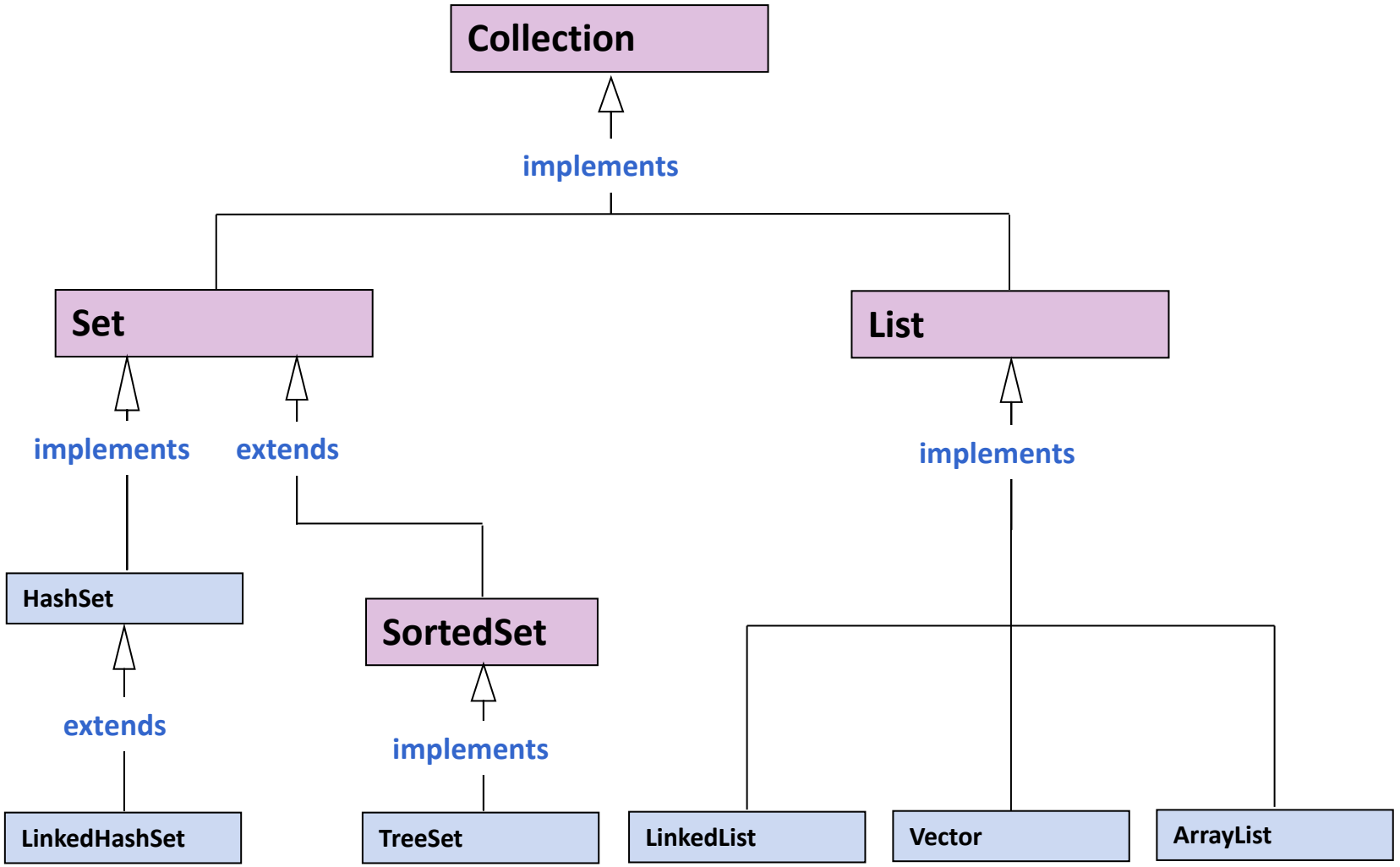
دانشکده مهندسی و علوم کامپیوتر

برنامه نویسی پیشرفته

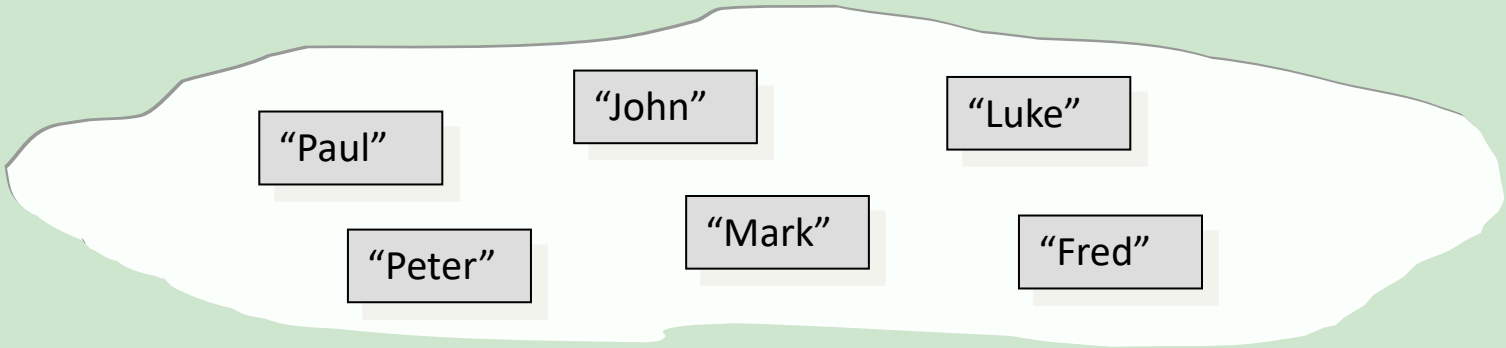
وحیدی اصل

بخش دوم–Collections

Set and List: Collections سلسله مراتب



set واسط



A Set cares about uniqueness, it doesn’t allow duplicates.

HashSet	LinkedHashSet	TreeSet

متدهای set

- یک Set (معادل مجموعه در ریاضی) دارای عناصری بدون ترتیب و تکرار عناصر است. می‌توانیم ترتیب را به کلاسهای آن اضافه کنیم.
- عملیاتش همانهایی هستند که در Collection موجود است.

```
int size( );
boolean isEmpty( );
boolean contains(Object e);
boolean add(Object e);
boolean remove(Object e);
Iterator iterator( );
```

```
boolean containsAll(Collection c);
boolean addAll(Collection c);
boolean removeAll(Collection c);
boolean retainAll(Collection c);
void clear( );
```

```
Object[ ] toArray( );
Object[ ] toArray(Object a[ ]);
```

پیاده‌سازیهای set

- Set یک واسط است و لذا نمی‌توانیم بگوییم: new Set( )
- Set را چهار کلاس زیر پیاده‌سازی می‌کنند:
  - HashSet برای بیشتر کارها بهترین گزینه می‌باشد.
  - TreeSet تضمین می‌کند که iterator عناصر را به طور مرتب‌شده برگرداند و این ترتیب در طول زمان تغییر نکند.
  - LinkedHashSet تضمین می‌کند که iterator عناصر را به ترتیب وارد شدن (insert) به مجموعه، برگرداند.
- AbstractSet یک کلاس انتزاعی است که امکان ایجاد پیاده‌سازیهای جدید را میسر می‌کند.

مثال (گزینه بهتر): `Set s = new HashSet( );`

مثال `HashSet s = new HashSet( );`

پیمایش set

- Set هم دارای متد Iterator iterator() می‌باشد که یک شیئ iterator را برروی مجموعه ایجاد می‌کند و پیمایش برروی اعضای مجموعه را انجام می‌دهد.
- این شیئ، حاوی سه متد زیر است:

```
boolean hasNext()
Object next()
void remove()
```

- می‌توانید از حلقه for تغییریافته برای انجام پیمایش برروی مجموعه استفاده کنید.
- متد remove() به شما امکان می‌دهد در حین پیمایش هر عنصر، عنصر مربوطه را از مجموعه حذف کنید.

```
import java.util.*;

public class SetExample2 {

    public static void main(String[] args) {
        String[ ] words = { "When", "all", "is", "said", "and", "done",
                            "more", "has", "been", "said", "than", "done" };
        Set mySet = new HashSet();

        for (int i = 0; i < words.length; i++) {
            mySet.add(words[i]);
        }
        for (Iterator iter = mySet.iterator(); iter.hasNext();) {
            String word = (String) iter.next();
            System.out.print(word + " ");
        }
        System.out.println();
    }
}
```

عملیات متداول set

- Testing if s2 is a *subset* of s1

```
s1.containsAll(s2)
```

- Setting s1 to the *union* of s1 and s2

```
s1.addAll(s2)
```

- Setting s1 to the *intersection* of s1 and s2

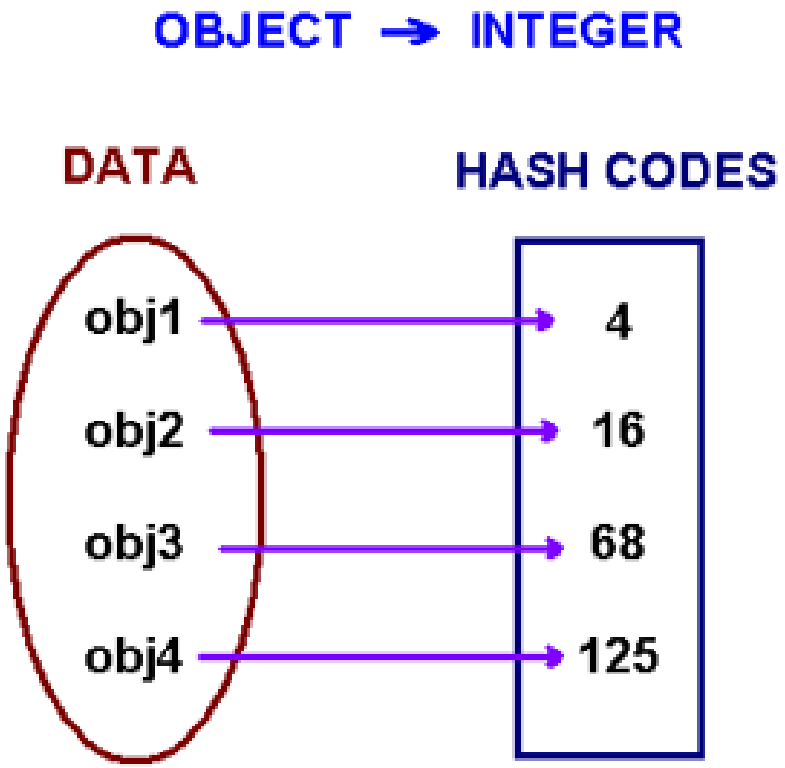
```
s1.retainAll(s2)
```

- Setting s1 to the *set difference* of s1 and s2

```
s1.removeAll(s2)
```

Hashing

- جستجو یک مسئله مهم در ساختمان داده‌ها می‌باشد.
- فرض کنید بخواهیم مقداری را در یک آرایه نامرتب جستجو کنیم.
- برای یافتن عنصر حاوی آن مقدار، تک‌تک عناصر آرایه باید بررسی شوند.
- اگر آرایه مرتب بود، با جستجوی دودویی، پیچیدگی زمانی O(log n) می‌بود.
- اگر بدانیم اندیس مکان احتمالی مقدار موردنظر چیست، سرعت جستجو به مراتب بالا خواهد رفت.
- تصور کنید یک تابع جادویی داریم که به ما می‌گوید، اندیس عنصر موردنظر چیست.
- با این تابع جادویی زمان جستجو به O(1) کاهش خواهد یافت!
- این تابع جادویی، تابع هَش (Hash Function) نامیده می‌شود.
- تابع هَش، تابعی است که با داشتن یک کلید، آدرسی را در جدول هش تولید می‌کند.

