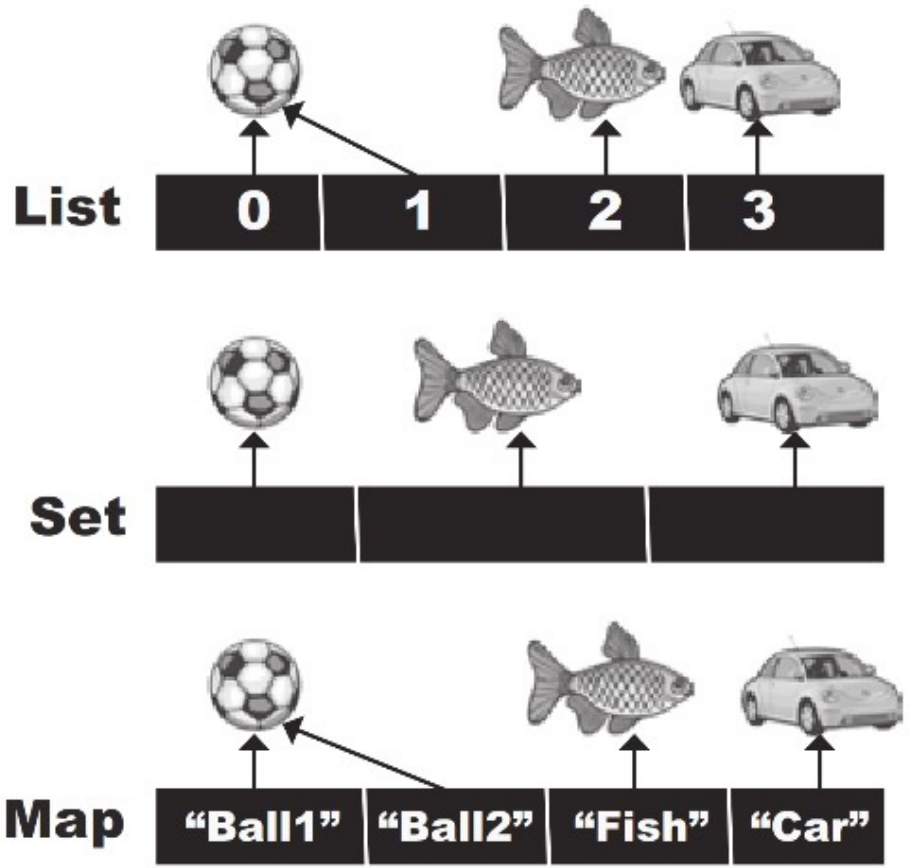
نگهداری از objectهایتان

معرفی

- جاوا روش های مختلفی برای نگهداری objectها(یا در واقع referenceای به objectها)دارد. نوع پشتیبانی شده توسط کامپایلر آرایه است که پیش تر در رابطه با آن صحبت شد.
- اما آرایه ها طول ثابت دارند و در حالت کلی تر،در زمان نوشتن برنامه نمی دانیم که به چه تعداد object احتیاج خواهیم داشت.
- کتابخانه java.util مجومه نسبتا کاملی از کلاس های container دارد که این مشکل را حل می کنند. نوع های پایه موجود عبارتند از:Set،List و Map

