1、List中可以添加任何对象，包括自己定义的新的类。

2、List是一个接口，不能实例化，需要实例化一个ArrayList或者LinkedList  
List myList = new ArrayList();  
3、使用myList.add(任何对象);就可以进行添加了。  
4、取值的时候myList.get(索引);取出来的值都是Object，使用时需要类型转换。  
5、可用Iterator迭代器对List中的元素进行迭代操作。

* class Person{  
  .....  
  }  
  上面定义了一个Person类，下面看好如何使用List  
  Person p1=new Person();  
  Person p2=new Person();  
  List list=new ArrayList();  
  list.add(p1);  
  list.add(p2);//这里是将对象加入到list中  
  for(int i=0;i  
  Person p=(Person)list.get(i);//注意，这里一定要强制类型转换，因为List中取出的对象都是Object类型的，希望对你有所帮助  
    
  }

List 集合中的对象按照一定的顺序排放，里面的内容可以重复。  
List接口实现的类：ArrayList(实现动态数组)， Vector（实现动态数组） ，LinkedList（实现链表）， Stack（实现堆栈）  
  
  
  
  
  
  
  
一 list接口  
1.java.util.ArrayList类实现一个可增长的动态数组，平时用的比较多的就是add()方法，其它方法  
请参考API帮助文档。使用该方法向集合内添加对象，并且这些对象是按照一定的顺序排列  
的，它内部原理是数组实现的，因此处理数据量较大时，不建议使用。  
  
public class TestArrayList {  
  
public static void main(String[] args) {  
//声明List 并实例化为ArrayList  
List al = new ArrayList();  
//使用add（）方法添加元素  
al.add("a");  
al.add("b");  
al.add("c");  
al.add("d");  
//使用Iterator迭代器遍历出集合的元素并打印  
for(Iterator i = al.iterator();i.hasNext(); ){  
String str = (String) i.next();  
System.out.println(str);  
}  
}  
  
}  
3.java.util.Vector（向量）类 与ArrayList类似的也是实现一个动态的数组，该向量可以随意的增长以存放更多的对象  
  
  
public class TestArrayList {  
  
public static void main(String[] args) {  
//声明List 并实例化为Vector  
List al = new Vector();  
//使用add（）方法添加对象  
al.add("a");  
al.add("b");  
al.add("c");  
al.add("d");  
//使用Iterator迭代器遍历出集合的对象并打印  
for(Iterator i = al.iterator();i.hasNext(); ){  
String str = (String) i.next();  
System.out.println(str);  
}  
}  
  
}  
3.java.util.LinkedList类实现了链表，可初始化化为空或者已存在的集合，通常使用以下方法  
add();向链表末尾添加对象。  
addFirst（）在链表开头添加对象。  
addLast（）在链表末尾添加对象。  
getFirst（）得到链表开头的对象。  
getLast（）得到链表末尾的对象。  
注意该类提供了随机访问列表中的元素的方法，但是底层依然必须遍历去查找随机访问的对象，因此性能依然有限.  
  
public static void main(String[] args) {  
//声明LinkedList并实例化  
LinkedList al = new LinkedList();  
//使用add（）方法添加元素  
al.add("a");  
al.add("b");  
al.add("c");  
al.add("d");  
//使用Iterator迭代器遍历出集合的元素并打印  
for(Iterator i = al.iterator();i.hasNext(); ){  
String str = (String) i.next();  
System.out.println(str);  
}  
System.out.println("\_\_\_\_\_");  
//向链表头和尾分别添加x和z  
al.addFirst("z");  
al.addLast("x");  
//遍历查看添加后的结果  
for(Iterator i = al.iterator();i.hasNext(); ){  
String str = (String) i.next();  
System.out.println(str);  
}  
  
}  
  
4.java.util.Stack类实现了堆栈数据结构，即按照先进后出的原则存放数据。创建时只能为空。  
使用push（）方法进行添加对象  
public static void main(String[] args) {  
//声明Stack并实例化  
Stack al = new Stack();  
//使用push（）方法添加元素  
al.push("a");  
al.push("b");  
al.push("c");  
al.push("d");  
al.push("f");  
//使用Iterator迭代器遍历出集合的元素并打印  
for(Iterator i = al.iterator();i.hasNext(); ){  
String str = (String) i.next();  
System.out.println(str);  
}  
}

-----------

Example added by myself：

public class Test {

 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  ArrayList list = new ArrayList();  
  list.add(0,"aa");  
  list.add(1,"bb");  
  list.add(2,"cc");  
  for(int i=0;i<10;i++)  
  {  
     
   list.add(Integer.toString(i));  
  }  
  for(int i=0;i<list.size();i++)  
  {  
   System.out.println(list.get(i));  
  }  
  for(Iterator i=list.iterator();i.hasNext();)  
   //首先定义一个list对象的Iterator，然后利用该Iterator进行循环迭代操作  
  {  
   String s =(String)i.next();  
   System.out.println(s);  
  }  
 }

