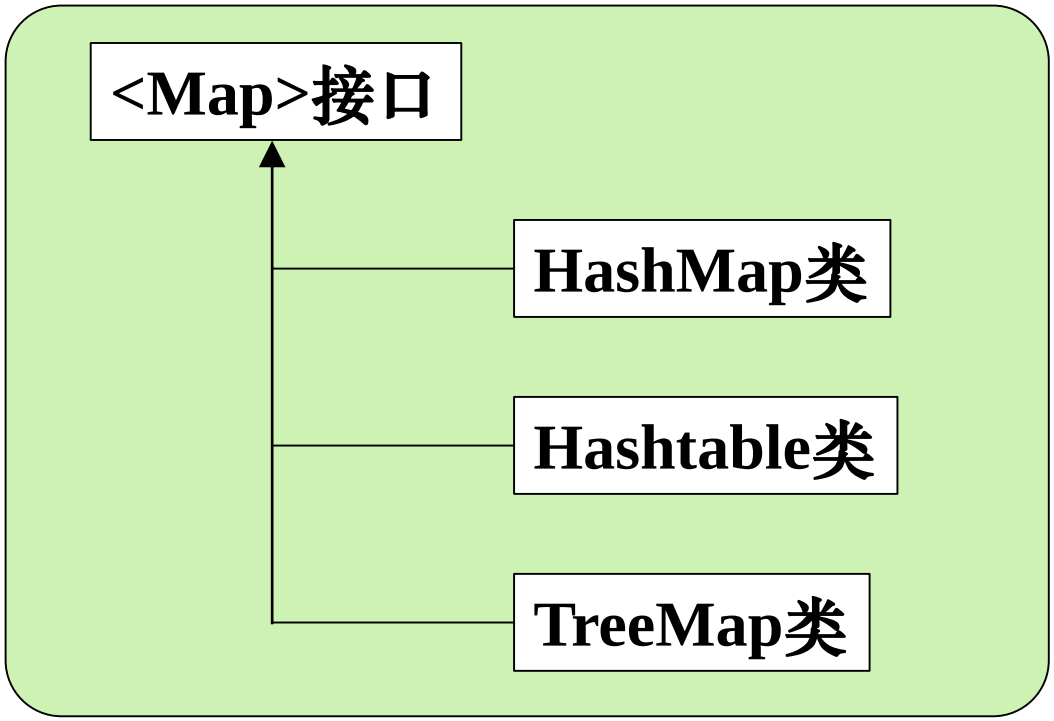
第11章 集合类


<Iterable>接口
<Collection>接口
<List>接口
ArrayList类
LinkedList类
Vector类
Stack类
<Set>接口
HashSet接口
TreeSet接口

第11章 集合类

◆ Map接口在集合框架之外，它是将键映射到值的对象。

◆一个映射不能包含重复的键，每个键只能映射一个值。



11.1 Iterable接口

- Iterable接口在java.lang包中。
- 集合总是要迭代的，需要查找集合中的每一个元素。

该接口的唯一方法：iterator()。实现这个接口允许对象成为 “foreach”语句的目标。

Iterator<T> iterator()

返回一个在一组 T 类型的元素上进行迭代的迭代器。

11.2 Collection接口

- Collection接口的方法分成基本、批量、数组和迭代几种。

- Collection接口定义如下:

```
public interface Collection<E> extends Iterable<E>
```

（1）基本方法

- int size()
  返回集合元素的个数。
- boolean isEmpty()
  如果collection不包含元素，则返回true。
- boolean contains(Object o)
  如果collection包含指定的元素，则返回true。
- boolean add(E e)
  向集合中增加一个元素，成功则返回true，否则返回false
- boolean remove(Object o)
  从collection中移除指定元素的单个实例。

（2）批量操作方法

- boolean addAll(Collection c)
  将指定collection中的所有元素都添加到当前collection，成功则返回true。
- boolean removeAll(Collection c)
  删除所有元素，成功则返回true。
- boolean containsAll(Collection c)
  ruguo collection包含指定collection中的所有元素，则返回true。
- boolean retainAll(Collection c)
  保留collection中那些也包含在指定collection中的元素。
- void clear()
  移除collection中的所有元素。