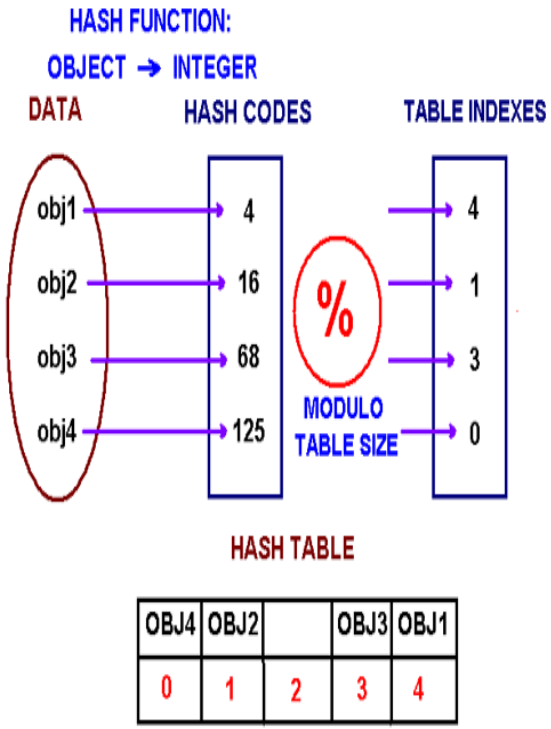


Hashing

Hashing

- مثالی از یک تابع هش، کدکتاب در کتابخانه می‌باشد.
- هر کدکتاب در یک کتابخانه دارای یک شماره منحصربفرد می‌باشد.
- این کد به مثابه یک آدرس بوده و به ما می‌گوید کتاب دقیقاً در کجای کتابخانه قرار دارد.
- این کد، ترکیبی استاندارد از حروف و اعداد می‌باشد که امکان دسته‌بندی موضوعی و ماهیتی کتابها را فراهم می‌کند.
- یک تابع هش که یک شماره هش برمی‌گرداند، **universal hash function** نامیده می‌شود.
- در عمل، اختصاص دادن شماره‌های منحصربفرد به اشیا دشوار می‌باشد.
- برای اینکار از شماره اشیایی که ایجاد و پردازش می‌کنیم، استفاده می‌شود.
- تابع هش باید خصوصیات زیر را داشته باشد:
  - برای یک شیئ، شماره آن را برگرداند.
  - دوشیئی معادل هم، باید یک شماره داشته باشند.
  - دو شیئی نابرابر باید شماره متفاوت داشته باشند.
- فرآیند ذخیره‌سازی اشیا با استفاده از تابع هش به صورت زیر می‌باشد:
  - یک آرایه به اندازه M ساخته می‌شود.
  - یک تابع هش h انتخاب می‌شود که قادر است اشیا را به ترتیب به اعداد M-1,....,1,0 نگاشت کند.
  - این اشیا را درون آرایه در اندیسهایی قرار می‌دهیم که با تابع هش مشخص شده‌اند:

index= h(object)



- به این جدول، اصطلاحاً hash table گفته می‌شود.

یک تابع هش چگونه انتخاب شود؟

- یک روش ایجاد این تابع، استفاده از متد ***hashCode()*** جاوا می‌باشد.
- این متد در کلاس پدرجد (Object) پیاده‌سازی شده است و در نتیجه همه کلاسهای جاوا، آن را به ارث می‌برند.
- کد هش، یک نمایش عددی از یک شیئ ایجاد می‌کند; مشابه متد toString() که اگر بازنویسی نشده باشد، رشته‌ای به عنوان نمایش آن شیئ را برمی‌گرداند:
- ClassName@Hexadecimal Representation of hashCode for that object
- مثال زیر را درنظر بگیرید:

```
Integer obj1 = new Integer(2009);
String obj2 = new String("2009");
System.out.println("hashCode for an integer is " + obj1.hashCode());
System.out.println("hashCode for a string is " + obj2.hashCode());
```

- خروجی:

```
hashCode for an integer is 2009
hashCode for a string is 1537223
```

یک تابع هش چگونه انتخاب شود؟

- متد hashCode در کلاسهای مختلف به شیوه‌های مختلف پیاده‌سازی می‌شود.
- در کلاس String کد هَش با فرمول زیر محاسبه می‌شود:

- $s.charAt(0) * 31^{n-1} + s.charAt(1) * 31^{n-2} + ... + s.charAt(n-1)$
- *s* is a string and *n* is its length.

- برای مثال:

- $"ABC" = 'A' * 31^2 + 'B' * 31 + 'C' = 65 * 31^2 + 66 * 31 + 67 = 64578$

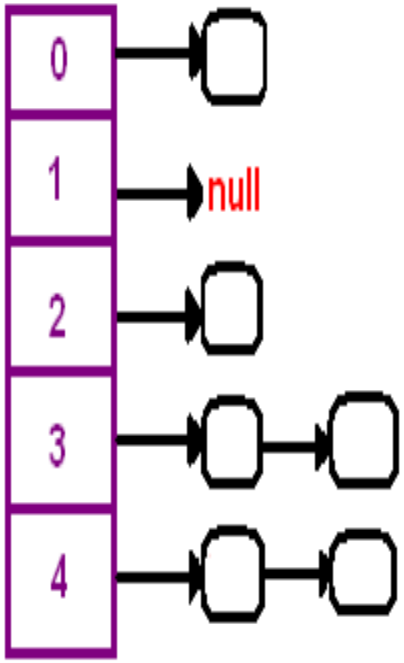
- ممکن است متد hashCode مقدار منفی برگرداند.
- در مواقعی که رشته بزرگ باشد، کدهش آن ممکن است در فضای ۳۲ بیتی CPU جا نشود. در نتیجه به خاطر سرریزی ممکن است مقدار کد هش، منفی گردد!

تصادمها (collisions)

- هنگامی که اشیا را در یک جدول هش قرار می‌دهیم، این احتمال وجود دارد که اشیای مختلف دارای کد هش یکسانی باشند.
- به این اتفاق collision، گفته می‌شود.
- مثالی از collision را ببینید که دو رشته مختلف"Aa" و "BB"کلید یکسانی دارند:

- "Aa" = 'A' * 31 + 'a' = 2112
  "BB" = 'B' * 31 + 'B' = 2112

- یک راه‌حل در چنین مواقعی استفاده از جدول هش زنجیره‌ای می‌باشد: *separate chaining resolution*
- کافیست اشیایی با کد هش یکسان را در یک اندیس جدول اما به صورت زنجیره‌ای قرار دهیم.
- بدیهی است که در این حالت، زمان جستجو اندکی بیشتر خواهد شد!



Perfect hash function

- تابع هش بی عیب، تابعی است که در آن collision وجود نداشته باشد. به جدول هش مربوطه، جدول هش بی عیب گفته می‌شود.
- در این مثال، یک جدول هش بی عیب به طول ۱۱ نشان داده شده است:

element	hash(element)
"beer"	5
"afterlife"	9
"wisdom"	4
"politics"	10
"schools"	1
"fear"	3

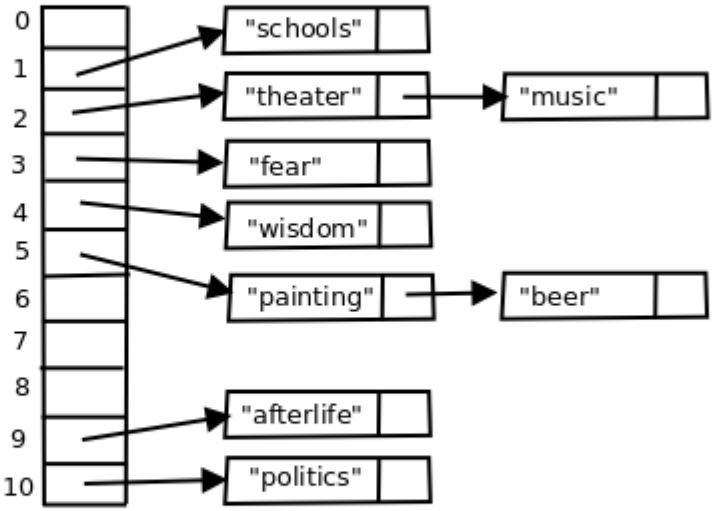
0	
1	"schools"
2	
3	"fear"
4	"wisdom"
5	"beer"
6	
7	
8	
9	"afterlife"
10	"politics"

- با اضافه شدن عنصر زیر، تصادم به وقوع می‌پیوندد:

element	hash(element)
"painting"	5

یک جدول هش حاوی تصادم

```
"music"
    hash code: 104263205
    array index: 2
"beer"
    hash code: 3019824
    array index: 5
"afterlife"
    hash code: 1019963096
    array index: 9
"wisdom"
    hash code: -787603007
    array index: 4
"politics"
    hash code: 547400545
    array index: 10
"theater"
    hash code: -1350043631
    array index: 2
"schools"
    hash code: 1917457279
    array index: 1
"painting"
    hash code: 925981380
    array index: 5
"fear"
    hash code: 3138864
    array index: 3
```



- مشخص است که در این ساختار، ترتیب عناصر وارد شده به جدول جایی ثبت نمی‌شود. در نتیجه ترتیب عناصر بازیابی شده، با ترتیب ورود آنها به جدول لزوماً یکی نخواهد بود.

