

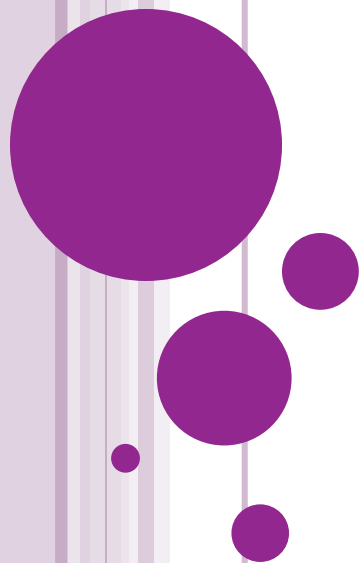
انجمن جاواکاپ تقدیم می‌کند

دوره برنامه‌نویسی جاوا

ظرف‌ها و ساختمان‌های داده

Containers and Data Structures

صادق علی اکبری



- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- بازنشر یا تدریس آن چه توسط جاواکاپ و به صورت عمومی منتشر شده است، با ذکر مرجع (جاواکاپ) بلامانع است
- اگر این اثر توسط جاواکاپ به صورت عمومی منتشر نشده است و به صورت اختصاصی در اختیار شما یا شرکت شما قرار گرفته، بازنشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست



سرفصل مطالب

- ظرف‌ها و ساختمان داده‌ها در جاوا
- واسط‌ها و کلاس‌های مهم در این زمینه
- Collection, Set, List, Map
- ArrayList, LinkedList, HashSet, HashMap
- اشیاء مبتنی بر Hash و کاربرد متد hashCode
- مفهوم Iterator
- ترتیب و مقایسه اشیاء
- کلاس‌های کمکی Arrays و Collections
- سایر کلاس‌ها و واسط‌های مهم در زمینه ظرف‌ها
- آشنایی اولیه با کلاس‌های عام (Generics)



مقدمه

ظرف‌ها و ساختمان‌های داده

- جاوا دارای امکانات متنوعی برای نگهداری اشیاء است
- کلاس‌هایی مثل انواع لیست‌ها، مجموعه‌ها، جدول‌ها و ...
- هر یک از این کلاس‌ها، یک ساختمان داده (Data Structure) است
- هر نمونه ساختمان داده، یک ظرف (container) برای نگهداری اشیاء است
- امکانات و الگوریتم‌هایی بر روی اشیاء داخل ظرف هم پشتیبانی می‌شود
- مانند جستجو، تبدیل به انواع دیگر، مرتب‌سازی و ...
- امکاناتی که جاوا به این منظور ساخته : Java collections framework
- کتابخانه‌ای از کلاس‌ها و واسطه‌هایی که ساختمان‌های داده مختلف را ایجاد می‌کنند





لیست: نیازی که با آرایه تأمین نمی‌شود

محدودیت آرایه‌ها

- می‌دانیم آرایه، امکانی برای ایجاد ظرفی از اشیاء است

- مثال: فرض کنید آرایه‌ای از دانشجویان داریم

```
Student[] students = new Student[size];
```

- اما آرایه‌ها محدودیت‌هایی دارند. مثلاً نیازمندی‌های زیر را در نظر بگیرید:

- اگر طول موردنیاز آرایه (size) را پیشاپیش ندانیم، چه کنیم؟

- اگر بخواهیم بعد از ساختن یک آرایه، طول آن را افزایش دهیم چه کنیم؟

- اگر بخواهیم بعضی از عناصر و اعضای آرایه را حذف کنیم، چه کنیم؟

- راه‌حل ساده‌ای برای موارد فوق در آرایه‌ها وجود ندارد

- مثلاً متدی که یک خانه از آرایه را حذف کند یا طول آرایه را بیشتر کند



امکاناتی که آرایه‌ها ندارند

- تصور کنید که می‌توانستیم از آرایه‌ها، این‌گونه استفاده کنیم:

```
Student[] students = new Student[0];  
students.add(new Student("Ali Alavi"));  
students.add(new Student("Taghi Taghavi"));  
System.out.println(students[1]);  
students.remove(0);
```



- یعنی یک آرایه به طول صفر بسازیم
و بعداً عناصری به آن اضافه کنیم
یا برخی از عناصر آن را حذف کنیم
- اما چنین کاری با آرایه ممکن نیست و کد فوق اشکال نحوی دارد



ظرف‌هایی از اشیاء

- آن‌چه مطرح شد: نیاز به ظرفی از اشیاء (**object container**)
 - در واقع آرایه هم یک ظرف از اشیاء است، ولی محدودیت‌هایی دارد
 - امکاناتی مثل کم و زیاد کردن شیء از مجموعه داخل ظرف، در آرایه‌ها وجود ندارد
 - در جاوا، کلاس‌های مختلفی به عنوان ظرفی از اشیاء عمل می‌کنند
 - این کلاس‌ها امکانات موردنظر که در آرایه‌ها نیست، پشتیبانی می‌کنند
 - کلاس‌هایی مانند:
- ArrayList ، LinkedList ، HashSet ، HashMap و ...



مثال: کلاس ArrayList

- نمونه کاربرد کلاس `java.util.ArrayList`
- `ArrayList` مانند آرایه‌ای است که امکان تغییر اندازه (طول) آن وجود دارد (resizable array)

```
ArrayList students = new ArrayList();  
students.add(new Student("Ali Alavi"));  
students.add(new Student("Taghi Taghavi"));  
students.remove(0);
```

- در ابتدا، `ArrayList` خالی است، به مرور می‌توانیم عناصری به این فهرست اضافه یا کم کنیم
- شیء `students` در کد فوق، مانند ظرفی است که اشیاء مختلفی را در خود نگه می‌دارد
- اشکال شیء `students`: هر شیئی از هر نوعی قابل افزودن به `students` است
- اما معمولاً اعضای از یک جنس را در یک ظرف قرار می‌دهیم



محدود کردن نوع اشیاء لیست

```
ArrayList s = new ArrayList();  
s.add(new Student("Ali Alavi"));  
s.add("Taghi Taghavi");  
s.add(new Object());
```

- کد روبرو خطای کامپایل ندارد

- ولی معمولاً نمی‌خواهیم اجازه دهیم

که یک ظرف اشیائی از انواع مختلف را نگه دارد

- لیست‌ها، می‌توانند نوع اشیاء درون خود را مشخص کنند

- در کد زیر، به ظرف students فقط اشیائی از نوع Student می‌توان اضافه کرد:

```
ArrayList<Student> students = new ArrayList<Student>();
```

- به این تکنیک، اشیاء عام (generics) گفته می‌شود (بعداً در این باره صحبت می‌کنیم)

```
students.add(new Student("Ali Alavi"));  
students.add("Taghi Taghavi");  
students.add(new Object());
```



- مثال:



مثالهایی از ArrayList

```
ArrayList<Student> students = new ArrayList<Student>();  
students.add(new Student("Ali Alavi"));  
students.add(new Student("Taghi Taghavi"));  
students.remove(0);  
students.remove(new Student("Ali Alavi"));  
Student student = students.get(0);  
System.out.println(student);
```

```
ArrayList<String> names = new ArrayList<String>();  
names.add("Ali Alavi");  
names.add("Taghi Taghavi");  
names.remove(0);  
names.remove("Ali Alavi");  
String name = names.get(0);  
System.out.println(name);
```