（3）数组操作方法

- Object[ ] toArray()
  返回包含collection 中所有元素的数组。

- <T> T [ ] toArray(T[] a)
  返回包含collection中所有元素的数组。返回数组的类型与指定数组的运行时类型相同。

（4）迭代操作方法

- 迭代操作是为集合提供顺序获取元素的方法。Iterator iterator() 返回一个实现迭代接口的对象。

- 迭代接口定义的方法有：boolean hasNext()。只要集合存在下一个元素，可用Object next()方法获取下一个元素。

【例11.1】集合类实例

```
// IteratorExample.java
import java.util.*;
public class IteratorExample{
public static void main(String args[]){
    Collection intList =new ArrayList();
    //创建一个列表
    int[] values={9,11,-7,1,14,89,3,0};
    for(int i=0;i<values.length; i++)
          intList.add(new Integer(values[i]));
```

```
        System.out.println("迭代之前：  "+ intList);
        //显示迭代之前的列表
        Iterator myIterator=intList.iterator();  //定义迭代
        while(myIterator.hasNext()){        //用循环实现迭代
            Integer element=(Integer)myIterator.next();
             //获取下一个元素
            int value=element.intValue();
            //如果元素值不在1和10之间，删除该元素、
             if(value<1 || value >10)
               myIterator.remove();
        }
        System.out.println("迭代之后：  "+ intList);
        //显示迭代之后的列表
        }
}
```

运行结果：

迭代之前：[9,11,-7,1,14,89,3,0]

迭代之后：[9,1,3]

11.3 List接口

List接口是元素有序并可重复的集合。可以利用List的下标位置找到元素，下标从0开始。

List接口中定义的方法

- E get(int index)

  返回列表中指定位置的元素。
- E set(int index, E element)

  用指定元素替换列表中指定位置的元素。
- boolean add(E e)

  向列表的尾部添加指定元素。
- E remove(int index)

  移除列表中指定位置的元素。

11.3 List接口

List接口中定义的方法

- boolean addAll(Collection<extends E> c)

  添加指定collection中所有元素到此列表的结尾，顺序是指定collection迭代器返回元素的顺序。

- int indexOf(Object o)

  返回列表中第一次出现指定元素的索引。如果此列表不包含该元素，则返回-1。

- ListIterator<E> listIterator()

  返回此列表元素的列表迭代器。

- ListIterator<E> listIterator(int index)

  返回列表中元素的列表迭代器，从列表的指定位置开始。

11.3.1 LinkedList类

LinkedList提供额外的get，remove，insert方法。这些操作使LinkedList可用作堆栈（stack），队列（queue）或双向队列（deque）。

11.3.2 ArrayList类

调用ensureCapacity()方法来增加ArrayList的容量。

【例11.2】ArrayList实例

```
import java.util.*;
public class ListExample{
        public static void main(String args[]){
                List mylist= new ArrayList();
                mylist.add("Welcome");
                mylist.add("to");
                mylist.add("beijing");
                mylist.add(new Integer(2012));
                mylist.add("Welcome");
                String[] str={"J", "a", "v","a"};
                mylist.add(str);
                mylist.add(new Integer(2012));
                System.out.println(mylist);
         }
        }
```

运行结果:

[Welcome, to, beijing, 2012, Welcome, [Ljava.lang.String;@de6ced, 2012]

11.3.3 Vector类

- Vector类与ArrayList类相似，都是动态数组，区别在于Vector类是同步的。

- 同步是指多个线程同时访问某个对象时，保证只有唯一线程访问对象。

- Vector类的使用ArrayList类相似，可参考ArrayList类。

11.3.3 Stack 类

Vector类与ArrayList类相似，都是动态数组，区别在于Vector类是同步的。

- boolean empty()
  测试堆栈是否为空。
- E peek()
  查看堆栈顶部的对象，但不从堆栈中移除它。
- E pop()
  移除堆栈顶部的对象，并返回该对象。
- E push(E item)
  把对象E压入堆栈顶部。
- int search(Object o)
  返回对象在堆栈中的位置，以1为基数。