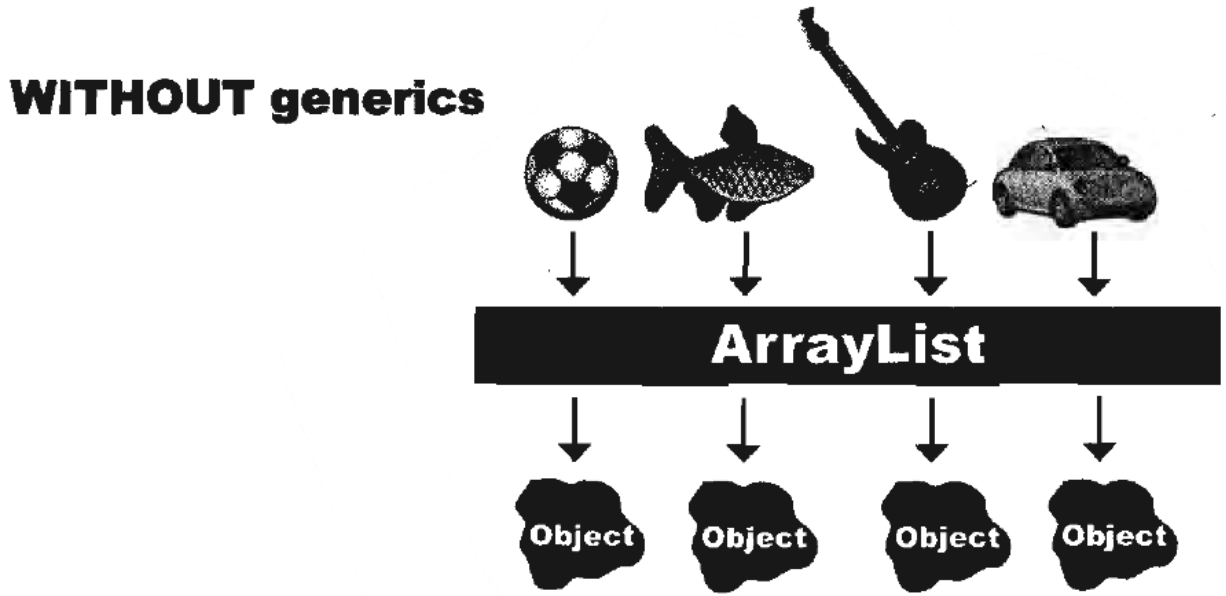
معرفی

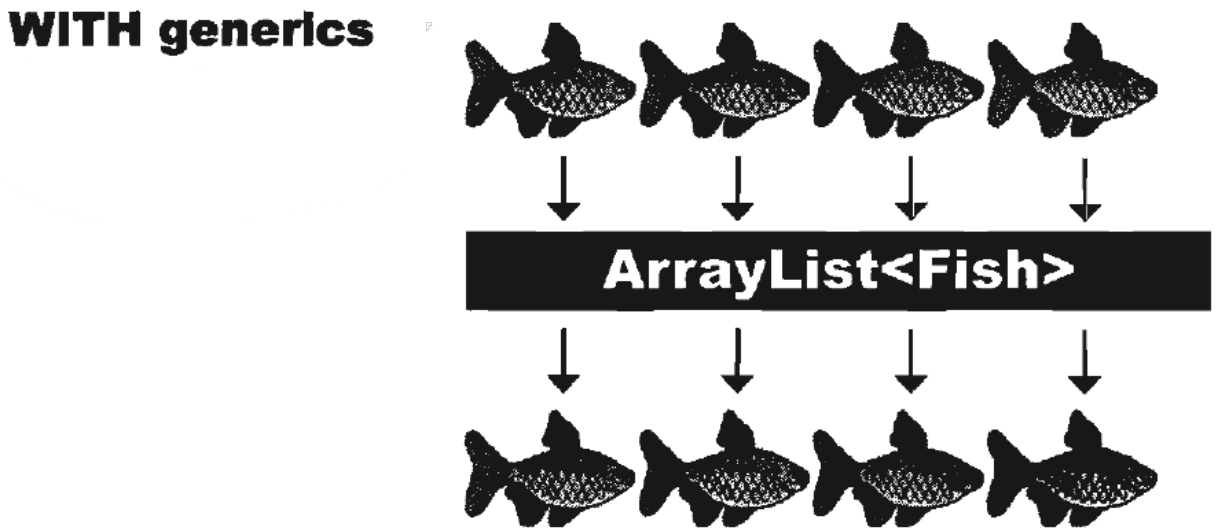


وgenerics

هایcontainer

typesafe





typesafeهای container و generics

- یکی از مشکلات استفاده از containerها در ورژن های قبل از java SE5 این بود که کامپایلر امکان قرار دادن نوع های اشتباه در یک ظرف را فراهم می کرد.
- در این مثال Apples و Orange هر دو در یک container قرار گرفته اند:

```
public static void main(String[] args) {
     ArrayList apples = new ArrayList();
     for(int i = 0; i < 3; i++)
       apples.add(new Apple());
    apples.add(new Orange());
    for(int i = 0; i < apples.size(); i++)
       ((Apple)apples.get(i)).id();
}
```