واسط `java.util.List`

درباره واسط List

```
public class ArrayList<E>  
implements List<E>
```

- برخی متدهای مهم کلاس ArrayList :
 - `int size()` : طول فهرست
 - `boolean isEmpty()` : فهرست خالی است یا خیر
 - `boolean contains(Object o)` : وجود شیء موردنظر در فهرست
 - `add(E e)` : یک عضو به فهرست اضافه می کند
 - `remove(Object o)` : یک عضو از فهرست حذف می کند
 - `remove(int index)` : عضوی با شماره اندیس موردنظر را حذف می کند
 - `clear()` : همه اعضای فهرست را حذف می کند
 - `get(int index)` : عضوی که در اندیس موردنظر است را برمی گرداند
 - `indexOf(Object o)` : شماره اندیس عضو موردنظر را برمی گرداند
- نکته: کلاس ArrayList واسط `java.util.List` را پیاده سازی کرده است
- متدهای فوق همگی در واسط List تعیین شده اند



```

List<String> list = new ArrayList<String>();
Scanner scanner = new Scanner(System.in);
while (true) {
    String input = scanner.next();
    if (input.equalsIgnoreCase("exit"))
        break;
    list.add(input);
}
if (list.isEmpty()) {
    System.out.println("No string entered");
} else {
    System.out.println(list.size());
    if (list.contains("Ali"))
        System.out.println("Ali Found!");

    for (String s : list) {
        System.out.println(s);
    }
}

```

مثال

(for each)



```
interface List<E> {  
    int size();  
    boolean isEmpty();  
    boolean contains(Object o);  
    boolean add(E e);  
    boolean remove(Object o);  
    void clear();  
    E get(int index);  
    E set(int index, E element);  
    void add(int index, E element);  
    E remove(int index);  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
    List<E> subList(int fromIndex, int toIndex);  
}
```



- **ArrayList** ظرفی از اشیاء است: هر یک از مقادیر داخل آن، یک شیء است
- انواع داده اولیه (primitive types) نمی‌توانند در ArrayList قرار گیرند
- این محدودیت برای سایر انواع ظرف‌ها (مثل LinkedList و Set و ...) هم وجود دارد
- در واقع این محدودیت برای همه انواع عام (generics)، از جمله ظرف‌ها، وجود دارد
- این محدودیت برای آرایه وجود ندارد
- مثلاً `ArrayList<int>` غیرممکن است، ولی `int[]` مجاز است



تمرین عملی برای ArrayList

- ایجاد فهرستی از
 - اعداد
 - رشته‌ها
 - دانشجویان
- مرور: فهرست از انواع اولیه ممکن نیست
- استفاده از متدهای متنوع List برای این اشیاء
- تعریف شیء با ارجاع List و نمونه‌سازی با ArrayList
- تأکید بر import برای List و ArrayList





آرایه بهتر است یا ArrayList ؟

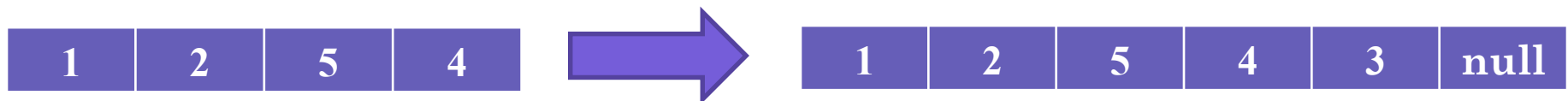
نگاهی به پیاده‌سازی کلاس ArrayList

```
public class ArrayList<E> implements List<E>, ... {  
    private Object[] elementData;  
    private int size;  
    public boolean add(E e) {  
        ensureCapacity(size + 1);  
        elementData[size++] = e;  
        return true;  
    }  
    public ArrayList(int initialCapacity) {  
        ...  
        this.elementData = new Object[initialCapacity];  
    }  
}
```

- برای حذف یک عضو ArrayList، خانه‌های بعدی در خانه قبلی کپی می‌شوند



- هنگام اضافه کردن یک عضو به ArrayList (مثلاً با کمک متد add)
- اگر آرایه‌ای که در دل ArrayList است حافظه کافی نداشته باشد (پر باشد)، یک آرایه جدید بزرگتر ایجاد می‌شود (معمولاً ۵۰٪ بزرگتر می‌شود) و همه اعضای آرایه قبلی در این آرایه کپی می‌شوند
- مثلاً اگر list یک ArrayList باشد که هر چهار خانه آرایه داخل آن پر باشد
- با فراخوانی `list.add(new Integer(3))` خواهیم داشت:



- حذف و اضافه از ArrayList ممکن است منجر به تعداد زیادی کپی ناخواسته شود



آرایه یا ArrayList؟ مسأله این است...

- گاهی آرایه و گاهی ArrayList بهتر است
- در هر کاربرد، باید انتخاب کنیم: مزایا و معایب هر یک را بررسی کنیم
- مزایای آرایه:
- امکان استفاده از انواع داده اولیه (مثل int و double)
- آرایه می‌تواند کارایی (performance) بیشتری داشته باشد
- مزایای ArrayList:
- ارائه متدها و امکاناتی که در آرایه نیست
- مانند اضافه و کم کردن اعضا به صورت پویا، جستجو در لیست و ...
- یادآوری: کلاس ArrayList با کمک یک آرایه پیاده‌سازی شده است
- در دل هر شیء از جنس ArrayList یک آرایه قرار دارد



تبدیل آرایه به ArrayList

- گاهی لازم است یک آرایه را به یک ArrayList تبدیل کنیم، یا برعکس
- مثال برای تبدیل آرایه به ArrayList:

```
String[] strings = {"ali", "taghi"};  
ArrayList<String> list = new ArrayList<String>();  
for (String str : strings)  
    list.add(str);
```

- مثال برای تبدیل ArrayList به آرایه:

```
String[] array = new String[list.size()];  
for (int i = 0; i < array.length; i++)  
    array[i] = list.get(i);
```

- راههای دیگری هم وجود دارد (بعداً می بینیم)

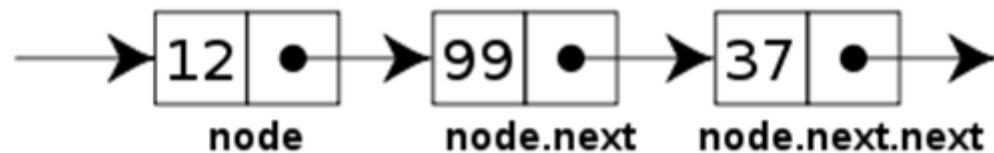




لیست پیوندی (LinkedList)

مفهوم لیست پیوندی (Linked List)

- لیست پیوندی یک ساختمان داده است (data structure)
- که در آن، برخلاف آرایه، همه اعضا پشت سرهم در حافظه قرار نمی‌گیرند
- بلکه هر عضو فهرست، محل (آدرس یا ارجاع) عضو بعدی را نگه می‌دارد

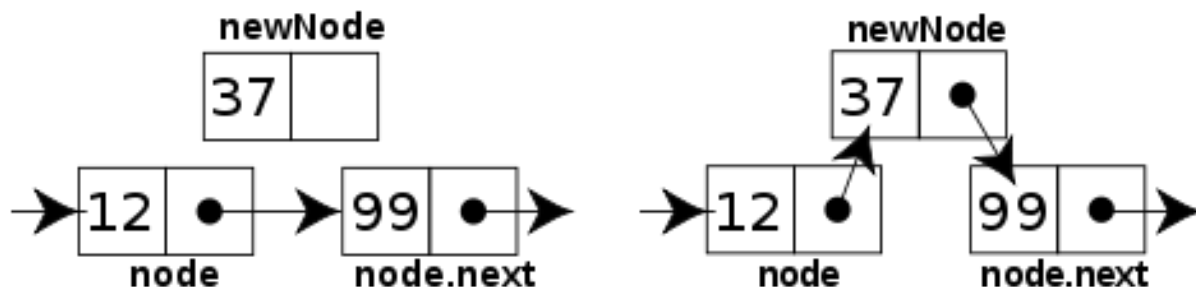


- برای اضافه کردن یک عضو به فهرست:
- یک شیء جدید ایجاد می‌شود
- و آخرین ارجاع (اشاره‌گر) به این شیء جدید اشاره خواهد کرد
- برای حذف یک عضو از فهرست: کافیست اشاره‌گر به این شیء، به شیء بعدی اشاره کند