Set: HashSet

```
import java.util.*;

public class MyHashSet {

    public static void main(String args[ ]) {

          HashSet hash = new HashSet( );

          hash.add("a");
          hash.add("b");
          hash.add("c");
          hash.add("d");

          Iterator iterator = hash.iterator( );

          while(iterator.hasNext( )) {
             System.out.println(iterator.next( ));
          }
    }
}
```

```
d
a
c
b
```

HashSet چگونه کار می کند؟

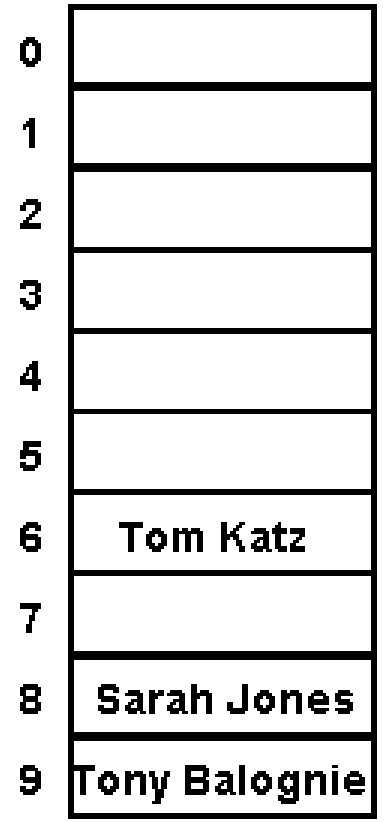
- هر شیئی یک عدد منحصربفرد دارد که به آن *hash code* گفته می‌شود.

  public int hashCode() in class Object

  - `HashSet` عناصرخود را در آرایه‌ای مثل `a` ذخیره می‌کند، به‌گونه‌ای که عنصر مربوطه مثلا `o` در اندیس زیر قرار گیرد:

    o.hashCode() % array.length

  - هر عنصر set باید دقیقا در یک اندیس آرایه قرار بگیرد.
  - برای جستجوی عنصر مربوطه نیاز به بررسی کل آرایه نیست؛ بلکه مستقیماً به اندیس مربوطه مراجعه می‌کنیم:
    - "Tom Katz".hashCode() % 10 == 6
    - "Sarah Jones".hashCode() % 10 == 8
    - "Tony Balognie".hashCode() % 10 == 9

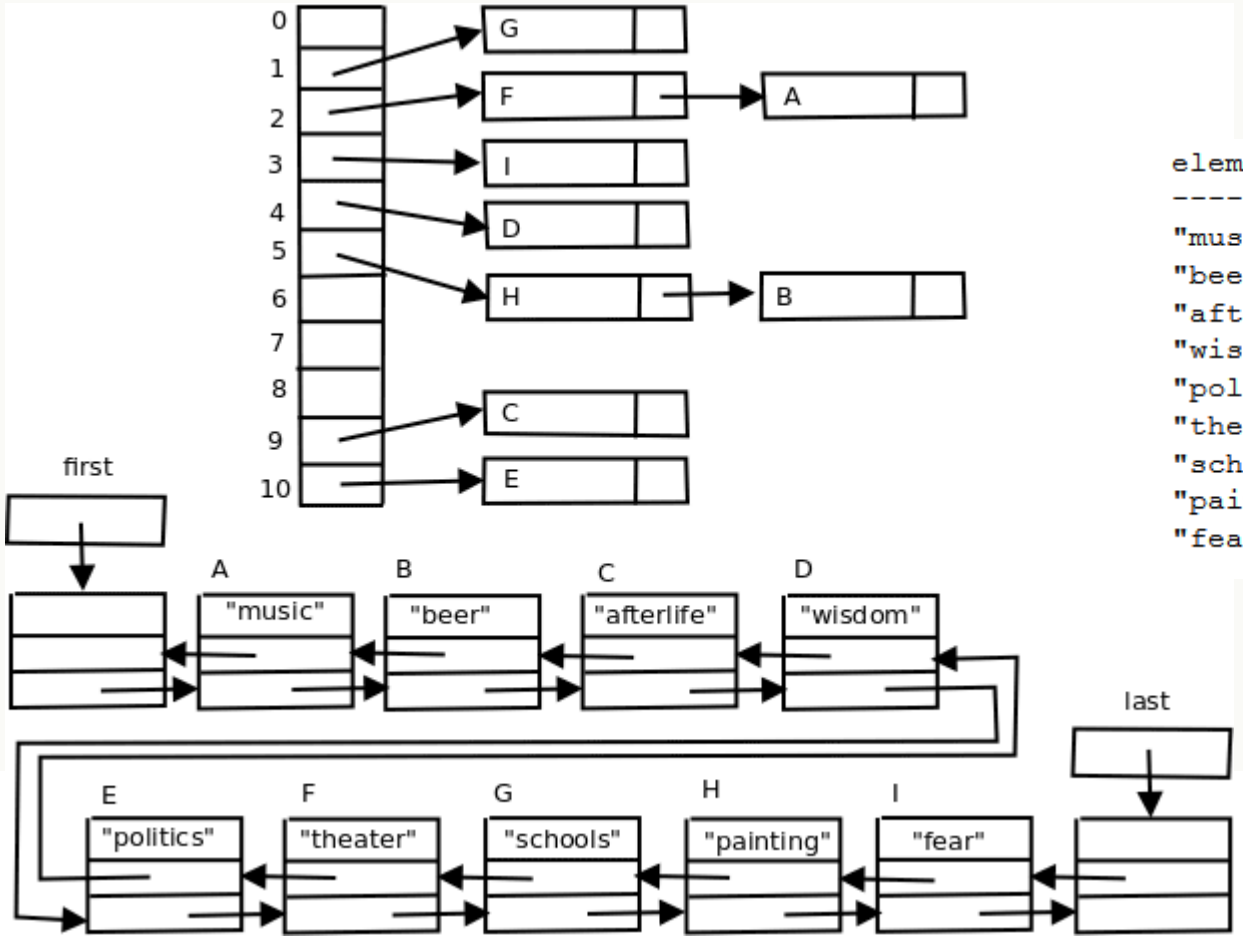


تست عضویت در HashSet

- برای اینکه جاوا بفهمد آیا HashSet حاوی شیئ مربوطه است:
  - ابتدا hashCode شیئ مربوطه محاسبه می‌شود.
  - در اندیس مربوطه در آرایه داخلی HashSet رجوع می‌شود.
  - شیئی داده شده با شیئ موجود در آن اندیس آرایه با متد equals مقایسه می‌شود، اگر باهم برابر بودند، مقدار true برگردانده می‌شود.
  - در نتیجه می‌گوییم شیئ مورد نظر در set قرار دارد، اگر هر دو شرط زیر برقرار باشند:
    - شیئی مربوطه دارای hash code برابر با شیئی موجود در آن خانه آرایه باشد.
    - و متد equals مقدار true را برگرداند.

Set :LinkedHashSet

- یک شیئ LinkedHashSet ترتیب عناصر وارد شده را نگهداری کرده و عملیات جستجویی آن، پیچیدگی زمانی ثابت (Constant time speed) دارد.
- برای رسیدن به این هدف، علاوه بر در اختیار داشتن جدول هش، یک لیست پیوندی دوطرفه (doubly) برای نگهداری عناصرش در اختیار دارد.

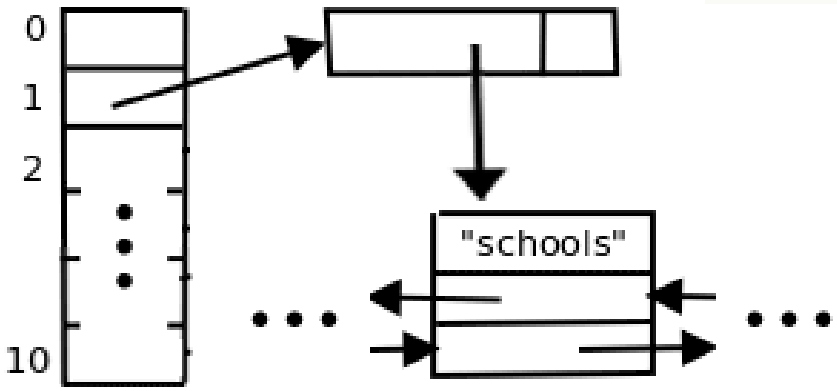


element	index
"music"	2
"beer"	5
"afterlife"	9
"wisdom"	4
"politics"	10
"theater"	2
"schools"	1
"painting"	5
"fear"	3

Set :LinkedHashSet

```
private class ListNode<E> {
  E data;
  ListNode<E> next;
  ListNode<E> prev;
  ListNode(E data, ListNode<E> next, ListNode<E> prev) {
    this.data = data;
    this.next = next;
    this.prev = prev;
  }
}

private static class Node<E> {
  ListNode<E> ptr;
  Node<E> next;
  Node(ListNode ptr, Node<E> next) {
    this.ptr = ptr;
    this.next = next;
  }
}

private ListNode<E> first, last;
```

- کلاس ListNode عناصر لیست پیوندی دوطرفه (doubly) را تشکیل می‌دهد که برای نگهداری عناصر اصلی Set به کار می‌رود.
- جدول هش ساده در این ساختار، کلاس Node اینگونه تغییر می کند که هر فیلد ptr آن ریموت کنترلی به ListNode است.
- حروف A, B,C... بیانگر اشاره‌گرهایی درون عناصر Node به عنصر ListNode مربوطه هستند.
- برای مثال نحوه دسترسی به "schools"را در شکل می‌بینید:



Set :LinkedHashSet

```
import java.util.LinkedHashSet;

public class MyLinkedHashSet {

   public static void main(String args[ ]) {

         LinkedHashSet lhs = new LinkedHashSet();

         lhs.add(new String("One"));
         lhs.add(new String("Two"));
         lhs.add(new String("Three"));

         Object array[] = lhs.toArray( );

         for(int x=0; x<3; x++) {
            System.out.println(array[x]);
         }
   }
}
```

```
One
Two
Three
```

LinkedHashSet

- LinkedHashSet یک لیست پیوندی از عناصر مجموعه را به ترتیبی که به لیست اضافه شده‌اند، نگهداری می‌کند.
- در هنگام پیمایش این مجموعه توسط یک iterator عناصر به ترتیب وارد شدنشان به مجموعه، برگردانده خواهند شد.
- این کلاس چهار سازنده مهم دارد که اولی یک مجموعه هَش پیش‌فرض می‌سازد:

- LinkedHashSet( )

- سازنده زیر یک مجموعه هش را با عناصر c می‌سازد:

- LinkedHashSet(Collection c)

- سازنده زیر اندازه اولیه (ظرفیت–تعداد سطرهای جدول هش) را مشخص می‌کند که این ظرفیت با افزایش تعداد عناصر بیشتر خواهد شد:

- LinkedHashSet(int capacity)

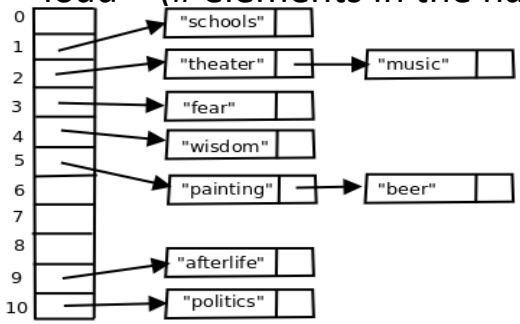
- سازنده زیر، هم ظرفیت جدول را مشخص می‌کند، هم ظرفیت بار را (fill ratio- load capacity)

- LinkedHashSet(int capacity, float fillRatio)

- ظرفیت بار که پیش فرض آن 0.75 است می گوید هرگاه تعداد عناصر وارد شده به جدول هش به ¾ ظرفیت جدول رسید، ظرفیت دوبرابر شده و rehashing صورت گیرد.