}

数组是大小固定的，并且同一个数组只能存放类型一样的数据（基本类型/引用类型），JAVA集合可以存储和操作数目不固定的一组数据。 所有的JAVA集合都位于 java.util包中！ JAVA集合只能存放引用类型的的数据，不能存放基本数据类型。

JAVA集合主要分为三种类型：

* Set(集)
* List(列表)
* Map(映射)

Collection 接口 ：Collection是最基本的集合接口，声明了适用于JAVA集合（只包括Set和List）的通用方法。 Set 和List 都继承了Conllection,Map

**Collection接口的方法：**

boolean add(Object o)      ：向集合中加入一个对象的引用

void clear()：删除集合中所有的对象，即不再持有这些对象的引用

boolean isEmpty()    ：判断集合是否为空

boolean contains(Object o) ： 判断集合中是否持有特定对象的引用

Iterartor iterator()  ：返回一个Iterator对象，可以用来遍历集合中的元素

boolean remove(Object o) ：从集合中删除一个对象的引用

int size()       ：返回集合中元素的数目

Object[] toArray()    ： 返回一个数组，该数组中包括集合中的所有元素

关于：Iterator() 和toArray() 方法都用于集合的所有的元素，前者返回一个Iterator对象，后者返回一个包含集合中所有元素的数组。

**Iterator接口声明了如下方法：**

hasNext()：判断集合中元素是否遍历完毕，如果没有，就返回true

next() ：返回下一个元素

remove()：从集合中删除上一个有next()方法返回的元素。

Set(集合)： Set是最简单的一种集合。集合中的对象不按特定的方式排序，并且没有重复对象。 Set接口主要实现了两个实现类：

* HashSet： HashSet类按照哈希算法来存取集合中的对象，存取速度比较快
* TreeSet ：TreeSet类实现了SortedSet接口，能够对集合中的对象进行排序。

Set 的用法：存放的是对象的引用，没有重复对象

1. Set set=**new** HashSet();
3. String s1=**new** String("hello");
5. String s2=s1;
7. String s3=**new** String("world");
9. set.add(s1);
11. set.add(s2);
13. set.add(s3);
15. System.out.println(set.size());//打印集合中对象的数目 为 2。

Set 的 add()方法是如何判断对象是否已经存放在集合中？

1. **boolean** isExists=**false**;
3. Iterator iterator=set.iterator();
5. **while**(it.hasNext())           {
7. String oldStr=it.next();
9. **if**(newStr.equals(oldStr)){
11. isExists=**true**;
13. }
15. }

List(列表)： List的特征是其元素以线性方式存储，集合中可以存放重复对象。

List接口主要实现类包括：

* ArrayList() : 代表长度可以改变得数组。可以对元素进行随机的访问，向ArrayList()中插入与删除元素的速度慢。
* LinkedList(): 在实现中采用链表数据结构。插入和删除速度快，访问速度慢。

对于List的随机访问来说，就是只随机来检索位于特定位置的元素。 List 的 get(int index) 方法放回集合中由参数index指定的索引位置的对象，下标从“0” 开始。最基本的两种检索集合中的所有对象的方法：

1： for循环和get()方法：

1. **for**(**int** i=0; i<list.size();i++){
3. System.out.println(list.get(i));
5. }

2： 使用 迭代器（Iterator）:

1. Iterator it=list.iterator();
3. **while**(it.hashNext){
5. System.out.println(it.next);
7. }

**Map(映射):**

Map 是一种把键对象和值对象映射的集合，它的每一个元素都包含一对键对象和值对象。 Map没有继承于Collection接口 从Map集合中检索元素时，只要给出键对象，就会返回对应的值对象。

Map 的常用方法：

1 添加，删除操作：

Object put(Object key, Object value)： 向集合中加入元素

Object remove(Object key)： 删除与KEY相关的元素

void putAll(Map t)：  将来自特定映像的所有元素添加给该映像

void clear()：从映像中删除所有映射

2 查询操作：

Object get(Object key)：获得与关键字key相关的值 。Map集合中的键对象不允许重复，也就说，任意两个键对象通过equals()方法比较的结果都是false.，但是可以将任意多个键独享映射到同一个值对象上。

Conllections： 集合实用类 。Conllections提供了供JAVA集合实用的静态方法

**总结：**

JAVA集合的基本用法，都归纳了，上面这些是平常最常用的JAVA集合，具体的其他的，还要参考JDK帮助文档了，呵呵 关于 Map的应用，还有很多，具体就是这个，Conllections提供了很多 List /Map 实用的方法，对平常开发非常有用。

boolean containsKey(Object key)： 判断映像中是否存在关键字key

boolean containsValue(Object value)：判断映像中是否存在值value

int size()：返回当前映像中映射的数量

boolean isEmpty() ：判断映像中是否有任何映射

List按对象进入的顺序保存对象，不做排序或编辑操作。Set对每个对象只接受一次，并使用自己内部的排序方法(通常，你只关心某个元素是否属于 Set,而不关心它的顺序--否则应该使用List)。Map同样对每个元素保存一份，但这是基于"键"的，Map也有内置的排序，因而不关心元素添加的 顺序。如果添加元素的顺序对你很重要，应该使用 LinkedHashSet或者LinkedHashMap.