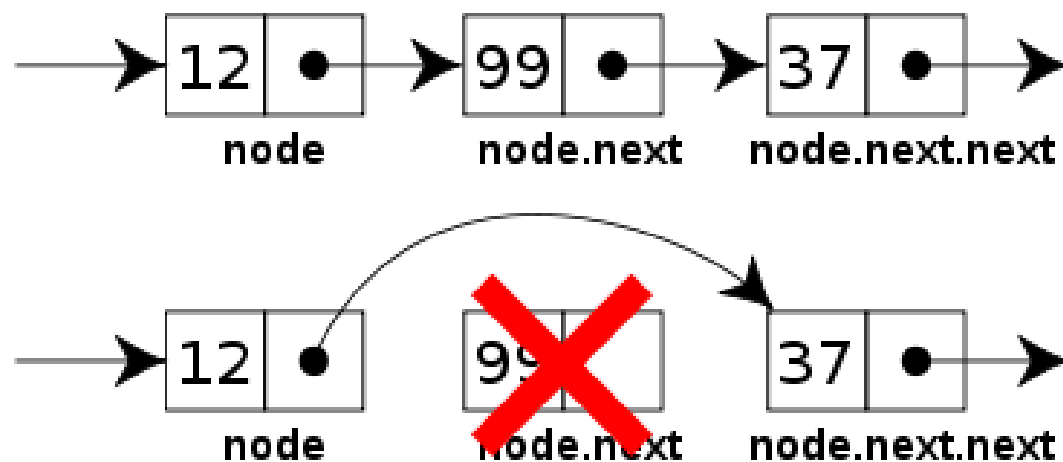


مرور حذف و اضافه به لیست پیوندی

● اضافه به لیست:



● حذف از لیست:



کلاس LinkedList

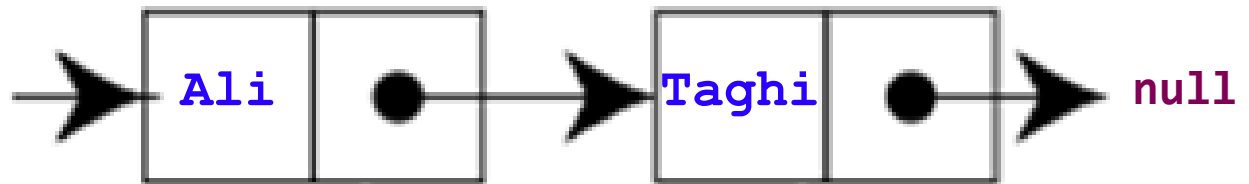
- کلاس `java.util.LinkedList` در جاوا پیاده‌سازی شده است
- یک لیست پیوندی دوطرفه که هر عضو، ارجاع به بعدی و قبلی دارد
- کلاس `LinkedList` هم مانند `ArrayList` واسط `List` را پیاده‌سازی کرده است
- پس همه متدهای مهم `List` را دارد، مانند `add` ، `get` ، `remove` و ...
- بنابراین نحوه کاربرد `LinkedList` مشابه `ArrayList` است
- ولی کارایی (performance) آن‌ها متفاوت است

```
LinkedList<Double> grades = new LinkedList<Double>();  
grades.add(new Double(18.5));  
grades.add(new Double(19.5));  
grades.add(new Double(17.5));  
for (Double d : grades)  
    System.out.println(d);
```



مثال برای لیست پیوندی

```
List<String> list = new LinkedList<String>();  
list.add("Ali");  
list.add("Taghi");  
System.out.println(list.get(1));  
list.remove("Taghi");  
for (String string : list) {  
    System.out.println(string);  
}
```



ArrayList بهتر است یا LinkedList؟

- کلاس‌های ArrayList و LinkedList واسط مشابهی را پیاده کرده‌اند (List)
- اما پیاده‌سازی متفاوتی دارند: درون هر LinkedList یک آرایه نیست، یک لیست پیوندی است
- در مجموع، کلاس ArrayList پرکاربردتر است
- البته در برخی موارد، استفاده از LinkedList کارتر است
- مثلاً: تعداد زیادی add و remove در لیست ← معمولاً لیست پیوندی بهتر است
- گاهی ArrayList برای افزودن یا حذف، مجبور به کپی تعداد زیادی از عناصر موجود می‌شود
- دسترسی فراوان به عناصر با کمک اندیس ← ArrayList بهتر است
- هزینه اجرای get(i) در ArrayList کم است
- ولی در لیست پیوندی i عنصر باید پیمایش شوند تا به عنصری با اندیس i برسیم



تمرین

- کار با متدهای متنوع List
- با کمک ArrayList و LinkedList



کوییز

- در کد زیر، متغیر list می‌تواند شیئی از نوع ArrayList یا LinkedList باشد

```
for(int i=0;i<1000000;i++){  
    for(int j=0;j<100;j++){  
        list.add(0, new Object());  
    }  
    for(int j=0;j<100;j++){  
        list.remove(0);  
    }  
}
```

- در کدام حالت این کد سریع‌تر اجرا می‌شود؟

```
List<Object> list = new ArrayList<Object>();  
List<Object> list = new LinkedList<Object>();
```



- تعداد زیادی حذف و اضافه در ابتدای ArrayList منجر به شیفت‌های فراوان می‌شود



- در کد زیر، list فهرستی از نوع ArrayList یا LinkedList است و شامل تعداد زیادی شیء است

```
Random random = new Random();  
Object temp;  
for(int i=0;i<100000;i++)  
    temp = list.get(random.nextInt(list.size()));
```

- اگر list یک ArrayList باشد کد فوق سریع تر اجرا می شود یا LinkedList؟

• پاسخ: ArrayList

- کد فوق، به دفعات به سراغ اندیسی تصادفی در میانه لیست می رود
- دسترسی به یک اندیس با متد get در ArrayList به مراتب سریع تر است



مجموعه (Set)

مجموعه (Set)

- معنای «مجموعه» در ریاضیات را به خاطر بیاورید:
 - تعدادی شیء متمایز که لزوماً بین اعضا ترتیبی وجود ندارد
 - مثلاً دو مجموعه زیر با هم برابر هستند
- $$\{1,2,3,1,4,2\} = \{4,3,2,1\}$$
- مجموعه (Set) یک واسط در جاوا است: `java.util.Set`
 - یکی از کلاس‌های جاوا که واسط `Set` را پیاده‌سازی می‌کند: `HashSet`

● مثال:

```
HashSet<String> set= new HashSet<String> ();  
set.add("Ali");  
set.add("Taghi");  
set.add("Naghi");
```



```

Set<String> set = new HashSet<String>();
set.add("Ali");
set.add("Taghi");
set.add("Taghi");
set.add("Ali");
set.add("Taghi");
System.out.println(set.size()); 2
for (String str : set)
    System.out.println(str); Taghi
                             Ali
set.remove("Ali");
System.out.println(set.contains("Ali")); false
System.out.println(set.contains("Taghi")); true
set.clear();
System.out.println(set.size()); 0
    
```



تفاوت‌های اصلی List و Set

- اشیاء داخل یک Set متمایز هستند، شیء تکراری در Set وجود ندارد
- اگر تلاش کنید شیئی تکراری به مجموعه اضافه شود، مجموعه تغییری نمی‌کند
- شیء تکراری: شیئی که با یکی از اعضای موجود مجموعه برابر است (براساس متد equals)
- اعضای List ترتیب دارند. بین اعضای Set لزوماً ترتیبی وجود ندارد
- واسط Set هیچ متدی که با اندیس کار کند، ندارد
- مثلاً در واسط Set ، متد `get(i)` نداریم، ولی در List داریم
- متدهای دیگری مثل موارد زیر هم در Set وجود ندارد:
- `set(int index, E element)`
- `int indexOf(Object o)`
- `int lastIndexOf(Object o)`
- `remove(int index)`