- load = (# elements in the hash table) / capacity

- در مثال زیر اندازه ظرفیت بار 0.82 = 9/11 است.



Rehash

- بزرگ شدن ظرفیت بار سبب کاهش کارایی جستجو در جدول هش می شود.
- هرگاه بار یک جدول بخواهد بیش از فاکتور بار شود، به عملیات rehash نیاز خواهیم داشت.
- این عملیات مشابه اضافه کردن اندازه ArrayList است: استفاده از یک آرایه بزرگتر و کپی مقادیر آرایه کوچکتر به آن
- در جدول هش، عملیات rehashing به این صورت است که
  - ابتدا یک ظرفیت capacity ≥ twice برابر یا بزرگتر از دوبرابر جدول قبلی انتخاب می‌شود.
  - ظرفیت اغلب یک عدد اول می‌باشد و دوبرابر کردن آن به معنای یافتن عدد اولی است که بزرگتر از دوبرابر ظرفیت قبلی باشد:2*capacity
  - آرایه‌ای جدید با ظرفیت مشخص‌شده تولید می‌شود.
  - جدول هش قبلی پیمایش می‌شود: برای هر عنصر داده‌ای، مقدار اندیس جدید (کد هش همان است، اما اندیس تغییر خواهد کرد) محاسبه می‌شود و عنصر مربوطه به جدول جدید منتقل می‌شود.

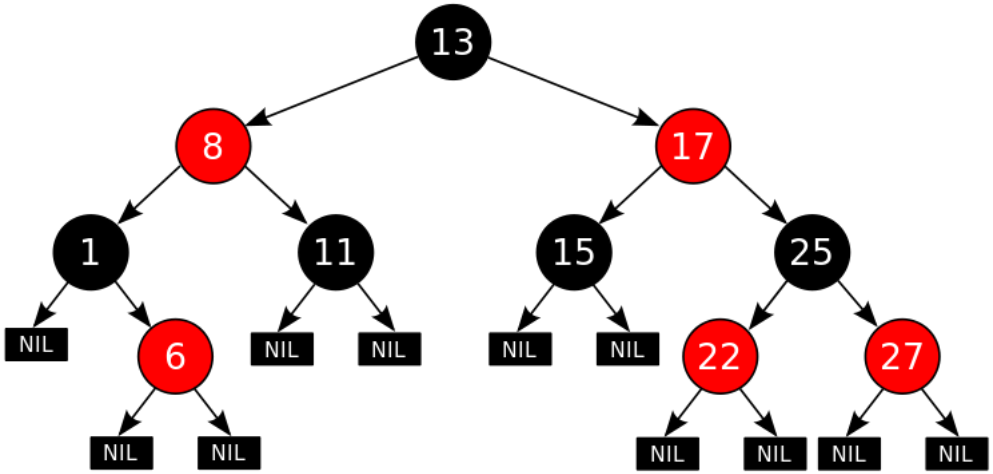
Set Implementations:TreeSet

```
import java.util.TreeSet;
import java.util.Iterator;

public class MyTreeSet {

   public static void main(String args[ ]) {

         TreeSet tree = new TreeSet();

         tree.add("Jody");
         tree.add("Remiel");
         tree.add("Reggie");
         tree.add("Philippe");

         Iterator iterator = tree.iterator( );

         while(iterator.hasNext( )) {
            System.out.println(iterator.next( ).toString( ));
         }
   }
}
```

```
Jody
Philippe
Reggie
Remiel
```

TreeSet

- TreeSet با استفاده از ساختار درختی (درخت سرخ–سیاه Red-Black tree) پیاده‌سازی شده است.
- عناصر در این مجموعه مرتب‌سازی شده‌اند، اما عملیات add، remove و contains دارای پیچیدگی زمانی O(log (n)) هستند.
- داده‌ها به ترتیب صعودی در این ساختار، ذخیره‌سازی می‌شوند.
- برای ذخیره‌سازی داده‌های مرتب با حجم بالا، بهترین گزینه می‌باشد.
- دارای متدهایی برای مجموعه‌های مرتب نظیر first()، last()، headSet()، tailSet() و غیره می‌باشد.



مثال TreeSet

```
TreeSet<Integer> tree = new TreeSet<Integer>();
tree.add(12);
tree.add(63);
tree.add(34);
tree.add(45);

Iterator<Integer> iterator = tree.iterator();
System.out.print("Tree set data: ");
while (iterator.hasNext()) {
    System.out.print(iterator.next() + " ");
}
```

```
Tree set data: 12 34 45 63
```

مثال TreeSet

- یک کلاس Dog تعریف می‌کنیم:

```
class Dog {
        int size;

        public Dog(int s) {
                size = s;
        }

        public String toString() {
                return size + "";
        }
}
```

- حال اشیایی از Dog را به TreeSet اضافه می‌کنیم:

```
import java.util.Iterator;
import java.util.TreeSet;

public class TestTreeSet {
        public static void main(String[] args) {
                TreeSet<Dog> dset = new TreeSet<Dog>();
                dset.add(new Dog(2));
                dset.add(new Dog(1));
                dset.add(new Dog(3));

                Iterator<Dog> iterator = dset.iterator();

                while (iterator.hasNext()) {
                        System.out.print(iterator.next() + " ");
                }
        }
}
```

مثال TreeSet

- کامپایل موفقیت‌آمیز است، اما با استثنای زیر روبرو می‌شویم؛ زیرا

```
Exception in thread "main" java.lang.ClassCastException: collection.Dog cannot be cast to java.lang.Comparable
        at java.util.TreeMap.put(Unknown Source)
        at java.util.TreeSet.add(Unknown Source)
        at collection.TestTreeSet.main(TestTreeSet.java:22)
```

- چون TreeSet مرتب‌سازی شده است، شیئ Dog باید متد compareTo() در java.lang.Comparable را مشابه زیر پیاده‌سازی کند:

```
class Dog implements Comparable<Dog>{
        int size;

        public Dog(int s) {
                size = s;
        }

        public String toString() {
                return size + "";
        }

        @Override
        public int compareTo(Dog o) {
                return size - o.size;
        }
}
```

```
1 2 3
```

همان مثال روی HashSet

```
HashSet<Dog> dset = new HashSet<Dog>();
dset.add(new Dog(2));
dset.add(new Dog(1));
dset.add(new Dog(3));
dset.add(new Dog(5));
dset.add(new Dog(4));
Iterator<Dog> iterator = dset.iterator();
while (iterator.hasNext()) {
        System.out.print(iterator.next() + " ");
}
```

```
5 3 2 1 4
```

- ترتیب ورود عناصر حفظ نمی‌شود!

LinkedHashSet همان مثال روی

```
LinkedHashSet<Dog> dset = new LinkedHashSet<Dog>();
dset.add(new Dog(2));
dset.add(new Dog(1));
dset.add(new Dog(3));
dset.add(new Dog(5));
dset.add(new Dog(4));
Iterator<Dog> iterator = dset.iterator();
while (iterator.hasNext()) {
        System.out.print(iterator.next() + " ");
}
```

- ترتیب ورود عناصر حفظ می‌شود!

```
2 1 3 5 4
```

متد compareTo() در کلاس Object

- این متد یک شیئ Numebr را که فراخواننده متد است با آرگومانش که همنوع خودش است، مقایسه می‌کند.
- به این روش می‌توان Byte، Long،Integer و غیره را مقایسه نمود.
- اگر شیئ فراخواننده از جنس آرگومان متد نباشد، امکان مقایسه وجود ندارد.
- قاعده نحوی:

```
public int compareTo( NumberSubClass referenceName )
```

- مقادیر برگشتی متد:

- If the Integer is equal to the argument then 0 is returned.
- If the Integer is less than the argument then -1 is returned.
- If the Integer is greater than the argument then 1 is returned.

```
public class Test{

   public static void main(String args[]){
      Integer x = 5;
      System.out.println(x.compareTo(3));
      System.out.println(x.compareTo(5));
      System.out.println(x.compareTo(8));
     }
}
```

```
1
0
-1
```

مقایسه LinkedHashSet با TreeSet با HashSet

- HashSet با استفاده از یک جدول هش پیاده‌سازی می‌شود.
  - عناصر ترتیب خاصی ندارند.
  - عملیات add، remove، و contains دارای پیچیدگی زمانی O(1) می‌باشد.

- TreeSet با استفاده از ساختار درختی پیاده‌سازی می‌شود.
  - عملیات add، remove، و contains دارای پیچیدگی زمانی O(log (n)) می‌باشد.
  - دارای متدهایی جهت کار روی مجموعه مرتب‌شده است؛ مانند first()، last()، headSet()، tailSet()، غیره

- LinkedHashSet ساختاری مابین HashSet و TreeSet دارد. با استفاده از هش به همراه یک لیست پیوندی پیاده‌سازی شده است که به جدول هش متصل می‌شود.
- بنابراین قادر به حفظ ترتیب ورود عناصر می‌باشد.
- پیچیدگی زمانی آن O(1) است.