11.3.3 Stack 类

```
import java.util.*;
public class StackExample{
    static String[]
        months={"January","February","March","April","
        May","June","July","August","September","Octob
        er","November","December"};
    public static void main(String args[]){
        Stack myStack=new Stack();
        for(int i=0;i<months.length; i++)
        // 将12个月单词从小到大压入堆栈
            myStack.push(months[i]+" ");
        System.out.println("Stack= "+myStack);
            myStack.addElement("lastOne ");
```

11.3.3 Stack 类

```
        // 放入最后一个单词
        //取下标是5的元素
        System.out.println("Element 5 is: "
                                        +myStack.elementAt(5));
        System.out.println("pop Element: ");
        while(!myStack.empty())
        //将堆栈中的元素依次提取
        System.out.print(myStack.pop());
        }
}
```

11.4 Set接口

- Set接口是一种不包含重复元素的Collection，即任意的两个元素e1和e2都满足e1.equals(e2)=false。

- Set中最多有一个null元素，并且元素的顺序不重要。

以下情况考虑它们是否适合用set集合表示：

（1）等待看医生的病人队列

（2）一系列数字，每个数字代表一年中52个星期中的一个星期

（3）停车许可证记录的汽车注册编号的集合

11.4 Set接口

（1）等待看医生的病人队列

等待看医生的病人队列不能被看作一个set集合，因为这个问题中顺序非常重要。

（2）一系列数字，每个数字代表一年中52个星期中的一个星期

数字允许有重复，因此这组元素也不适合用set集合表示。

（3）停车许可证记录的汽车注册编号的集合

汽车注册编码的组合可以看作set集合，因为没有重复元素，而且顺序并不重要。

11.4.1 Set接口常用方法

Set接口是一种不包含

- a.containsAll(b)
  对应的数学操作为： b ∈ a （子集）
- a.addAll(b)
  对应的数学操作为： a= a∪b （合集）
- removeAll(b)
  对应的数学操作为： a=a-b （差集）
- retainAll(b)
  对应的数学操作为： a=a∩b （交集）
- a.clear()
  对应的数学操作为： a=Φ （空集）

实现Set接口的常用类有HashSet类和TreeSet类。

11.4.2 Set接口实例

【例11.3】汽车注册管理

```
import java.util.*;
public class HashSetExample{
public static void main(String args[]){
        Set regNums=new HashSet();       // 创建Set集合
        regNums.add("V5230");            //向集合中添加元素
        regNums.add("X8901");
        regNums.add("L3319");
        regNums.add("W7034");
        //输出集合元素个数
        System.out.println("Number of items in set: "+ regNums.size());
        System.out.println(regNums);    //输出结合内的元素
        boolean ok;
        ok=regNums.add("W7034");
```

11.4.2 Set接口实例

【例11.3】汽车注册管理

```
//添加元素时判定集合中是否已经存在
        if(!ok){
                System.out.println("item is already in set.");
                regNums.remove("W7034");   //如果存在，则删除该元素
        }
        System.out.println("**********************");
        System.out.println("Number of items in set: "+ regNums.size());
        System.out.println(regNums);
        }
}
```

运行结果:

```
Number of items in set: 4
[X8901, V5230, L3319, W7034]
item is already in set.
**********************
Number of items in set: 3
 [X8901, V5230, L3319]
```

11.5 Map接口

11.5.1 Map常用方法

- Map接口不是继承自Collection接口，Map提供key到value的映射。
- Map不能包含相同的key，每个key只能映射一个value。
- 在Map中顺序并不重要，但关键字是唯一的。
- 通常将Map看成一个查找表，key(关键字对象)用于查找。
  - 例如：网络中用户的密码可以通过用户名来查询。