مجموعه یا لیست؟ کدام بهتر است؟

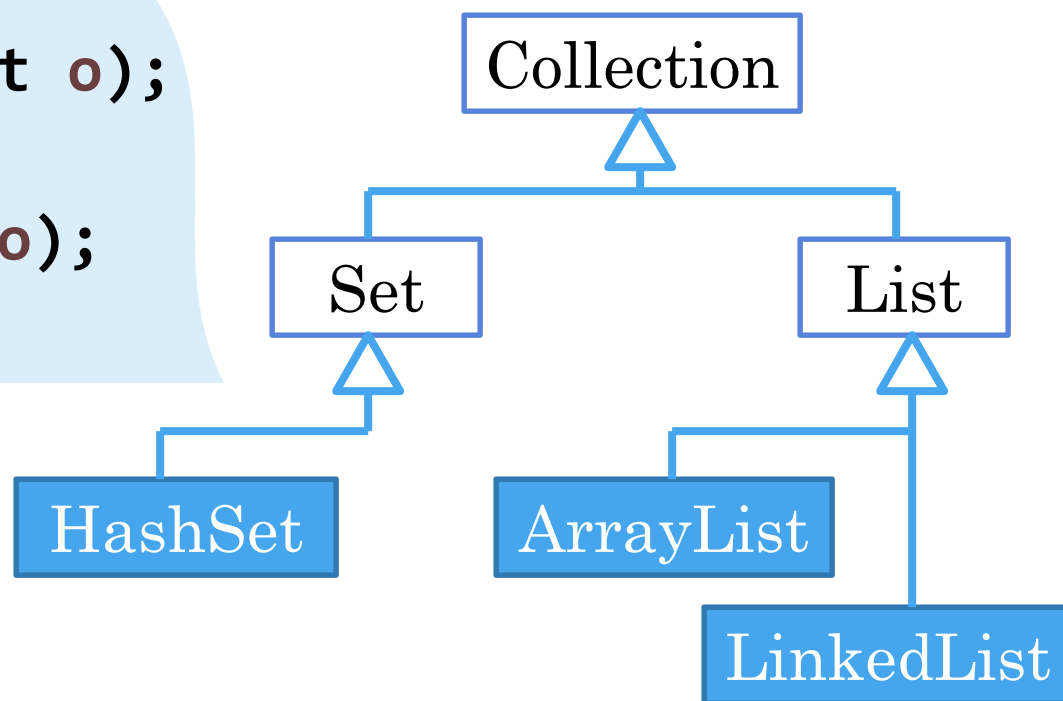
- در برخی کاربردها List و در برخی دیگر Set مناسب‌تر است
- List دسترسی به اعضا از طریق اندیس را ممکن می‌کند
- Set اجازه افزودن عضو تکراری به مجموعه را نمی‌دهد
 - تکراری بودن عضو جدید را چک می‌کند (سر بار محاسباتی)
 - می‌تواند از هدر رفتن حافظه جلوگیری کند (کاهش حافظه مصرفی)
- سؤال کلیدی: آیا در فهرست موردنظر، عضو تکراری مجاز است؟
 - اگر بله: List بهتر است، وگرنه Set بهتر است. مثال:
 - فهرست شماره دانشجویی اعضای یک دانشگاه: Set بهتر است
 - فهرست نمرات یک درس: List بهتر است (نمره تکراری ممکن است)



Collection

- واسط `java.util.Collection` در جاوا وجود دارد
- `List` و `Set` زیرواسط `Collection` هستند
- برخی از متدهای مهم `Collection` :

```
int size();  
boolean isEmpty();  
boolean contains(Object o);  
boolean add(E e);  
boolean remove(Object o);  
void clear();
```



تبدیل Collection به آرایه

- واسط Collection دو متد با نام toArray برای تبدیل به آرایه معرفی می کند:
- روش اول: `Object[] toArray()`
 - این متد پارامتری نمی گیرد
 - فهرست را به یک آرایه از Object ها تبدیل می کند
 - بدین ترتیب نوع واقعی اشیاء در آرایه معلوم نیست
- روش دوم: `T[] toArray(T[] a)`
 - در این روش، آرایه ای از اشیاء به عنوان پارامتر ارسال می شود
 - مقدار برگشتی از نوع داده مشخص شده است
 - اگر پارامتر موردنظر به اندازه کافی فضا داشته باشد، همان را پر می کند
 - وگرنه، یک آرایه جدید از همان جنس می سازد



```
ArrayList<Integer> list = new ArrayList<Integer>();  
list.add(new Integer(5));  
list.add(new Integer(4));  
list.add(new Integer(3));
```

```
Object[] array = list.toArray();  
for (Object object : array) {  
    Integer i = (Integer) object;  
    System.out.println(i);  
}
```

```
Integer[] array2 = list.toArray(new Integer[list.size()]);  
for (Integer i : array2)  
    System.out.println(i);  
Integer[] array3 = list.toArray(new Integer[0]);  
for (Integer i : array3)  
    System.out.println(i);
```

تمرین

- استفاده از Set و HashSet

- مرور ویژگی‌های Set : اعضای متمایز، عدم وجود ترتیب

- استفاده از Collection





اهمیت متدهای hashCode و equals

اهمیت تعریف متد equals در ساختمان داده‌های جاوا

- بسیاری از ساختمان داده‌های جاوا تساوی اعضای فهرست را بررسی می‌کنند
- مثلاً: متد contains به دنبال یک شیء مساوی شیء موردنظر می‌گردد
- این کار با کمک متد equals انجام می‌شود
- متد equals روی اشیاء فهرست فراخوانی می‌شود و شیء موردنظر به آن پاس می‌شود
- متدهایی مثل indexOf(Object o) و remove(Object o) نیز equals را صدا می‌کنند
- در مجموعه‌ها (مثل HashSet) تکراری بودن عضو جدید با کمک equals بررسی می‌شود
- بنابراین اگر بخواهیم ظرفی از جنس یک کلاس دلخواه داشته باشیم، باید متد equals مناسبی برای کلاس موردنظر پیاده‌سازی شده باشد



```
class Student {
    private String name;
    public Student(String name) {
        this.name = name;
    }
}
```

```
List<Student> list = new ArrayList<Student>();
list.add(new Student("Ali"));
System.out.println(list.contains(new Student("Ali")));
```

false

• راه حل: باید برای کلاس Student متد equals مناسبی پیاده کنیم

```
public boolean equals(Object obj) {
    Student other = (Student) obj;
    if (!name.equals(other.name))
        return false;
    return true;
}
```

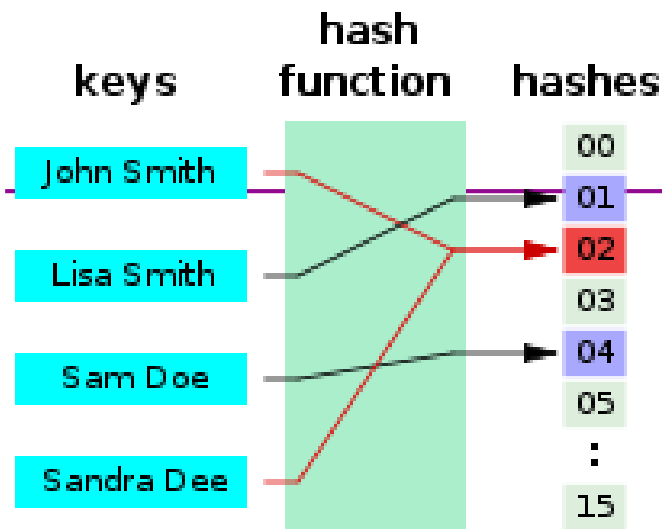
• مثلاً:

• البته متد equals فوق کامل و دقیق نیست

• جزئیاتی مثل null بودن پارامتر را بررسی نمی کند



ساختمان‌های داده مبتنی بر Hash



- برخی از ساختمان داده‌های جاوا مبتنی بر تکنیک Hash هستند (مانند HashSet و HashMap)
- تکنیک Hash :