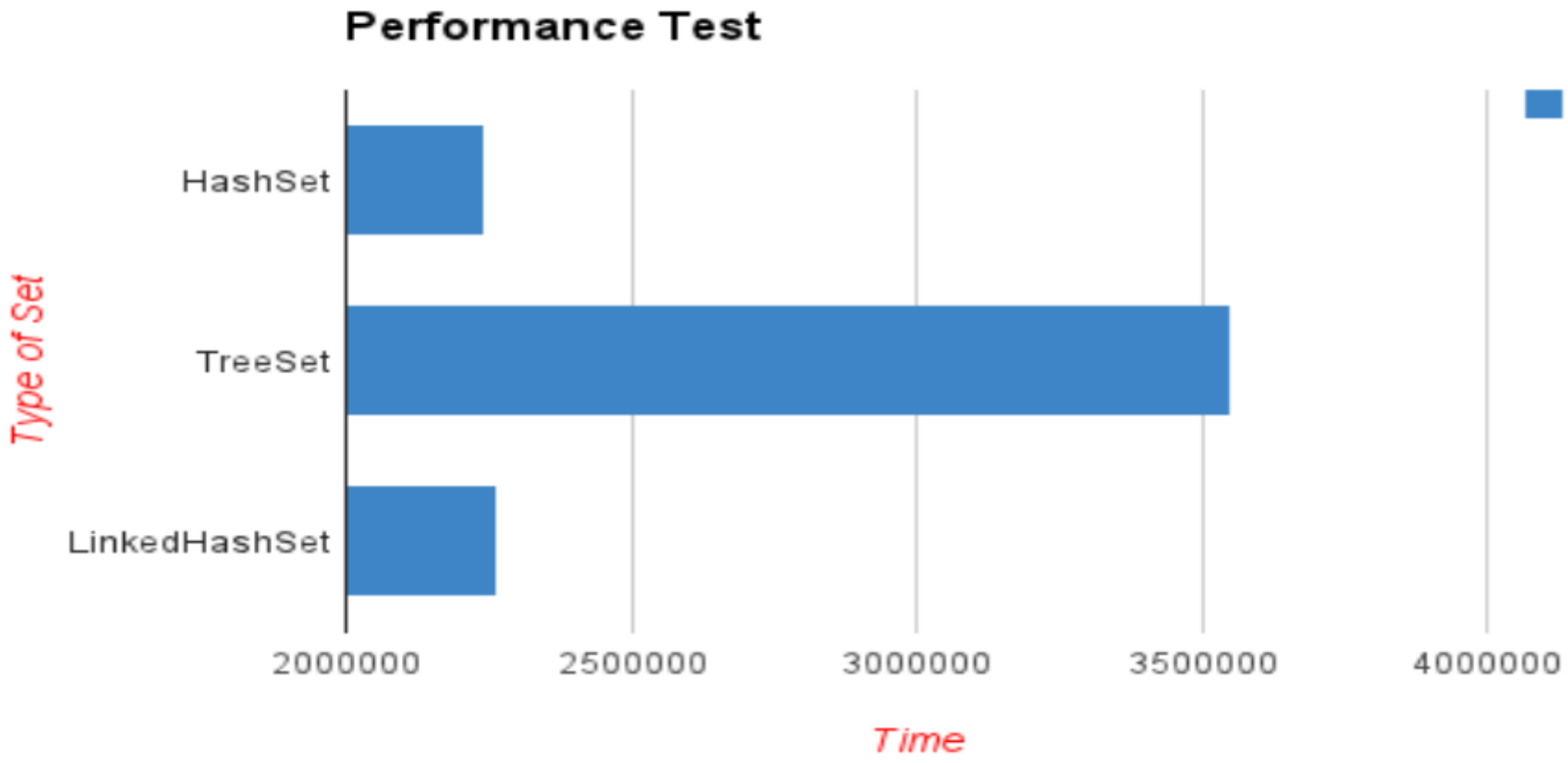
آزمون کارایی

```
public static void main(String[] args) {

        Random r = new Random();

        HashSet<Dog> hashSet = new HashSet<Dog>();
        TreeSet<Dog> treeSet = new TreeSet<Dog>();
        LinkedHashSet<Dog> linkedSet = new LinkedHashSe

        // start time
        long startTime = System.nanoTime();

        for (int i = 0; i < 1000; i++) {
                int x = r.nextInt(1000 - 10) + 10;
                hashSet.add(new Dog(x));
        }
        // end time
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println("HashSet: " + duration);

        // start time
        startTime = System.nanoTime();
        for (int i = 0; i < 1000; i++) {
                int x = r.nextInt(1000 - 10) + 10;
                treeSet.add(new Dog(x));
        }
        // end time
        endTime = System.nanoTime();
        duration = endTime - startTime;
        System.out.println("TreeSet: " + duration);
```

```
        // start time
        startTime = System.nanoTime();
        for (int i = 0; i < 1000; i++) {
                int x = r.nextInt(1000 - 10) + 10;
                linkedSet.add(new Dog(x));
        }
        // end time
        endTime = System.nanoTime();
        duration = endTime - startTime;
        System.out.println("LinkedHashSet: " + duration);

}
```

```
HashSet: 2244768
TreeSet: 3549314
LinkedHashSet: 2263320
```

آزمون کارایی



- این آزمون دقیق نیست؛ اما نشان می دهد که TreeSet چون باید مرتب‌سازی را در نظر داشته باشد، کندتر از دو کلاس دیگر عمل می‌کند.

تفاوت میان Set و List

1) List یک collection دارای ترتیب می‌باشد که ترتیب ورود عناصررا حفظ می‌کند.
2) Set یک collection بدون ترتیب است و ترتیب را حفظ نمی‌کند.
   - LinkedHashSet یکی از معدود پیاده‌سازیهای Set است که ترتیب ورود عناصر را حفظ می‌کند.
3) List امکان نگهداری عناصر تکراری را به برنامه‌نویس می‌دهد، اما در Set امکان تکرار عناصر وجود ندارد.
   - همه عناصر یک Set باید منحصربفرد باشند و در صورت تلاش برنامه‌نویس در قراردادن عنصر تکرای به مجموعه، عنصر جدید با عنصر قبلی جایگزین خواهد شد.

تفاوت میان Set و List

- پیاده‌سازیهای List: ArrayList، LinkedList
- پیاده‌سازیهای Set: LinkedHashSet، HashSet و TreeSet

۴) لیست به شما امکان ذخیره‌سازی هر تعداد مقدار null را می‌دهد. اما Set حداکثر یک مقدار null نگهداری می‌کند.

۵) ListIterator می‌تواند برای پیمایش لیست در دو جهت به کار رود؛ اما برای پیمایش مجموعه نمی‌توان از آن استفاده نمود. برای مجموعه‌ها از Iterator استفاده می‌کنیم.

چه موقع از set و چه موقع از List استفاده کنیم؟

- انتخاب ساختار ذخیره‌سازی مناسب به کاربرد شما بستگی دارد:
- اگر بخواهید فقط عناصر غیرتکراری را نگهداری کنید، Set بهترین گزینه است.
- اگر بخواهید ترتیب عناصر وارد شده را بدون توجه به تکراری بودن عنصر نگهداری کنید، List گزینه مناسب است.
- اگر هم ترتیب ورود عناصر و هم غیرتکراری بودن عناصر برایتان مهم است، از LinkedHashSet استفاده کنید.
- اگر می‌خواهید عناصر غیرتکراری اما مرتب‌شده باشند، از TreeSet استفاده کنید.

کاربردهای متفاوت مجموعه و لیست

```java
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;

public class ListExample {

 public static void main(String[] args) {
   List<String> al = new ArrayList<String>();
   al.add("Chaitanya");
   al.add("Rahul");
   al.add("Ajeet");
   System.out.println("ArrayList Elements: ");
   System.out.print(al);

   List<String> ll = new LinkedList<String>();
   ll.add("Kevin");
   ll.add("Peter");
   ll.add("Kate");
   System.out.println("\nLinkedList Elements: ");
   System.out.print(ll);
 }
}
```

```
ArrayList Elements:
[Chaitanya, Rahul, Ajeet]
LinkedList Elements:
[Kevin, Peter, Kate]
```

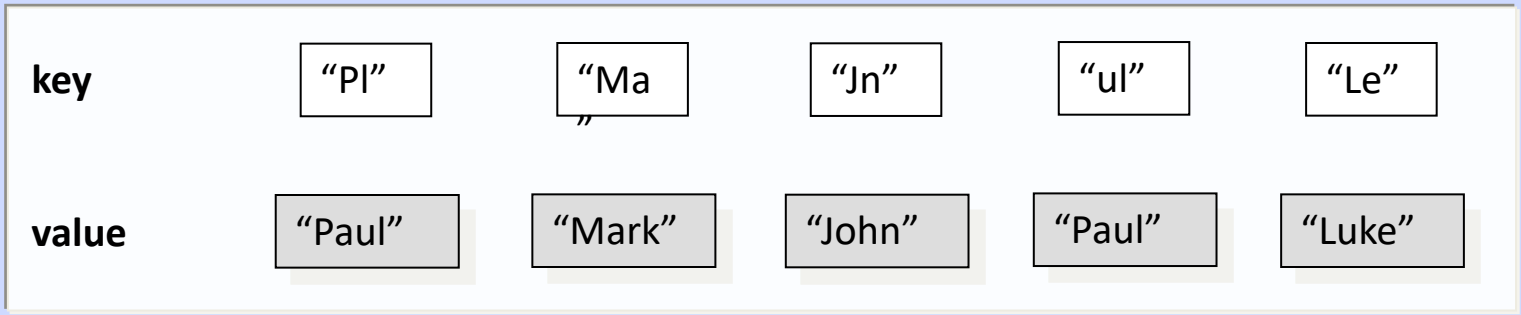
```java
import java.util.Set;
import java.util.HashSet;
import java.util.TreeSet;

public class SetExample {

  public static void main(String args[]) {
    int count[] = {11, 22, 33, 44, 55};
    Set<Integer> hset = new HashSet<Integer>();
    try{
      for(int i = 0; i<4; i++){
         hset.add(count[i]);
      }
      System.out.println(hset);

      TreeSet<Integer> treeset = new TreeSet<Integer>(hset);
      System.out.println("The sorted list is:");
      System.out.println(treeset);
    }
    catch(Exception e){
        e.printStackTrace();
    }
  }
}
```

```
[33, 22, 11, 44]
The sorted list is:
[11, 22, 33, 44]
```

نگاشت (Map) میان مجموعه ها (Set)

- برخی مواقع می‌خواهیم یک نگاشت یک‌به‌یک (mapping) بین عناصر یک set با عناصر set دیگر برقرار کنیم.
- برای مثال، نام افراد را به شماره تلفنشان نگاشت دهیم:
- "Marty Stepp" --> "253-692-4540"
- "Jenny" --> "253-867-5309"
- یک List این‌کار را برای ما انجام نمی‌دهد. تنها کاری که انجام می‌دهد نگاشت اعداد صحیح ۰ تا size-1 به اشیای موجود در لیست است.
- یک Set نیز اگرچه از تکرار عناصر جلوگیری کرده و زمان جستجوی بسیار سریعی دارد، برای نگاشت یک کلید به مقدار، مناسب نیست.
- اگر بخواهیم با دادن نام شخص، شماره تلفن او را در کوتاهترین زمان ممکن بیابیم، از چه روشی استفاده کنیم؟

Map

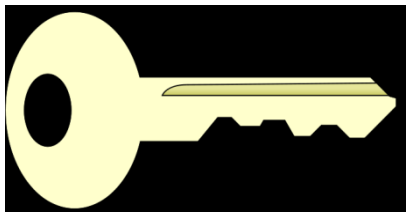


A Map cares about unique identifiers.