11.5.1 Map常用方法

- put (K key, V value)
  将指定的值与映射中的指定键关联。
- get (Object key)
  返回指定键所映射的值。如果映射不包含该键的映射关系，则返回null。
- remove(Object key)
  如果存在一个键的映射关系，则将其从此映射中移除。
- boolean containsKey(Object key)
  如果映射包含指定键的映射关系，则返回true。
- boolean containsValue(Object value)
  如果映射将一个或多个键映射到指定值，则返回true。

11.5.1 Map常用方法

- int size()

  返回映射中键-值映射关系数。
- boolean isEmpty()

  如果映射未包含键-值映射关系，则返回true。
- void putAll(Map m)

  将所有映射关系复制到映射中。
- void clear()

  从映射中移除所有映射关系。

实现Map接口的常用类有Hashtable类、HashMap类、WeakHashMap类等。

11.5.2 HashMap管理网络名和密码

```
import java.util.*;
public class HashMapExample{
        public static void main(String args[]){
                Map<String,String> users=new
                                        HashMap<String,String>();
                users.put("Marry","monkey");
                //向Map中添加映射对。
                users.put("Jenny","network");
                users.put("Sussan","network");
                System.out.println("Number of items in Map: "+
                                        users.size());
                System.out.println(users);
```

11.5.2 HashMap管理网络名和密码

```
                //检测用户名是否已经被占用，
                                    如果使用则删除旧的，再添加。
                if(users.containsKey("Tommy"))
                        users.remove("Tommy");
                //删除给出键值的映射。
                users.put("Tommy","banner");
                System.out.println("**********************");
                System.out.println("Number of items in Map: "+
                                                    users.size());
                System.out.println(users);
        }
}
```

11.5.2 HashMap管理网络名和密码

- 聚集类的类型定义为Map接口，但使用<String, String> 明确了Key和value的数据类型，这种方式称为泛型机制。

- 程序中向Map添加数据采用put方法，添加时提供成对的两个参数，分别代表key和value。

```
users.put(“Marry”, “monkey”);
```

11.5.2 HashMap管理网络名和密码

- 判断Map中是否已有某个关键字时使用containsKey()方法。
- 该方法接受一个对象，如果该对象是Map中的一个关键字，则返回true。

  users.containsKey(“Tommy”)。

- 输出时，元素显示的顺序不取决于它们被加入的顺序，而是取决于在内部的存储顺序。
- Map接口也提供了size和isEmpty方法，与Set和List中的使用方式相同。

聚集类框架使用小结

- Collection是集合接口，有Set和List子接口。Set子接口无序，不允许重复。List子接口有序，可以有重复元素。Set和List对比：
- Set接口：检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。HashSet子类，以哈希表的形式存放元素，插入删除速度很快。
- List接口：和数组类似，List可以动态增长，查找元素效率高，插入删除元素效率低，因为会引起其他元素位置改变。List接口的子类ArrayList是动态数组
- LinkedList可以表示链表、队列、堆栈。
- Map接口是键映射到值得关系。一个映射不能包含重复的键，每个键最多只能映射一个值。某些映射可以保证元素顺序，如TreeMap类；某些映射实现不保证元素顺序，如HashMap类。

11.6 泛型

【例11.6】未使用泛型实例

```
import java.util.Hashtable;
public class HashtableExample{
        public static void main(String args[]){
                Hashtable h = new Hashtable();
                h.put(new Integer(0),"value");
                String s=(String)h.get(Integer(0));
                System.out.println(s);
        }
}
```

11.6 泛型

【例11.7】泛型实例

```
import java.util.Hashtable;
public class HashtableExample{
        public static void main(String args[]){
                Hashtable<Integer,String> h=new
                        Hashtable<Integer,String>();
                h.put(0,"value");
                String s=h.get(0);
                System.out.println(s);
        }
}
```

泛型定义

- 泛型是创建以类作为参数的类。
- 泛型类应用程序的参数用尖括号(<>)括起来。
- 泛型的本质是参数化类型，即所操作的数据类型被定义为参数。
- 这种参数可以用在类、接口和方法的创建中。