- از هر شیء که قرار است ذخیره شود، یک عدد صحیح استخراج شود
- این عدد صحیح (hash)، مبتنی بر ویژگی‌های داخل شیء محاسبه شود
- از hash برای محاسبه محل ذخیره شیء استفاده می‌شود
- ممکن است دو شیء مقدار hash مساوی داشته باشند
- ولی تابع hash مناسب، اعدادی حتی‌الامکان متفاوت برای اشیاء متفاوت برمی‌گرداند
- دو شیء با ویژگی‌های مساوی، باید مقدار hash مساوی برگردانند (مقدار hash تصادفی نیست)



متد hashCode

- برخی از ساختمان داده‌های جاوا مبتنی بر تکنیک Hash هستند
- این کلاس‌ها، علاوه بر متد equals از متد hashCode استفاده می‌کنند
- متد hashCode از کلاس Object به همه کلاس‌ها به ارث می‌رسد
- می‌توانیم این متد را override کنیم و معنای مناسبی برای آن پیاده کنیم
- با کمک hashCode یک شیء به یک عدد صحیح (hash) تبدیل می‌شود
- از hash برای جایابی در حافظه و دسترسی سریع به اشیاء استفاده می‌شود
- متد hashCode مناسب، از فیلدهای شیء استفاده می‌کند و عددی حتی‌الامکان متفاوت برمی‌گرداند
- از امکانات IDE (مثلاً eclipse) برای تولید متدهای equals و hashCode استفاده کنید
- اگر برای مقایسه دو شیء متد equals مقدار true برمی‌گرداند، متد hashCode این دو شیء هم باید مساوی باشند، و نه لزوماً برعکس.



```
class Student {
    private String name;
    public Student(String name) {
        this.name = name;
    }
}
```

```
Set<Student> set = new HashSet<Student>();
set.add(new Student("Ali"));
System.out.println(set.contains(new Student("Ali")));
```

false

● راه حل:

- باید برای کلاس Student هم متد equals و هم hashCode مناسبی پیاده کنیم
- پیاده‌سازی equals کافی نیست، زیرا HashSet مبتنی بر hashCode کار می‌کند
- مثلاً:

```
public int hashCode() {
    return 31 + ((name == null) ? 0 : name.hashCode());
}
```



تمرین

● نقش hashCode و equals

- تولید خودکار این متدها
- استفاده در لیست و مجموعه
- تأکید و یادآوری: متد equals باید پارامتری از نوع Object بگیرد



نگاشت (Map)

نگاشت (Map)

- کلاس‌ها و واسط‌هایی که تا این‌جا دیدیم، همه Collection بودند
 - Collection, List, ArrayList, LinkedList, Set, HashSet, ...
- واسط دیگری به نام `java.util.Map` وجود دارد که یک Collection نیست
- یک Map مانند یک جدول یا نگاشت از اشیاء عمل می‌کند
 - همانند جدولی که دو ستون دارد (هر سطر یک زوج مرتب)
 - ستون اول را کلید (Key) و ستون دوم را مقدار (Value) می‌گویند
 - اعضای ستون اول (کلیدها) یکتا هستند: کلید تکراری نداریم
 - اعضای ستون دوم (مقادیر) ممکن است تکراری باشند
- مثال: یک map شامل نمرات دانشجویان:
(جدول یا نگاشتی از رشته‌ها به اعداد حقیقی)

| کلید | مقدار |
|----------|-------|
| علی علوی | ۱۸.۵ |
| تقی تقوی | ۱۹.۵ |
| نقی نقوی | ۱۸.۵ |

درباره Map

- نوع ستون اول و ستون دوم قابل تعیین است
- مثلاً در `Map<String, Double> map;` ← یک جدول است که:
 - کلید آن (ستون اول) رشته‌ها و مقادیر آن (ستون دوم) اعداد حقیقی هستند (نگاشتی از رشته به عدد حقیقی)
- `Map<Integer, Student>` ← نگاشتی از عدد صحیح به دانشجو
- هر نوع شیئی به عنوان کلید یا مقدار، قابل استفاده است
- انواع داده اولیه مثل `int` و `double` در هیچ‌یک از ظرف‌های جاوا قابل استفاده نیستند
- `Map` یک واسط است، یکی از کلاس‌هایی که `Map` را پیاده‌سازی کرده: `java.util.HashMap`




```
Map<Integer, String> map = new HashMap<Integer, String>();  
map.put(87300876, "Ali Alavi");  
map.put(87234431, "Taghi Taghavi");  
map.put(87300876, "Naghi Naghavi");  
String name = map.get(87300876);  
System.out.println(name);  
System.out.println(map.get(87234431));
```

Naghi Naghavi

Taghi Taghavi

• یادآوری:

- در کد فوق به جای Integer از int استفاده شده است
- تبدیل int به Integer به صورت خودکار انجام می شود (auto-boxing)
(از جاوا ۵ به بعد)

- نوع مورد استفاده در همه کلاس های java collections framework باید شیء باشند



نگاهی به واسط Map

```
public interface Map<K,V> {  
    V get(Object key);  
    V put(K key, V value);  
    int size();  
    boolean isEmpty();  
    boolean containsKey(Object key);  
    boolean containsValue(Object value);  
    V remove(Object key);  
    void putAll(Map m);  
    void clear();  
    Set<K> keySet();  
    Collection<V> values();  
}
```



تغییر مقدار با کمک put

- اگر سطری با کلید تکراری به یک map اضافه شود: مقدار قبلی آن کلید حذف می‌شود
- مثال: `Map<Integer, String> map = new HashMap<Integer, String>();`

- `map.put(76, "Ali")`

| | |
|----|-----|
| 76 | Ali |
|----|-----|

- `map.put(31, "Taghi")`

| | |
|----|-------|
| 76 | Ali |
| 31 | Taghi |

- `map.put(76, "Naghi")`

| | |
|----|-------|
| 76 | Naghi |
| 31 | Taghi |



```
Map<Student, Double> map = new HashMap<Student, Double>();  
map.put(new Student("Ali Alavi"), new Double(18.76));  
map.put(new Student("Taghi Taghavi"), new Double(15.43));  
map.put(new Student("Naghi Naghavi"), new Double(17.26));  
map.put(new Student("Naghi Naghavi"), new Double(15.26));  
map.remove(new Student("Naghi Naghavi"));
```

```
Double grade = map.get(new Student("Taghi Taghavi"));  
System.out.println("Grade of Taghi=" + grade);
```

```
for (Student student : map.keySet())  
    System.out.println(student.toString());
```

```
Double totalSum = 0.0;  
for (Double avg : map.values())  
    totalSum += avg;
```

```
System.out.println("Average = " + (totalSum / map.size()));
```

این برنامه به شرطی درست
کار می کند که متدهای
equals و hashCode به
Student خوبی در کلاس
پیاده سازی شده باشند



کوییز

کوئیز: خروجی برنامه زیر چیست؟

```
Map<String, String> map = new HashMap<String, String>();  
map.put("Laptop", "Computers");  
map.put("Shahnameh", "Books");  
map.put("Tablet", "Books");  
map.put("Tablet", "Computers");  
System.out.println(map.size());  
System.out.println(map.get("Tablet"));  
System.out.println(map.get("GOLESTAN"));  
System.out.println(map.containsKey("TABLET"));  
System.out.println(map.containsValue("Books"));
```

3
Computers
null
false
true



تمرین عملی

● تمرین Map

● متدهای

keySet○

values○





مفهوم پیمایشگر (Iterator)

مفهوم پیمایشگر (Iterator)

- تا قبل از جاوا ۵ ، امکان for each برای پیمایش وجود نداشت

- از جاوا ۵ به بعد، for each برای آرایه‌ها و collection ها ممکن شد

- مثال:

```
int[] array = {1,2,3,7}; List<Integer> list ;
for (int i : array) ...
    System.out.println(i); for (Integer i : list)
    System.out.println(i);
```