HashMap | **Hashtable** | **LinkedHashMap** | **TreeMap**

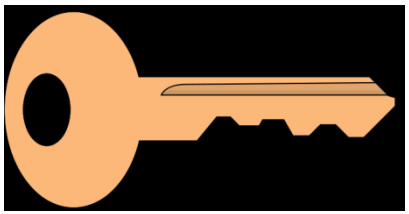
فرق List با Set با Map

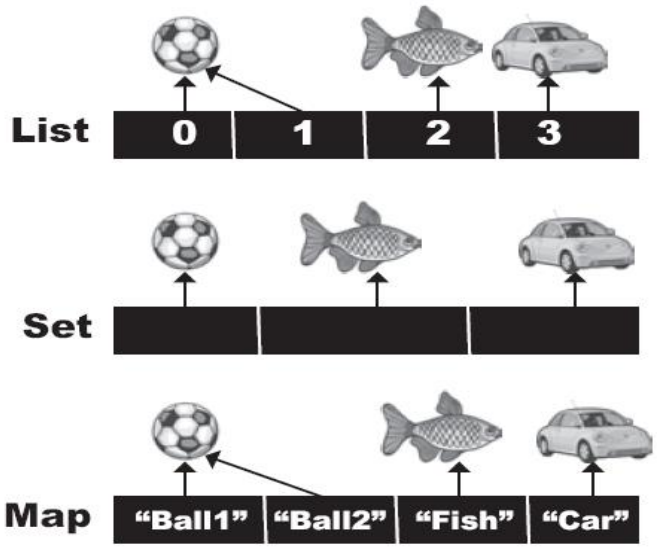
- هریک از ساختمان داده‌های معرفی‌شده در چارچوب collections برای جستجو/دسترسی خاصی طراحی شده‌اند:
  - با داشتن یک Set می‌پرسیم: "آیا آیتم X موجود است؟"
  - با داشتن یک List می‌گوییم: " آیتم به شماره X را بدهید؟"
  - با داشتن یک Map می‌پرسیم: "مقدار ذخیره‌شده برای کلید X را می‌خواهم."



Map چیست؟

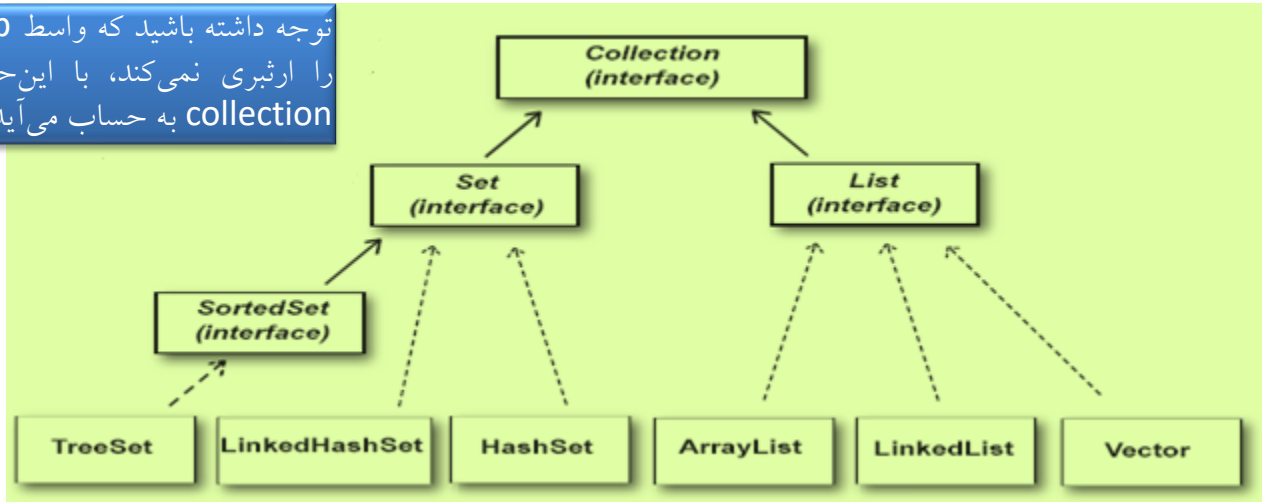
- **map**: **یک** collection بدون‌ترتیب است که مجموعه‌ای از اشیا (مقادیر مختلف) را به مجموعه‌ای از کلیدها مرتبط می‌کند، به شکلی که دسترسی به مقدار هر عنصر در زمانی بسیار اندک انجام شود.
- یک کلید به همراه مقدار آن یک جفت را تشکیل می‌دهند که در Map ذخیره می‌شوند.
- برای بازیابی یک مقدار باید کلید آن را داشته باشیم.
- هر کلید می‌تواند حداکثر یکبار ظاهر شود (تکرار کلیدها مجاز نیست).
- یک کلید به حداکثر یک عنصر نگاشت می‌شود. کلیدهای متفاوت ممکن است مقادیر یکسان داشته باشند.
- یک List در حقیقت Map ای است که کلیدهای آن مقادیر صحیح از ۰ تا size-1 (طول لیست) هستند.
  - عملیات اصلی:
    - **put**(key,value)
      "Map this key to that value."
    - **get**(key)
      "What value, if any, does this key map to?"
  - اسامی دیگر برای map ها:
    - hashes or hash tables
    - dictionaries
    - associative arrays
    - Table
    - search table
    - associative container

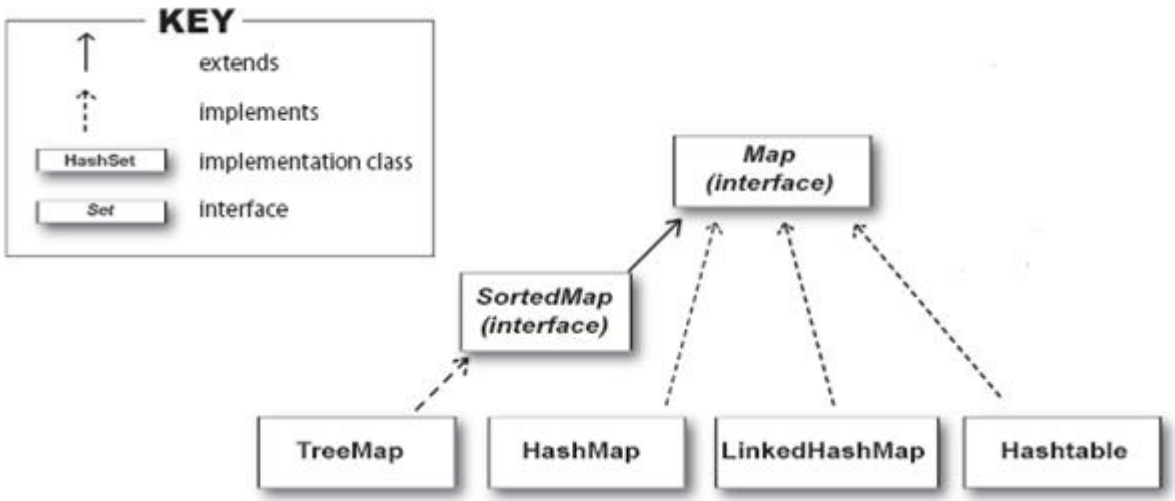




نگاهی دوباره به چارچوب collection

توجه داشته باشید که واسط Map درواقع واسط Collection را ارثبری نمی‌کند، با این‌حال Map بخشی از چارچوب collection به حساب می‌آید.





Map واسط

عملیات مهم

```
public interface Map {
    - Object put(Object key, Object value); //Associates the specified value with the specified key in map
    - Object get(Object key) //Returns the value to which this map maps the specified key.
    - Object remove(Object key);
    - boolean containsKey(Object key);      //Returns true if this map contains a mapping for the specified key.
    - boolean containsValue(Object value);  //  Returns true if this map maps one or more keys to the specified value.
    - int size();    //Returns the number of key-value mappings
    - Set<K> keySet()  //Returns a Set view of the keys contained in this map.
    - boolean isEmpty();  //Returns true if this map contains no key-value mappings.
```

عملیات برروی کل عناصر

```
void putAll(Map map);
```

- Map مشخص شده در آرگومان را در Map فراخواننده متد کپی می‌کند. در مثال زیر، محتوای oldMap در newMap کپی می‌شود.
- مثال:

```
newMap.putAll(oldMap);
```

```
void clear(); // Removes all mappings from this map
```

مثال:

```
oldMap.clear();
```

نکات مهم

- اگر map از قبل حاوی کلید مربوطه باشد، فراخوانی متد put(key, value) مقدار مرتبط با کلید موجود را با مقدار آرگومان value جدید، جایگزین می‌کند.
  - این‌کار به معنای آن است که جاوا آزمون برابری (equality) را برروی کلیدها انجام می‌دهد.
  - در کلاس HashMap لازم است برای کلیدهایی از اشیای غیرعددی و رشته‌ای، متدهای equals و hashCode را بازنویسی کنید.
  - در کلاس TreeMap لازم است متد equals را تعریف نموده و متد مقایسه در واسط Comparable را برای همه کلیدهایتان بازنویسی کنید.

- تفاوت میان Hashtable و HashMap:
- Hashtable سنکرون، اما HashMap آسنکرون است.
- HashMap امکان نگهداری مقدار و کلید null را می‌دهد.
- اما در Hashtable این کار مجاز نیست.

پیاده سازیهای Map در جاوا

- Map یک واسط (interface) است؛ در نتیجه نمی توانیم بگوییم:

```
new Map()
```

- دو پیاده سازی مهم ازMap وجود دارد:
  - java.util.HashMap برای بسیاری از اهداف مناسب است و سریع عمل می کند
  - TreeMap: تضمین می‌کند ترتیب عناصر در طول استفاده حفظ شود.