generics و typesafe های container

- استفاده از کلاس های عمومی از پیش تعریف شده معمولا ساده است. به عنوان مثال، برای تعریف ArrayList در نظر گرفته شده برای نگهداری اشیای Apple، به جای استفاده از عبارت ArrayList می توان از ArrayList <Apple> استفاده کرد.
- براکت های زاویه ای (<>) نوع مد نظر را دربرمی گیرند و از این طریق نوعی را که می توان توسط نمونه ای از container نگهداری کرد مشخص می کنند.
- استفاده از generics ،در زمان کامپایل شما را از قراردادن نوع اشتباهی از object در داخل یک container منع می کند.

مفاهیم پایه

- کتابخانه collection ها جاوا به دو مفهوم مجزا تقسیم می شود:

1) Collection: دنباله ای از عناصر جدا با یک یا چند قانون اعمال شده به آنها.list، set
2) Map:یک map به شما این امکان را می دهد که از طریق یک object به عنوان کلید به دنبال object دیگری بگردید.به map آرایه پیوندی نیز می گویند،چرا که objectهایی را با object ها دیگر و یا مجموعه ای پیوند می دهد

- بنابراین می توانید listای مانند list روبرو ایجاد کنید:

```
List<Apple> apples = new ArrayList<Apple>();
List<Apple> apples = new LinkedList<Apple>();
```

```
public class PrintingContainers {
  static Collection fill(Collection<String> collection) {
     collection.add("rat");
     collection.add("cat");
     collection.add("dog");
     collection.add("dog");
     return collection;
  }
  static Map fill(Map<String,String> map) {
     map.put("rat", "Fuzzy");
     map.put("cat", "Rags");
     map.put("dog", "Bosco");
     map.put("dog", "Spot");
     return map;
  }
  public static void main(String[] args) {
     print(fill(new ArrayList<String>()));
     print(fill(new LinkedList<String>()));
     print(fill(new HashSet<String>()));
     print(fill(new TreeSet<String>()));
     print(fill(new LinkedHashSet<String>()));
     print(fill(new HashMap<String,String>()));
     print(fill(new TreeMap<String,String>()));
     print(fill(new LinkedHashMap<String,String>()));
  }
}
```

چاپ و نمایش containerها

چاپ و نمایش containerها

- شیوه نمایش collection به این صورت است که عناصر در داخل براکت های]و[قرار می گیرند و از طریق کاما از هم جدا می شوند.یک map در داخل براکت های }و{ قرار می گیرد و هر key و value متناظر با آن از طریق علامت تساوی نمایش داده می شوند.(key در سمت چپ و مقدار متناظر با آن در سمت راست = قرار می گیرد)
- ArrayList وLinkedList هر دو از نوع list هستند و از روی خروجی آن ها مشخص است که هردو به یک شیوه اشیاء را نگهداری میکنند
- TreeSet، HashSet وLinkedHashSet از نوع set. هستند
- این مثال از سه نوع پایه map استفاده کرده است: TreeMap، HashMap و LinkedHashMap

List

- لیست عناصر را به ترتیبی خاص نگهداری می کند.interface لیست تعدادی متد را به مجموعه اضافه کرده که از طریق آن می توان در بین list عناصری را اضافه یا حذف کرد.
- دو نوع list وجود دارد:

✓ ArrayList که در دسترسی تصادفی به عناصر بهتر عمل می کند اما هنگام اضافه یا حذف کردن المان ها در بینlist. آهسته عمل می کند

✓ LinkedList که دسترسی ترتیبی بهینه ای را با اضافه و حذف عناصر در بین list به شیوه ای کم هزینه فراهم می کند. یک لیست پیوندیLinkedList در دسترسی تصادفی به نسبت آهسته تر عمل می کند اما دارای مجموعه قابلیت های بیشتری از ArrayList است.