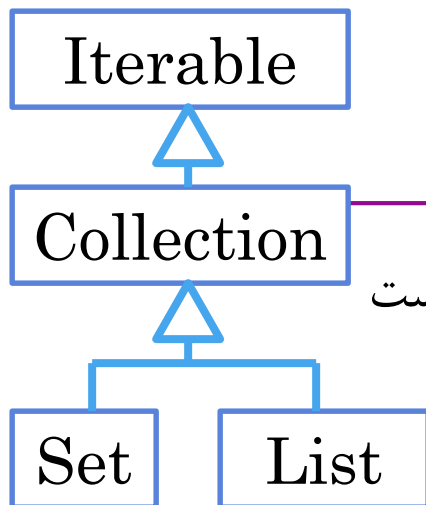
- قبل از جاوا ۵ با کمک iterator پیمایش روی collection ها انجام می‌شد

- این امکان همچنان وجود دارد و کاربردهایی نیز دارد

- مثال:

```
List<Integer> list = ... ;
Iterator<Integer> iterator = list.iterator();
while(iterator.hasNext()){
    Integer i = iterator.next();
    System.out.println(i);
}
```





- متد **iterator** در واسط **java.lang.Iterable** تعریف شده است

- واسط **Collection**، از واسط **Iterable** ارث‌بری کرده است

- بنابراین **List** ها و **Set** ها همگی **Iterable** هستند

- در واقع همه کلاس‌هایی که **Iterable** هستند، امکان **for each** دارند

- امکان **for each** در نسخه‌های جدید جاوا با کمک **iterator** پیاده شده است

```
public interface Iterable<T> {  
    Iterator<T> iterator();  
    ...  
}
```

```
public interface Iterator<E> {  
    boolean hasNext();  
    E next();  
    void remove();  
}
```

```
public interface Collection<E> extends Iterable<E> {...}
```



```
List<Integer> list = new ArrayList<Integer>();
for(int i=0;i<10;i++)
    list.add(i);

Iterator<Integer> iterator = list.iterator();
while(iterator.hasNext()){
    Integer value = iterator.next();
    if(value%2==0)
        iterator.remove();
}
System.out.println(list.size());
```

5



تغییر همزمان در یک ظرف

تغییر همزمان (Concurrent Modification)

- فرض کنید: چند بخش برنامه به صورت همزمان در حال استفاده از یک ظرف باشند (مثلاً یک لیست یا مجموعه)
- و در همین حال، یک بخش از برنامه، تغییری در ظرف ایجاد کند
- مثلاً شیئی به آن اضافه یا کم کند
- این **تغییر همزمان** نباید ممکن باشد
- زیرا یک ظرف توسط یک بخش در حال پیمایش است و در بخش دیگری تغییر می‌کند
- مثلاً شاید در بخشی که پیمایش انجام می‌شود، روی طول ظرف حساب شده باشد
- و یا شیئی که پیمایش و پردازش شده، توسط بخش دیگری از برنامه حذف شود
- جاوا از تغییر همزمان جلوگیری می‌کند



مفهوم شکست سریع (Fail Fast)

- اگر یک ظرف به واسطه یکی از متدهایش تغییر کند، همه `iterator` هایی که قبلاً روی این ظرف گرفته شده، غیرمعتبر می‌شوند
- هر عملیاتی که از این پس روی این `iterator` های غیرمعتبر انجام شود، منجر به پرتاب خطای `ConcurrentModificationException` می‌شود
- به این تکنیک، شکست سریع (Fail Fast) گفته می‌شود
- با تغییر یک ظرف توسط یک `iterator`، سایر `iterator` ها غیرقابل استفاده می‌شوند
- این تکنیک، روش جاوا برای جلوگیری از تغییر همزمان است

```
Collection<String> c = new ArrayList<String>();  
Iterator<String> itr = c.iterator();  
c.add("An object");  
String s = itr.next();
```

مثال:

`itr` نامعتبر می‌شود

پرتاب `ConcurrentModificationException`

مثال دیگری برای ConcurrentModificationException

```
List<String> list = new ArrayList<String>();
```

```
list.add("A");
```

```
list.add("B");
```

```
list.add("C");
```

```
for (String s : list)
```

```
if(s.equals("A"))
```

```
list.remove(s);
```

بلافاصله بعد از اجرای متد remove روی این خط خطا دریافت می کنیم

iterator ای که با کمک آن حلقه for در حال اجراست نامعتبر می شود



کوییز

- متدی بنویسید که لیستی از رشته‌ها به عنوان پارامتر بگیرد
- و همه رشته‌هایی که با Ali شروع می‌شوند را از لیست حذف کند

```
void removeAlis(List<String> names){...}
```



```
void removeAli(List<String> list) {  
    for (String string : list)  
        if (string.startsWith("Ali"))  
            list.remove(string);  
}
```

- این راه حل منجر به ConcurrentModificationException می شود



```
public static void removeAli(List<String> list){  
    Iterator<String> iterator = list.iterator();  
    while(iterator.hasNext()) {  
        String string = iterator.next();  
        if(string.startsWith("Ali"))  
            iterator.remove();  
    }  
}
```



یک راه حل صحیح دیگر

```
public static void removeAli(List<String> list){  
    for (int i = list.size()-1; i >= 0; i--)  
        if(list.get(i).startsWith("Ali"))  
            list.remove(i);  
}
```



مقایسه ترتیب اشیاء

مقایسه دو شیء

- گاهی لازم است دو شیء با هم مقایسه شوند و ترتیب آنها مشخص شود (کدام یک کوچکتر است؟)

- متد `equals` ، فقط تساوی اشیاء را بررسی میکند

- برای بسیاری از انواع داده (کلاس) معنای مشخصی برای «ترتیب» اشیاء وجود دارد
- و این معنا باید برای برخی از انواع داده تعریف شود. مثال:

| | |
|---|-------------------------------|
| "Apple"<"Orange" | $5 < 6$ |
| دانشجو علوی < دانشجو تقوی (براساس معدل) | اول تیر ۱۳۹۴ < پنج خرداد ۱۳۹۴ |

- ولی برای داده‌های غیر عددی، عملگرهای مقایسه‌ای (< و > و <= و >=) کار نمی‌کنند
- کاربرد مقایسه اشیاء: مرتب‌سازی و جستجوی سریع‌تر (مثلاً در ظرفی از اشیاء)



```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

واسط Comparable

- اگر کلاسی واسط Comparable را پیاده کند، یعنی برای اشیاء این شیء ترتیب معنا دارد
- متد compareTo : شیء جاری را با پارامتر متد مقایسه می کند و یک عدد برمی گرداند
- منفی، صفر و مثبت بودن این عدد به ترتیب یعنی این شیء کوچک تر، مساوی یا بزرگ تر از

پارامتر متد است

```
class java.util.Date implements Comparable<Date>{
```

```
    public int compareTo(Date anotherDate) {
        long thisTime = getMillisOf(this);
        long anotherTime = getMillisOf(anotherDate);
        return (thisTime<anotherTime ? -1 :
                (thisTime==anotherTime ? 0 : 1));
    }
}
```

مثال

//Deprecated Constructors:

```
Date d1 = new Date(2015, 10, 21);
Date d2 = new Date(2013, 7, 26);
Date d3 = new Date(2013, 7, 26);
System.out.println(d1.compareTo(d2));
System.out.println(d2.compareTo(d1));
System.out.println(d2.compareTo(d3));
```

1

-1

0

برای هر کلاس جدید که ترتیب اشیاء در آن معنی و اهمیت دارد، کلاس را فرزند Comparable کنید و متد compareTo مناسب برای آن پیاده سازی کنید

واسط Comparator

- گاهی می‌خواهیم اشیاء را با ترتیبی غیر از آنچه خودشان تعریف کرده‌اند مقایسه کنیم
- مثلاً کلاس دانشجو واسط Comparable را پیاده‌سازی کرده و متد compareTo را بر اساس معدل دانشجو تعریف کرده ولی ما می‌خواهیم فهرست دانشجویان را بر اساس «سن» مرتب کنیم (ترتیب بر اساس سن)
- گاهی نیز می‌خواهیم اشیائی را مقایسه کنیم که کلاسشان Comparable نیست
- در این موارد واسط java.util.Comparator را برای مقایسه این اشیاء پیاده‌سازی می‌کنیم