- روش مناسب ایجاد شیئ از HashMap

```
Map m = new HashMap();
```

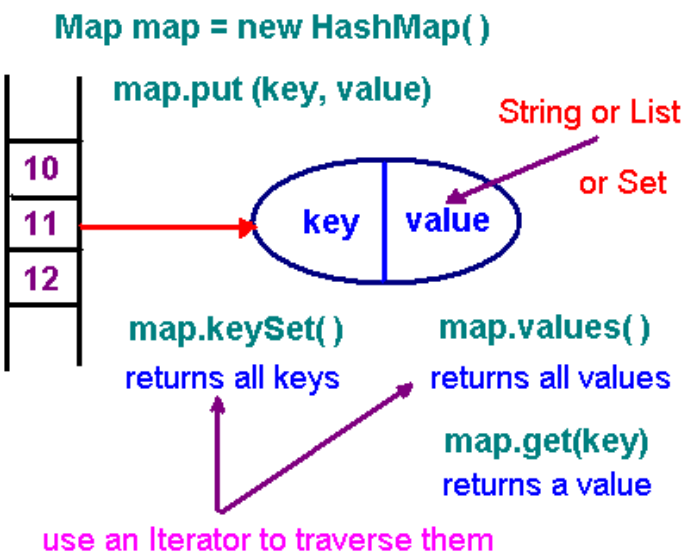
پیاده سازی HashMap

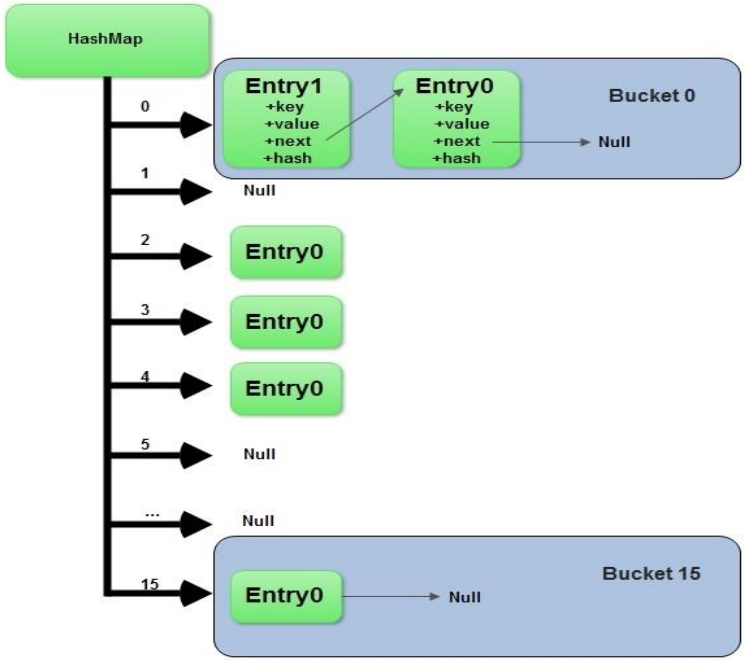
- HashMap از یک جدول هش به عنوان فضای ذخیره‌سازی داخلی خود استفاده می‌کند.
  - کلیدها بر مبنای کدهای هش شان و اندازه جدول هش، ذخیره سازی می‌شوند.
- روشهای ایجاد یک HashMap:

```
public HashMap()
public HashMap(int initialCapacity)
public HashMap(int initialCapacity, float loadFactor)
```

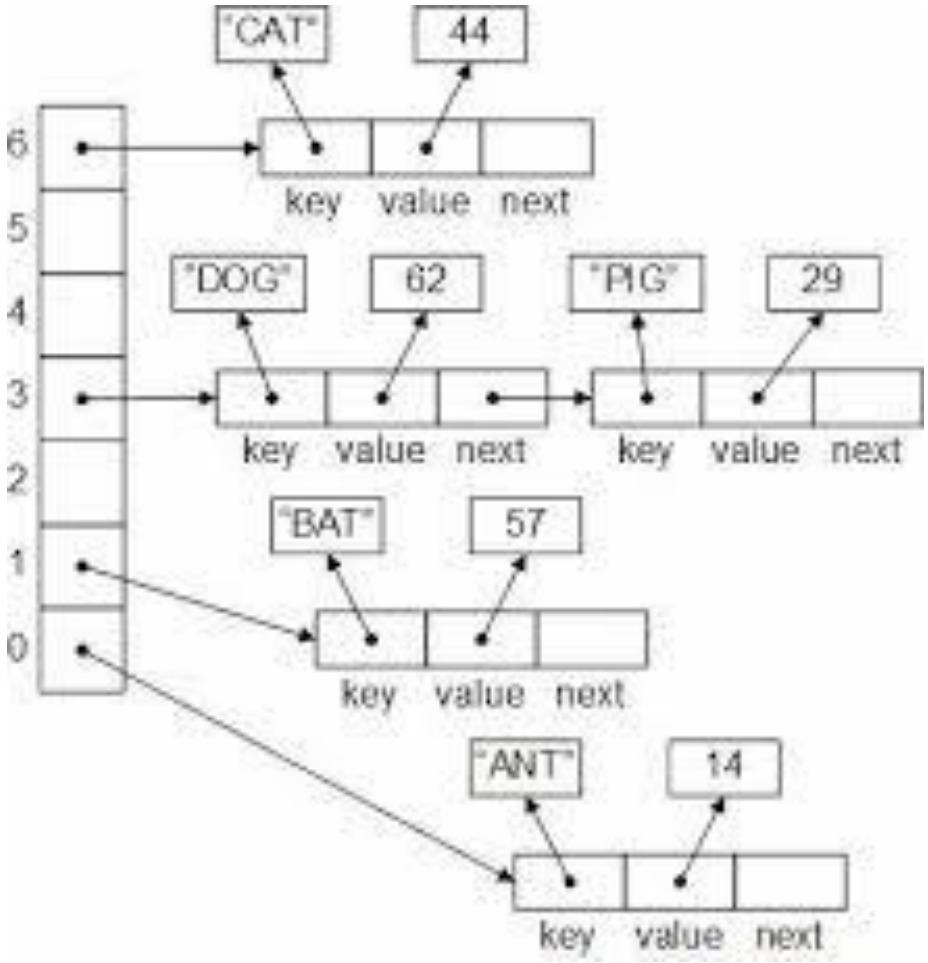
- سازنده زیر یک سازنده کپی استاندارد است که یک HashMap جدید را از map موجود می‌سازد:

```
public HashMap(Map map)
```

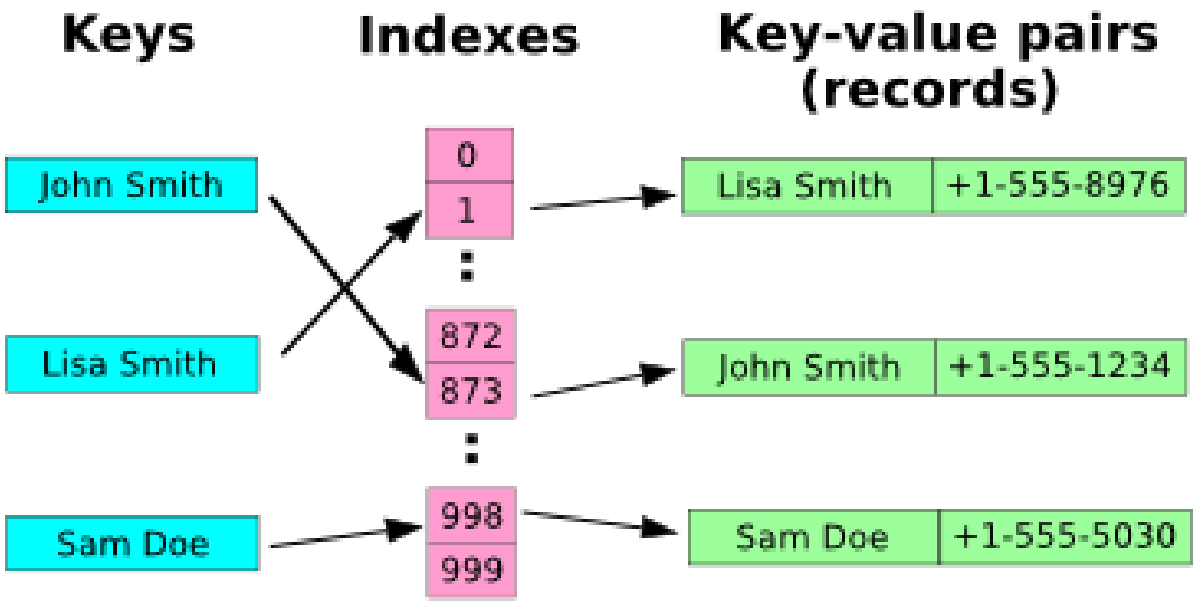




۱-HashMap تصویری از نحوه پیاده سازی

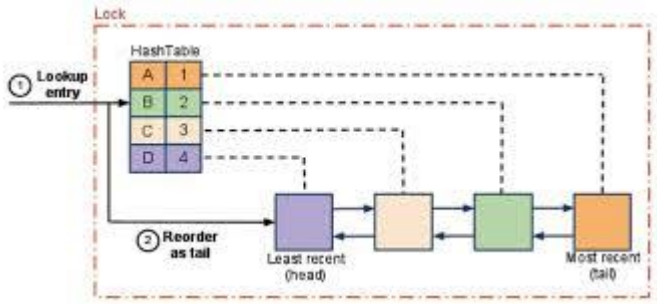


۲-HashMap تصویری از نحوه پیاده سازی



پیاده سازی LinkedHashMap و TreeMap

- HashMap مانند HashSet ترتیب ورود عناصر را حفظ نمی‌کند.
- اگر ترتیب ورود عناصر برایتان مهم است، از *LinkedHashMap* استفاده کنید که مانند *LinkeHashSet* پیاده‌سازی شده است.



- اگر می‌خواهید عناصر به صورت مرتب‌شده ذخیره و بازیابی شوند از TreeMap استفاده کنید که مانند TreeSet با استفاده از درخت سرخ–سیاه پیاده‌سازی شده است.
- این کلاس نیز مانند TreeSet متکی بر متد `compareTo` برروی کلیدها است.