泛型定义

使用泛型要注意如下几点：

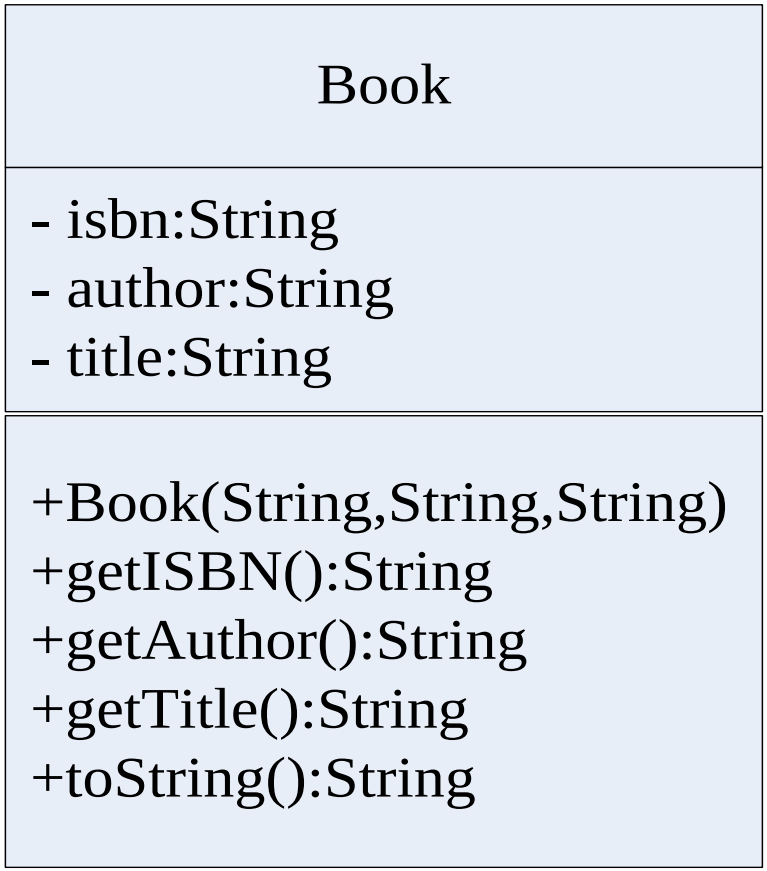
- 定义泛型类的时候在“< >”之间是形式类型参数，
  如：“Hashtable <Integer, String> myTable”

- 实例化泛型对象时，也要在类名后面指定参数的类型。
  例如：

```
Hashtable <Integer, String> myTable =new Hashtable
<Integer, String> ();
```