```
public interface Comparator<T> {  
    int compare(T o1, T o2);  
}
```



مثال برای Comparator

```
class Student implements Comparable<Student> {  
    int age;  
    double grade;  
    public int compareTo(Student s) {  
        return (this.grade<s.grade ? -1 :  
                (this.grade==s.grade ? 0 : +1));  
    }  
    public Student(int age, double grade) {  
        this.age = age;  
        this.grade = grade;  
    }  
}
```

```
class StudentComparator implements Comparator<Student>{  
    public int compare(Student s1, Student s2) {  
        return s1.age<s2.age? -1:(s1.age==s2.age?0 : +1);  
    }  
}
```

```
StudentComparator comparator = new StudentComparator();  
Student s1 = new Student(21, 17.5);  
Student s2 = new Student(20, 18.5);  
System.out.println(s1.compareTo(s2));  
System.out.println(comparator.compare(s1,s2));
```

-1
1





کلاس‌های کمکی Collections و Arrays

کلاس‌های Arrays و Collections

- جاوا دو کلاس، با متدهای کمکی مفید برای کار با آرایه‌ها و Collection ها ارائه کرده است
- کلاس `java.util.Arrays` برای کار با آرایه‌ها
- کلاس `java.util.Collections` برای کار با Collection ها
- این کلاس‌ها دارای متدهای استاتیک متنوعی هستند، برای:
 - کپی اشیاء درون آرایه یا ظرف
 - پر کردن همه اعضا با یک مقدار مشخص (`fill`)
 - جستجو (`search`)
 - مرتب‌سازی (`sort`)
 - و ...



مثال برای کاربرد Arrays

```
Random random = new Random();
Long[] array = new Long[100];
Arrays.fill(array, 5L);
Long[] copy = Arrays.copyOf(array, 200);
for (int i = 100; i < copy.length; i++)
    copy[i] = random.nextLong()%10;
//An unmodifiable list:
List<Integer> asList = Arrays.asList(1, 2 , 3, 4);
List<Long> asList2 = Arrays.asList(array);
Arrays.sort(array);
int index = Arrays.binarySearch(array, 7L);
int[] a1 = {1,2,3,4};
int[] a2 = {1,2,3,4};
System.out.println(a1==a2); false
System.out.println(a1.equals(a2)); false
System.out.println(Arrays.equals(a1, a2)); true
System.out.println(a1); [I@7852e922
System.out.println(a1.toString()); [I@7852e922
System.out.println(Arrays.toString(a1)); [1, 2, 3, 4]
```

```

List<String> list = new ArrayList<String>();
Collections.addAll(list, "A", "Book", "Car", "A");
int freq = Collections.frequency(list, "A"); 2
Collections.sort(list); A, A, Book, Car
Comparator<String> comp = new Comparator<String>(){
    public int compare(String o1, String o2) {
        return o1.length() < o2.length() ? -1 :
            (o1.length() == o2.length() ? 0 : +1);
    }
};
Collections.sort(list, comp); A, A, Car, Book
Collections.reverse(list);
String min = Collections.min(list); A
String max = Collections.max(list); Car
String max2 = Collections.max(list, comp); Book
Collections.shuffle(list);
Collections.fill(list, "B");

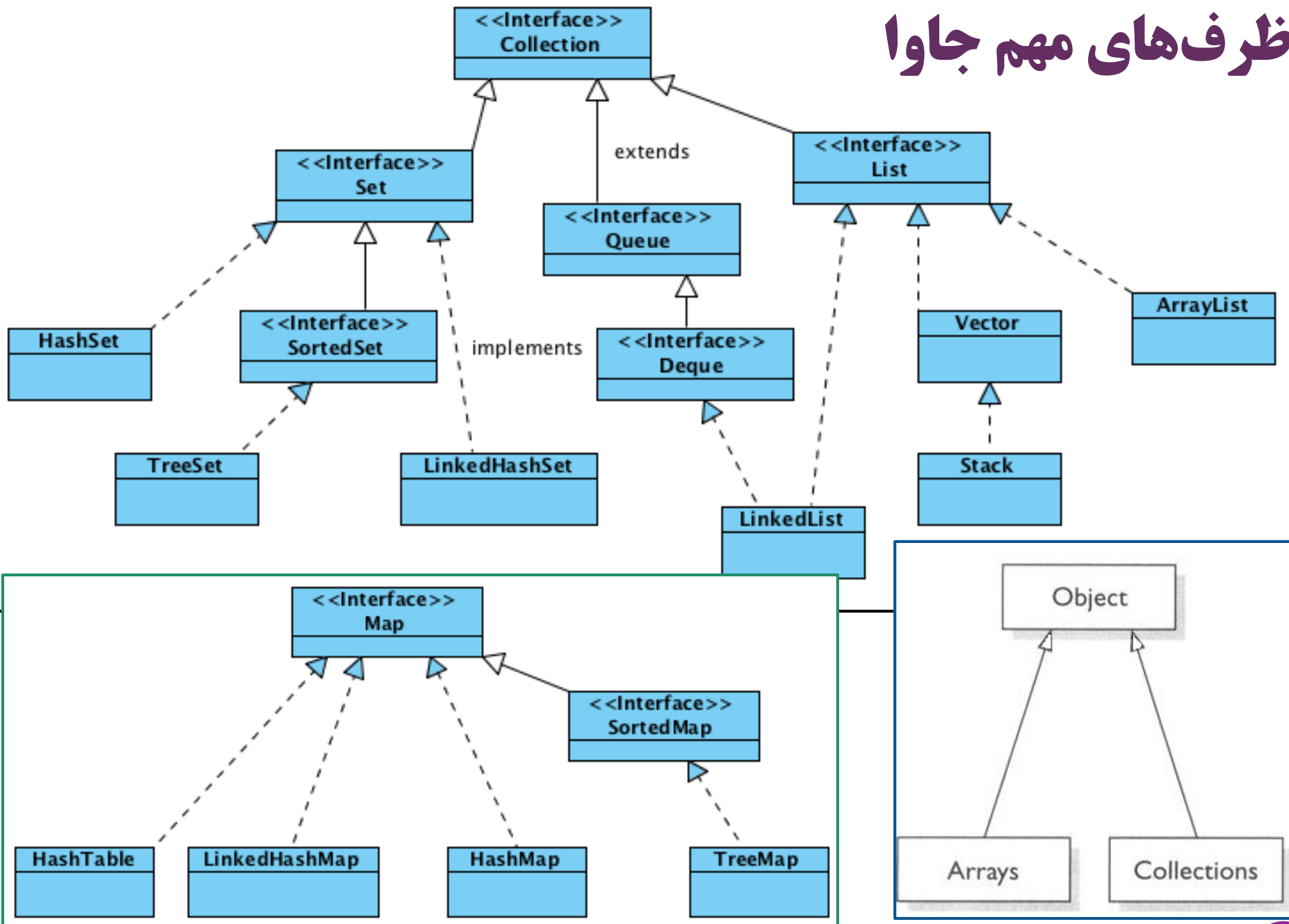
```

مثال برای کاربرد Collections



سایر ظرفها

ظرف‌های مهم جاوا



مرور برخی واسط‌ها و کلاس‌های مهم دیگر

| نام | نوع | پدر | توضیح |
|---------------|------|------------|---|
| SortedSet | واسط | Set | یک مجموعه مرتب |
| TreeSet | کلاس | SortedSet | یک مجموعه مرتب که براساس یک درخت پیاده شده |
| SortedMap | واسط | Map | یک نگاشت (جدول) که بر اساس کلیدهایش مرتب است |
| TreeMap | کلاس | SortedMap | نگاشت مرتبی (براساس کلید) که با درخت پیاده شده |
| Queue | واسط | Collection | یک صف از اشیاء (FIFO) |
| PriorityQueue | کلاس | Queue | یک صف اولویت‌دار (بر اساس مقایسه و ترتیب اشیاء) |
| Stack | کلاس | Vector | یک پشته از اشیاء (LIFO) |



اشاره به برخی مفاهیم

- برخی از ظرف‌ها synchronized هستند (Synchronized Collections)
 - ظرف‌هایی که استفاده از آن‌ها در چند thread همزمان، امن است
 - کلاس‌های thread-safe (مراجعه به مبحث thread)
 - مثل ConcurrentHashMap و Vector
 - اگر نیازی به استفاده همزمان از اشیاء کلاس نیست، از اینها استفاده نکنید
- برخی ظرف‌ها غیرقابل تغییر هستند (Unmodifiable Collections)
 - فقط می‌توانیم اعضای آن‌ها را پیشمایش کنیم (کم‌وزیاد کردن اعضا ممکن نیست)
 - مثال:

```
List<String> unmod1 = Arrays.asList("A", "B");  
List<String> mod1 = new ArrayList<>(unmod1);  
Collection<String> unmod2 = Collections.unmodifiableCollection(mod1);
```



- امکان عدم ذکر نوع عام (generic) هنگام ایجاد شیء

- از جاوا ۷ به بعد

```
List<String> mylist1 = new ArrayList<String>();  
List<String> mylist2 = new ArrayList<>();
```

```
Set<Integer> set1 = new HashSet<Integer>();  
Set<Integer> set2 = new HashSet<>();
```

```
Map<String, Integer> m2 = new HashMap<String, Integer>();  
Map<String, Integer> m1 = new HashMap<>();
```



کوییز

```

class Car implements Comparable<Car> {
    String name;
    Integer price, speed;
    public Car(String name, Integer price, Integer speed) {
        this.name = name;
        this.price = price;
        this.speed = speed;
    }
    public int compareTo(Car o) {
        return this.price.compareTo(o.price);
    }
}

```

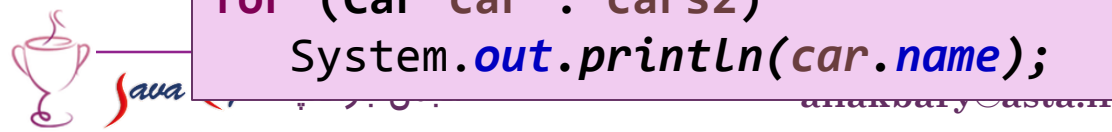
```

Comparator<Car> comp = new Comparator<Car>() {
    public int compare(Car o1, Car o2) {
        return o1.speed.compareTo(o2.speed);
    }
};
Set<Car> cars1 = new TreeSet<>(comp);
Collections.addAll(cars1, new Car("Pride", 20, 200),
    new Car("Samand", 25, 180));
Set<Car> cars2 = new TreeSet<>(cars1);
for (Car car : cars1)
    System.out.println(car.name);
for (Car car : cars2)
    System.out.println(car.name);

```

خروجی
برنامه
زیر
چيست؟

Samand
Pride
Pride
Samand



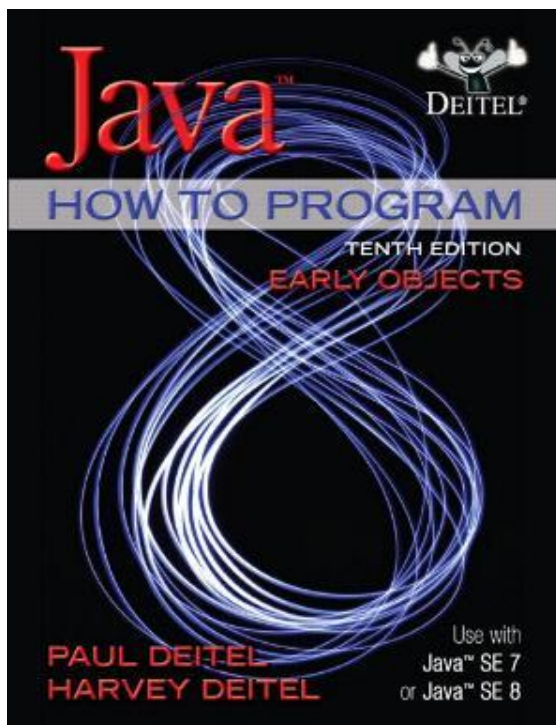
جمع بندی

- واسط‌ها و کلاس‌های مهم در زمینه ظرف‌ها و ساختمان داده‌ها در جاوا
 - Collection, Set, List, Map
 - ArrayList, LinkedList, HashSet, HashMap
- اشیاء مبتنی بر Hash و کاربرد متد hashCode
- مفهوم Iterator
- ترتیب و مقایسه اشیاء: Comparable و Comparator
- کلاس‌های کمکی Arrays و Collections
- کلاس‌ها و واسط‌های مهم در حوزه ظرف‌ها



- فصل ۱۶ کتاب دایتل

Java How to Program (Deitel & Deitel)



16 Generic Collections **684**

- تمرین‌های همین فصل از کتاب دایتل



- تعدادی عدد از کاربر بگیرید و سپس:
- اعداد را به ترتیب صعودی مرتب کنید و سپس چاپ کنید
- اعداد را به ترتیب نزولی مرتب کنید و سپس چاپ کنید
- یک کلاس خودرو تعریف کنید و اطلاعات چندین خودرو را در یک Set قرار دهید
- مفهوم تساوی دو خودرو را با «تساوی نام آنها» پیاده کنید
- مطمئن شوید که اگر دو خودروی همنام به مجموعه اضافه شود، خودروی دومی اضافه نمی‌شود
- جدولی (map) با کلید شماره دانشجویی و مقدار شیء دانشجو ایجاد کنید
- اطلاعات این جدول را از کاربر بگیرید
- متدی بنویسید که این جدول را بگیرد و جدولی با کلید «نمره» و مقدار «تعداد» برگرداند





جستجو کنید و بخوانید

- موضوعات پیشنهادی برای جستجو:
- امکانات جاوا ۸ و جویبارها (Stream)
- ارتباط برنامه‌نویسی multi-thread با مبحث ظرف‌ها
- چه کلاس‌هایی در نسخه‌های قدیمی جاوا وجود داشته‌اند که منسوخ شده‌اند؟
(به خصوص قبل از نسخه ۱.۲)
- چه امکانات دیگری در کلاس‌های کمکی Arrays و Collections هست؟
- چه امکانات مفیدی در این کلاس‌ها پیاده‌سازی نشده است؟
- مثلاً Apache Commons چه امکاناتی فراهم کرده است؟



پایان

- جلسه شماره ۱۴: ظرف ها و ساختمان های داده
- اسلاید شماره ۳۹
- دقیقه ۴۸ در فیلم آموزشی
- آن چه گفته شده بود:
- اگر شیئی تکراری به مجموعه اضافه شود، شیء قدیمی حذف می شود
- آن چه صحیح است:
- اگر تلاش کنید شیئی تکراری به مجموعه اضافه شود، مجموعه تغییری نمی کند
- شیء تکراری: شیئی که با یکی از اعضای موجود مجموعه برابر است (براساس متد equals)
- بنابراین اگر سعی کنید یک شیء تکراری به یک Set اضافه کنید، شیء جدید جایگزین شیء قبلی نمی شود، بلکه شیء قبلی حفظ می شود و شیء جدید به مجموعه اضافه نمی شود



- جلسه شماره ۱۴: ظرف ها و ساختمان های داده

- اسلاید شماره ۹۷

- دقیقه ۱۵۱ در فیلم آموزشی (۲ ساعت و ۳۱ دقیقه بعد از شروع ویدیو)

- آن چه گفته شده بود:

- مطمئن شوید که اگر دو خودروی همنام به مجموعه اضافه شود، خودروی اول حذف می شود

- آن چه صحیح است:

- مطمئن شوید که اگر دو خودروی همنام به مجموعه اضافه شود، خودروی دومی اضافه نمی شود