متدهای مفید مورد استفاده برای List

```
list.add(h); //add new Object
list.remove(h); //remove Object from List
list.add(3, new Mouse());//add in specific index
list.subList(1,4);//extract a list from-to index
Collections.sort(list);//sort the entire content
Collections.shuffle(list);//random the order
list.retainAll(subList);//keep just given sub list
list.removeAll(subList);//remove the given sub List
```

List

```
list.addAll(anotherList);//add all elements of sub list
list.addAll(2, sub);//add all elements of sub list
                    //from index 2
list.set(1, new Mouse());//replace object in index 1
list.clear();//empty the list
list.isEmpty();//check is list empty?
Object[] o= list.toArray();//convert List to object array
Pet[] pa = list.toArray(new Pet[0]);//convert
//list to specific array
```

Iterator

- می توان به آسانی در collectionها پیمایش کرد اما مشکلی نیز وجود دارد:
- فرض کنید می خواهید از ابتدا کدی را بنا به هدفی بنویسید و نمی دانید و یا برایتان اهمیتی ندارد که چه نوع containerای با آن کار می کند.
- مفهوم iterator(یکی دیگر از الگوهای طراحی) می تواند برای رسیدن به این انتزاع استفاده گردد.

Iterator

- java.util.Iterator برای حرکت برروی یک Collectionاستفاده می‌شود و می تواند در یک جهت حرکت کند. عملیاتی که می توان از طریقiterator انجام داد عبارت است از :
  - از collection بخواهید که به شما iteratorای بدهد.این کار از طریق متد iterator() انجام می گیرد و این iterator برای بازگشت به اولین عنصر توالی آماده خواهد بود.
  - دسترسی به object بعد در توالی: next()
  - مشاهده این که object دیگری در توالی وجود دارد یا نه: hasNext()
  - حذف آخرین عنصری که از طریق iterator برگشت داده شده است: remove()

Iterator

```
List<Pet> pets = Pets.arrayList(12);
Iterator<Pet> it = pets.iterator();
    while(it.hasNext()) {
       Pet p = it.next();
        System.out.print(p.id() + ":" + p + " ");
    }
```

ListIterator

- ListIteratorزیرمجموعه قوی تری از iteratorها است که تنها برای کلاس های listایجاد شده است .ListIterator دو طرفه است و می تواند indexهایی برای عناصر قبل و بعد،نسبت به نقطه ای که iteartorبه آن اشاره می کند،ایجاد نماید . همچنین از طریق متد() setمی توان آخرین عنصر مشاهده شده را جایگزین کرد.متدهای دیگر مورد استفاده عبارتند از :

hasPrevious()
previous()

LinkedList

LinkedList متدهایی دارد که به آن اجازه می دهد تا به عنوان یک پشته (stack)، صف (Queue) و یا یک صف دو طرفه (doubleended queue) یا deque (استفاده شود.

Offer :مانند add ()و addLast ()عمل می کند و عنصری را به انتهای list اضافه می کند.

Poll :حذف و برگرداندن اولین عنصر لیست.در صورتی که لیست خالی باشد مقدار null را برمیگرداند.

peek: اولین عنصر لیست را برمی گرداند. در صورتی که لیست خالی باشد مقدار null را برمیگرداند.

addFirst :عنصری را به ابتدای لیست اضافه می کند.

getFirstو element : اولین عنصر لیست را بدون حذف آن برمی گردانند و در صورتی که لیست خالی باشد استثنای NoSuchElementException را نشان می دهد.

removeFirstو remove : حذف و برگرداندن اولین عنصر لیست و در صورتی که لیست خالی باشد استثنای NoSuchElementException را نشان می دهد.