- 泛型中<key extends Object>表示参数必须是Object类型，不可以是简单类型,类型参数可以有多个。

11.7 集合类实例：书籍管理



Book类的UML类图。

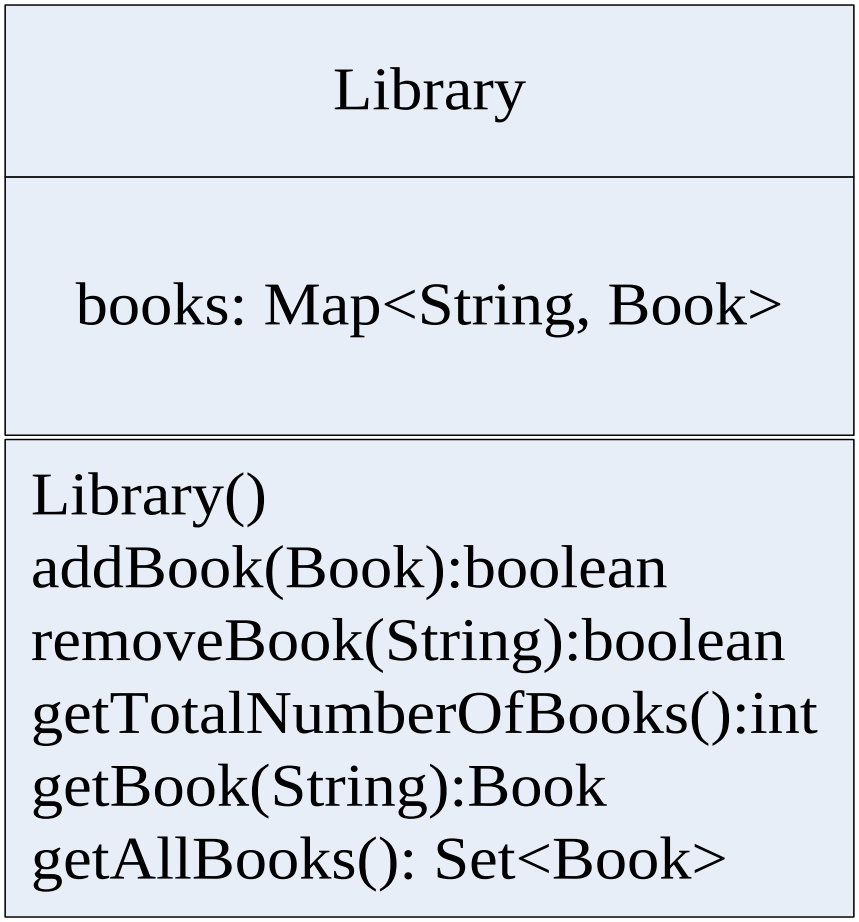
11.7 集合类实例：书籍管理

```
//Book.java
public class Book{
        private String isbn;
        private String title;
        private String author;
        public Book(String isbnIn,String titleIn,String authorIn){
                isbn=isbnIn;
                title=titleIn;
                author=authorIn;
        }
        public String getISBN(){
                return isbn;
        }
```

11.7 集合类实例：书籍管理

```
    public String getTitle(){
        return title;
    }
    public String getAuthor(){
        return author;
    }
    public String toString(){
        return "(" + isbn +"," + author+ ", " +title +")\n";
    }
}
```

11.7 集合类实例：书籍管理



Library类的UML设计

11.7 集合类实例：书籍管理

```
// Library.java
import java.util.*;
public class Library{
        Map<String,Book> books;
        public Library(){
                books=new HashMap<String,Book>();
        }
        public boolean addBook(Book bookIn){
                String keyIn=bookIn.getISBN();
                if(books.containsKey(keyIn))
                        return false;
                else{
                        books.put(keyIn,bookIn);
                        return true;
         }      }
```

11.7 集合类实例：书籍管理

```
public boolean removeBook(String isbnIn){
        if(books.remove(isbnIn)!=null)
                return true;
        else
                return false;
}
public int getTotalNumberOfBooks(){
        return books.size();
}
public Book getBook(String isbnIn){
        return books.get(isbnIn);
}
```

11.7 集合类实例：书籍管理

```
        public Set<Book> getAllBooks(){
                Set<Book> bookSet=new HashSet<Book>();
                Set<String> thekeys=books.keySet();
                for(String isbn: thekeys){
                        Book theBook=books.get(isbn);
                        bookSet.add(theBook);
                }
                return bookSet;
        }
}
```

11.7 集合类实例：书籍管理

```
import java.util.*;
public class TestBook{
    public static void main(String args[]){
        Library myLibrary=new Library();
        Book book1=new Book("978-1-283","Java","JD");
        Book book2=new Book("925-6-257","Database","MQ");
        Book book3=new Book("421-8-925","NetWork","SU");
        if (myLibrary.addBook(book1))
            System.out.println("添加成功");
        else
            System.out.println("添加失败");
        if (myLibrary.addBook(book2))
            System.out.println("添加成功");
        else
            System.out.println("添加失败");
```

11.7 集合类实例：书籍管理

```
        if (myLibrary.addBook(book3))
            System.out.println("添加成功");
        else
            System.out.println("添加失败");

        System.out.print(myLibrary.getBook("978-1-283"));
        System.out.println("Total Number is:"+
                           myLibrary.getTotalNumberOfBooks());
        Set<Book> myAllBooks;
        myAllBooks=myLibrary.getAllBooks();
        System.out.print(myAllBooks);
        }
}
```