Map views

- `Set<K> keySet()`
  - Returns a set view of the keys contained in this map.
- `Collection<V> values()`
  - Returns a collection view of the values contained in this map
  - Can't be a set—keys must be unique, but values may be repeated
- `Set<Map.Entry<K, V>> entrySet()`
  - Returns a set view of the mappings contained in this map.
- A view is *dynamic access* into the Map
  - If you change the Map, the view changes
  - If you change the view, the Map changes
- The Map interface does not provide any Iterators
  - However, there are iterators for the above Sets and Collections

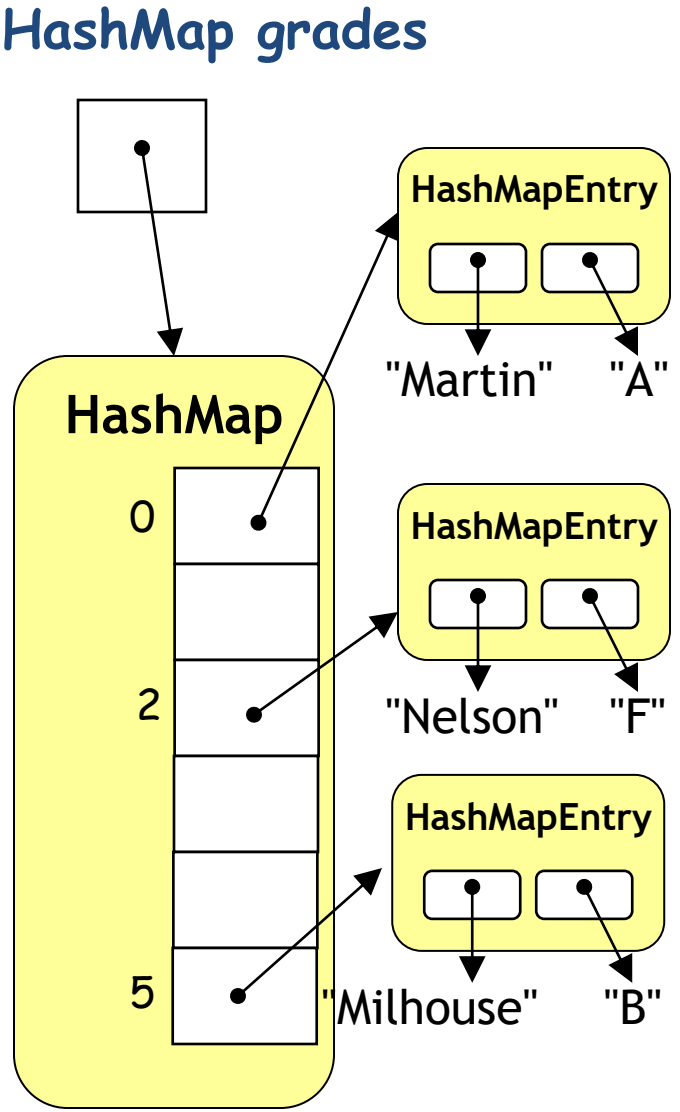
مثال از پیمایش Map

```
import java.util.*;

public class MapExample {

    public static void main(String[] args) {
        Map<String, String> fruit = new HashMap<String, String>();
        fruit.put("Apple", "red");
        fruit.put("Pear", "yellow");
        fruit.put("Plum", "purple");
        fruit.put("Cherry", "red");
        for (String key : fruit.keySet()) {
            System.out.println(key + ": " + fruit.get(key));
        }
    }
}
```

```
Plum: purple
Apple: red
Pear: yellow
Cherry: red
```

مثالی از HashMap

```
HashMap grades = new HashMap();
grades.put("Martin", "A");
grades.put("Nelson", "F");
grades.put("Milhouse", "B");

// What grade did they get?
System.out.println(
  grades.get("Nelson"));
System.out.println(
  grades.get("Martin"));

grades.put("Nelson", "W");
grades.remove("Martin");

System.out.println(
  grades.get("Nelson"));
System.out.println(
  grades.get("Martin"));
```



نگهداری تعداد رخدادهای کلمات در متن با HashMap

```
String[] data = new String("Nothing is as easy as it looks").split(" ");

HashMap<String, Integer> hm = new HashMap<String, Integer>();

for (String key : data)
{
        Integer freq = hm.get(key);
        if(freq == null) freq = 1; else freq ++;
        hm.put(key, freq);
}
System.out.println(hm);
```

```
{as=2, Nothing=1, it=1, easy=1, is=1, looks=1}.
```

پیمایش یک Map بدون استفاده از کلیدهای آن

```java
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;

public class Main {

  public static void main(String[] args) {
    HashMap<String, String> hMap = new HashMap<String, String>();
    hMap.put("1", "One");
    hMap.put("2", "Two");
    hMap.put("3", "Three");

    Collection c = hMap.values();
    Iterator itr = c.iterator();
    while (itr.hasNext()) {
      System.out.println(itr.next());
    }
  }
}
/*
Three
Two
One
*/
```

نمایش محتوای یک Map با متد toString()

```
import java.util.HashMap;
import java.util.Map;

public class MainClass {
  public static void main(String[] a) {
    Map map = new HashMap();
    map.put("key1", "value1");
    map.put("key2", "value2");
    map.put("key3", "value3");

    System.out.println(map);
  }
}
```

```
{key1=value1, key3=value3, key2=value2}
```

Map :Hashtable

- اگر کلید HashMap یک شیئی تعریف شده توسط برنامه‌نویس باشد، باید از قواعد equals() و hashCode() پیروی شود.

```
class Dog {
        String color;

        Dog(String c) {
                color = c;
        }
        public String toString(){
                return color + " dog";
        }
}

public class TestHashMap {
        public static void main(String[] args) {
                HashMap<Dog, Integer> hashMap = new HashMap<Dog, Integer>();
                Dog d1 = new Dog("red");
                Dog d2 = new Dog("black");
                Dog d3 = new Dog("white");
                Dog d4 = new Dog("white");

                hashMap.put(d1, 10);
                hashMap.put(d2, 15);
                hashMap.put(d3, 5);
                hashMap.put(d4, 20);

                //print size
                System.out.println(hashMap.size());

                //loop HashMap
                for (Entry<Dog, Integer> entry : hashMap.entrySet()) {
                        System.out.println(entry.getKey().toString() + " - " +
                        entry.getValue());
                }
        }
}
```

```
4
white dog - 5
black dog - 15
red dog - 10
white dog - 20
```

- این برنامه تعداد سگها به رنگهای مختلف را در یک Map قرار می‌دهد.
- "white dogs"به اشتباه دوبار اضافه شده است اما HashMap ایرادی نمی‌گیرد! چرا؟
- اما این مسئله سبب سردرگمی کاربر می‌شود، چون نمی‌فهمد در اصل چند سگ سفید وجود دارد!

Map :Hashtable

- اگر کلید HashMap یک شیئی تعریف شده توسط برنامه نویس باشد، باید equals() و hashCode() بازنویسی شوند.

```
class Dog {
        String color;

        Dog(String c) {
                color = c;
        }

        public boolean equals(Object o) {
                return ((Dog) o).color.equals(this.color);
        }

        public int hashCode() {
                return color.length();
        }

        public String toString(){
                return color + " dog";
        }
}
```

```
3
red dog - 10
white dog - 20
black dog - 15
```

```
public class TestHashMap {
        public static void main(String[] args) {
                HashMap<Dog, Integer> hashMap = new HashMap<Dog, Integer>();
                Dog d1 = new Dog("red");
                Dog d2 = new Dog("black");
                Dog d3 = new Dog("white");
                Dog d4 = new Dog("white");

                hashMap.put(d1, 10);
                hashMap.put(d2, 15);
                hashMap.put(d3, 5);
                hashMap.put(d4, 20);

                //print size
                System.out.println(hashMap.size());

                //loop HashMap
                for (Entry<Dog, Integer> entry : hashMap.entrySet()) {
                        System.out.println(entry.getKey().toString() + " - " +
                        entry.getValue());
                }
        }
}
```

- به طور پیش فرض، متدهای hashCode() و equals() در کلاس Object پیاده‌سازی شده‌اند.
- متد hashCode() مقادیر عددی متمایزی برای اشیای متفاوت تولید می‌کند.
- متد equals() هم به طور پیش‌فرض در صورتی true برمی‌گرداند که دو ریموت کنترل به یک شیئ اشاره کنند.
- درنتیجه در هنگام به‌کارگیری اشیای کاربر به عنوان کلید Map باید هم متد equals و هم hashcode بازنویسی شوند.

Map :LinkedHashMap

- مانند LinkedHashSet با استفاده از یک لیست پیوندی دوطرفه ترتیب ورود کلیدها را نگهداری می‌کند.

```
import java.util.LinkedHashMap;
import java.util.Set;
import java.util.Iterator;
import java.util.Map;
public class LinkedHashMapDemo {
    public static void main(String args[]) {
        // HashMap Declaration
        LinkedHashMap<Integer, String> lhmap =
                new LinkedHashMap<Integer, String>();

        //Adding elements to LinkedHashMap
        lhmap.put(22, "Abey");
        lhmap.put(33, "Dawn");
        lhmap.put(1, "Sherry");
        lhmap.put(2, "Karon");
        lhmap.put(100, "Jim");

        // Generating a Set of entries
        Set set = lhmap.entrySet();
        // Displaying elements of LinkedHashMap
        Iterator iterator = set.iterator();
        while(iterator.hasNext()) {
            Map.Entry me = (Map.Entry)iterator.next();
            System.out.print("Key is: "+ me.getKey() +
                    "& Value is: "+me.getValue()+"\n");
        }
    }
}
```

```
Key is: 22& Value is: Abey
Key is: 33& Value is: Dawn
Key is: 1& Value is: Sherry
Key is: 2& Value is: Karon
Key is: 100& Value is: Jim
```

- برای دسترسی و مقداردهی کلیدها و مقادیر، واسط Entry در داخل Map تعریف و پیاده‌سازی شده است:

```
public interface Entry { // Inner interface of Map
    Object getKey( );
    Object getValue( );
    Object setValue(Object value);
}
```

تفاوت Hashtable و HashMap:

- برخلاف Hashtable ، کلید و مقدار null در HashMap مجاز است.

```
import java.util.HashMap;
import java.util.Map;

public class MainClass {
  public static void main(String[] a) {

    Map map = new HashMap();
    map.put("key1", "value1");
    map.put("key2", "value2");
    map.put("key3", "value3");
    map.put(null, null);

    System.out.println(map);
  }
}
```

```
{key1=value1, key3=value3, null=null, key2=value2}
```

نکات مهم

- برای آنکه یک HashMap عملکرد صحیحی داشته باشد:
  - `equals` باید به درستی برروی کلیدها تعریف شود.
  - `hashCode` باید به درستی برروی کلیدها تعریف شود.
- در بیشتر مواقع از رشته‌ها (`Strings`) برای کلیدها استفاده می‌شود.
  - به طور کلی اشیای تغییرناپذیر () برای کلیدها توصیه می‌شوند.
  - اگر کلید شما از اشیای تغییرپذیر باشد و در طول برنامه مقدار آن شیئ عوض شود، نتایجی ناخواسته به بار خواهد آمد!
- اگر از اشیایی از کلاسهای جدید به عنوان کلید استفاده می‌کنید، `equals` و `hashCode` را بازنویسی کنید.