Stack(پشته)

- LinkedList متدهایی دارد که از طریق آن می توان مستقیما ساختار stackرا پیاده سازی کرد. همچنین می توان ازjava.util.Stackاستفاده نمود
- یک stack گاهی به عنوان ظرف LIFO(last in,first out) شناخته می شود و گاهی پشته pushdown نامیده می شود چرا که هر چیزی را که در آخر در پشته قرار می دهید،اولین عنصری خواهد بود که از پشته خارج می شود.

✓ Push این متد برای اضافه کردن یک آبجکت از نوعT.به پشته استفاده می شود

✓Peek. این متد بالاترین عنصر پشته را بر می گرداند اما آن را حذف نمی کند :

✓Pop. این متد بالاترین عنصر پشته را حذف و آن را بر می گرداند

Set

- یک set از تنها یک instance از مقدار هر object را نگهداری می کند و در صورتی که بخواهید بیش از یک نمونه از object متناظر را نگه دارید،set از این تکرار جلوگیری می کند.
- معمولا می توان از پیاده سازی HashSet استفاده کرد که برای جستجوی سریع،بهینه شده است.
- LinkedHashSet نیز از hashبرای سریع جست و جو کردن استفاده می کند و به نظر می رسد که عناصر را به ترتیب درج،در یک لیست پیوندی نگه می دارد .
- TreeSet.عناصر را به صورت ساختار داده ای مرتب،طبقه بندی می کند

Set

- یکی از عملیات رایج،تست بررسی عضویت از طریق متد contains() می باشد.
- در صورتی که از TreeSet<String> استفاده می شود و میخواهید به ترتیب الفبا مرتب شود، از عبارت String.CASE_INSENSITIVE_ORDER Comparator مطابق زیر به constructor درخت اضافه کنید.(یک مقایسه گر،object ای است که ترتیب را مشخص می کند.)

```
Set<String> words = new
TreeSet<String>(String.CASE_INSENSITIVE_ORDER);
```

Map

- توانایی نگاشت دادن objectهایی به objectهای دیگر می تواند یک راه فوق العاده قدرتمند برای حل مشکلات برنامه نویسی باشد.

```
Map<Integer,Integer> m = new HashMap<Integer,Integer>();
```

//نمی توان از نوع داده های اولیه با containerها استفاده کرد.

- get () این متد در صورتی که :key در داخلcontainer نباشد مقدارnull. را برمی گرداند
- Put () این متد مقادیر :key وvalue متناظر را برای نگهداری درmap. می پذیرد
- یک map می تواند یک set از keyهای موجود،یک collection از valueهای موجود و یا یک set از جفت های key و value را برگرداند.متد keyset() یک set از همه keyهای موجود را تولید می کند.

Queue(صف)

- یک صف در واقع یک ظرف FIFO(first-in, first-out)می باشد.این به این معنی است که می توانید مقادیر را از یک طرف وارد کنید و از طرف دیگر توالی خارج کنید و مقادیر به همان ترتیبی که وارد می شوند،از صف خارج می گردند.
- صف ها معمولا به عنوان روشی جهت انتقال مطمئن objectها از یک نقطه از برنامه به نقطه ای دیگر،استفاده می شود.
- LinkedList متدهایی دارد که از روش صف پشتیبانی کرده و interface صف را پیاده سازی می کند .

حلقه foreach و iteratorها

- حلقه Foreach با هر شی collection کار می کند.تا به حال مثال های کوچکی از این حالت را در استفاده از ArrayList دیده اید.اما در این جا یک مصداق کلی ارائه شده است:

```
for(String currentValue : list)
   System.out.print("‘" + currentValue + "‘ ");
```

- در صورتی که از کلاسی استفاده می کنید که به صورت iterable پیاده سازی می شود،می توانید آن را در یک حلقه foreach استفاده کنید.

container رده بندی ساده