**List的功能方法**  
  
实际上有两种Lis：一种是基本的ArrayList,其优点在于随机访问元素，另一种是更强大的LinkedList,它并不是为快速随机访问设计的，而是具有一套更通用的方法。

List：次序是List最重要的特点：它保证维护元素特定的顺序。List为Collection添加了许多方法，使得能够向List中间插入与移除元素(这只推 荐LinkedList使用。)一个List可以生成ListIterator,使用它可以从两个方向遍历List,也可以从List中间插入和移除元 素。

ArrayList：由数组实现的List。允许对元素进行快速随机访问，但是向List中间插入与移除元素的速度很慢。ListIterator只应该用来由后向前遍历 ArrayList,而不是用来插入和移除元素。因为那比LinkedList开销要大很多。

LinkedList ：对顺序访问进行了优化，向List中间插入与删除的开销并不大。随机访问则相对较慢。(使用ArrayList代替。)还具有下列方 法：addFirst(), addLast(), getFirst(), getLast(), removeFirst() 和 removeLast(), 这些方法 (没有在任何接口或基类中定义过)使得LinkedList可以当作堆栈、队列和双向队列使用。

**Set的功能方法**

Set具有与Collection完全一样的接口，因此没有任何额外的功能，不像前面有两个不同的List。实际上Set就是Collection,只 是行为不同。(这是继承与多态思想的典型应用：表现不同的行为。)Set不保存重复的元素(至于如何判断元素相同则较为负责)

Set : 存入Set的每个元素都必须是唯一的，因为Set不保存重复元素。加入Set的元素必须定义equals()方法以确保对象的唯一性。Set与Collection有完全一样的接口。Set接口不保证维护元素的次序。

* HashSet：为快速查找设计的Set。存入HashSet的对象必须定义hashCode()。
* TreeSet： 保存次序的Set, 底层为树结构。使用它可以从Set中提取有序的序列。
* LinkedHashSet：具有HashSet的查询速度，且内部使用链表维护元素的顺序(插入的次序)。于是在使用迭代器遍历Set时，结果会按元素插入的次序显示。

**Map的功能方法**

方法put(Object key, Object value)添加一个“值”(想要得东西)和与“值”相关联的“键”(key)(使用它来查找)。方法get(Object key)返回与给定“键”相关联的“值”。可以用containsKey()和containsValue()测试Map中是否包含某个“键”或“值”。 标准的Java类库中包含了几种不同的Map：HashMap, TreeMap, LinkedHashMap, WeakHashMap, IdentityHashMap。它们都有同样的基本接口Map，但是行为、效率、排序策略、保存对象的生命周期和判定“键”等价的策略等各不相同。

执行效率是Map的一个大问题。看看get()要做哪些事，就会明白为什么在ArrayList中搜索“键”是相当慢的。而这正是HashMap提高速 度的地方。HashMap使用了特殊的值，称为“散列码”(hash code)，来取代对键的缓慢搜索。“散列码”是“相对唯一”用以代表对象的int值，它是通过将该对象的某些信息进行转换而生成的。所有Java对象都 能产生散列码，因为hashCode()是定义在基类Object中的方法。

HashMap就是使用对象的hashCode()进行快速查询的。此方法能够显着提高性能。

Map : 维护“键值对”的关联性，使你可以通过“键”查找“值”

HashMap：Map基于散列表的实现。插入和查询“键值对”的开销是固定的。可以通过构造器设置容量capacity和负载因子load factor，以调整容器的性能。

LinkedHashMap： 类似于HashMap，但是迭代遍历它时，取得“键值对”的顺序是其插入次序，或者是最近最少使用(LRU)的次序。只比HashMap慢一点。而在迭代访问时发而更快，因为它使用链表维护内部次序。

TreeMap ： 基于红黑树数据结构的实现。查看“键”或“键值对”时，它们会被排序(次序由Comparabel或Comparator决定)。TreeMap的特点在 于，你得到的结果是经过排序的。TreeMap是唯一的带有subMap()方法的Map，它可以返回一个子树。

WeakHashMao ：弱键(weak key)Map，Map中使用的对象也被允许释放: 这是为解决特殊问题设计的。如果没有map之外的引用指向某个“键”，则此“键”可以被垃圾收集器回收。

IdentifyHashMap： : 使用==代替equals()对“键”作比较的hash map。专为解决特殊问题而设计。

迭代器（Iterator）

　　迭代器是一种设计模式，它是一个对象，它可以遍历并选择序列中的对象，而开发人员不需要了解该序列的底层结构。迭代器通常被称为“轻量级”对象，因为创建它的代价小。

　　Java中的Iterator功能比较简单，并且只能单向移动：

　　(1) 使用方法iterator()要求容器返回一个Iterator。第一次调用Iterator的next()方法时，它返回序列的第一个元素。注意：iterator()方法是java.lang.Iterable接口,被Collection继承。

　　(2) 使用next()获得序列中的下一个元素。

　　(3) 使用hasNext()检查序列中是否还有元素。

　　(4) 使用remove()将迭代器新返回的元素删除。