به نام پروردگار دانایی

# طراحی سیستم‌های شی‌گرا

## برنامه‌نویسی شی‌گرا

درس سوم: گروه‌بندی اشیا

سید کاوه احمدی

# مفاهیم این درس

- مجموعه‌ها
  - اشیایی که تعداد نامشخصی از اشیا دیگر را نگهداری می‌کنند.
  - می‌توانیم اشیایی را اضافه یا حذف کنیم.
  - مجموعه‌های با نوع مشخص
- حلقه‌ها (بسیار شبیه به C)
  - `for, while, do/while`
  - یک نوع حلقه جدید در جاوا ۵
- `iterators`
  - روش دیگری برای پیمایش در جاوا
- آرایه‌ها
  - بسیار شبیه به C

# نیاز به گروه‌بندی اشیا

- بسیاری از برنامه‌های کاربردی شامل یک مجموعه (collection) از اشیا هستند.
  - Personal organizers.
  - Library catalogs.
  - Student-record system.
- تعداد اقلامی که باید ذخیره شوند مشخص نیست.
  - اقلامی اضافه می‌شوند.
  - اقلامی حذف می‌شوند.

# یک دفترچه یادداشت شخصی

- می‌توان یادداشت‌هایی در سیستم ذخیره کرد.
- تعداد از پیش مشخص شده‌ای برای یادداشت‌ها مشخص نشده است.
- می‌توان تعداد یادداشت‌های ذخیره شده را نمایش داد.
- می‌توان یادداشت‌ها را نمایش داد.

# Demo

# کتابخانه کلاس (Class libraries)

- یک مجموعه از کلاس‌های سودمند.
- لازم نیست همه چیز را از اول بنویسیم.
  - قرار نیست چرخ را دوباره اختراع کنیم، از آن استفاده می‌کنیم.
- در جاوا این کتابخانه‌ها، *package* (بسته) خوانده می‌شود.
- گروه‌بندی اشیا یک نیاز مهم است:
  - `java.util` بسته‌ای است در جاوا که شامل کلاس‌هایی برای انجام این کار است.
  - باید برای استفاده به کلاس الحاق (`import`) شود.
  - الحاق باید پیش از تعریف کلاس باشد.

```
import java.util.ArrayList;

/**
 * ...
 */
public class Notebook
{
    // Storage for an arbitrary number of notes.
    private ArrayList<String> notes;

    /**
     * Perform any initialization required for the
     * notebook.
     */
    public Notebook()
    {
        notes = new ArrayList<String>();
    }

    ...
}
```

# مجموعه‌ها

```
private ArrayList<String> notes;
```

- مشخص می‌کنیم:
  - نوع مجموعه: `ArrayList`
  - نوع اشیایی که درون مجموعه خواهند بود: `<String>`
- می‌گوییم یک `ArrayList` از `String`ها

- برای استفاده باید بسته زیر را به کلاس الحاق کنیم:
  - `import java.util.ArrayList;`

```
public class Notebook
{
    private ArrayList<String> notes;

    public Notebook()
    {
        notes = new ArrayList<String>();
    }
    ...
}
```

اعلان

تعریف

## یا؟

```
public class Notebook
{
    private ArrayList<String> notes = new ArrayList<String>();

    ...
}
```

این کار را انجام نمی‌دهیم. ممکن است بخواهیم فیلد را به روش دیگری تعریف کنیم. بنابراین تعریف فیلد را تا زمان ایجاد شی به تعویق می‌اندازیم

# کلاس‌های وابسته به جنس (Generic classes)

- مجموعه‌ها به عنوان وابسته به جنس (*generic* types) یا پارامتردار (*parameterized*) شناخته می‌شوند.
  - ArrayList یک کلاس است که قابلیت پذیرش نوع‌های داده‌ای متفاوتی را دارد
    - چند ریختی پارامتری (Parametric polymorphism)
- نوع پارامتری شده مشخص می‌کند یک لیست از چه نوعی می‌خواهیم.
  - ArrayList<Person>
  - ArrayList<TicketMachine>
  - etc.

# ArrayList

- `ArrayList` عملیات مورد انتظار برای یک لیست را **پیاده‌سازی** می‌کند.
  - تجرید رفتار
- عملیاتی که روی یک `ArrayList` می‌توان اضافه کرد:
  - اضافه کردن المان جدید، حذف المان، دریافت یک المان
- از کجا عملیات مربوط به یک کتابخانه کلاس و نحوه استفاده از آنرا بشناسیم؟
  - پاسخ: `API` جاوا

# API

- بسته‌ها در واقع بخشی از API جاوا هستند.

API: **A**pplication **P**rogrammer's **I**nterface

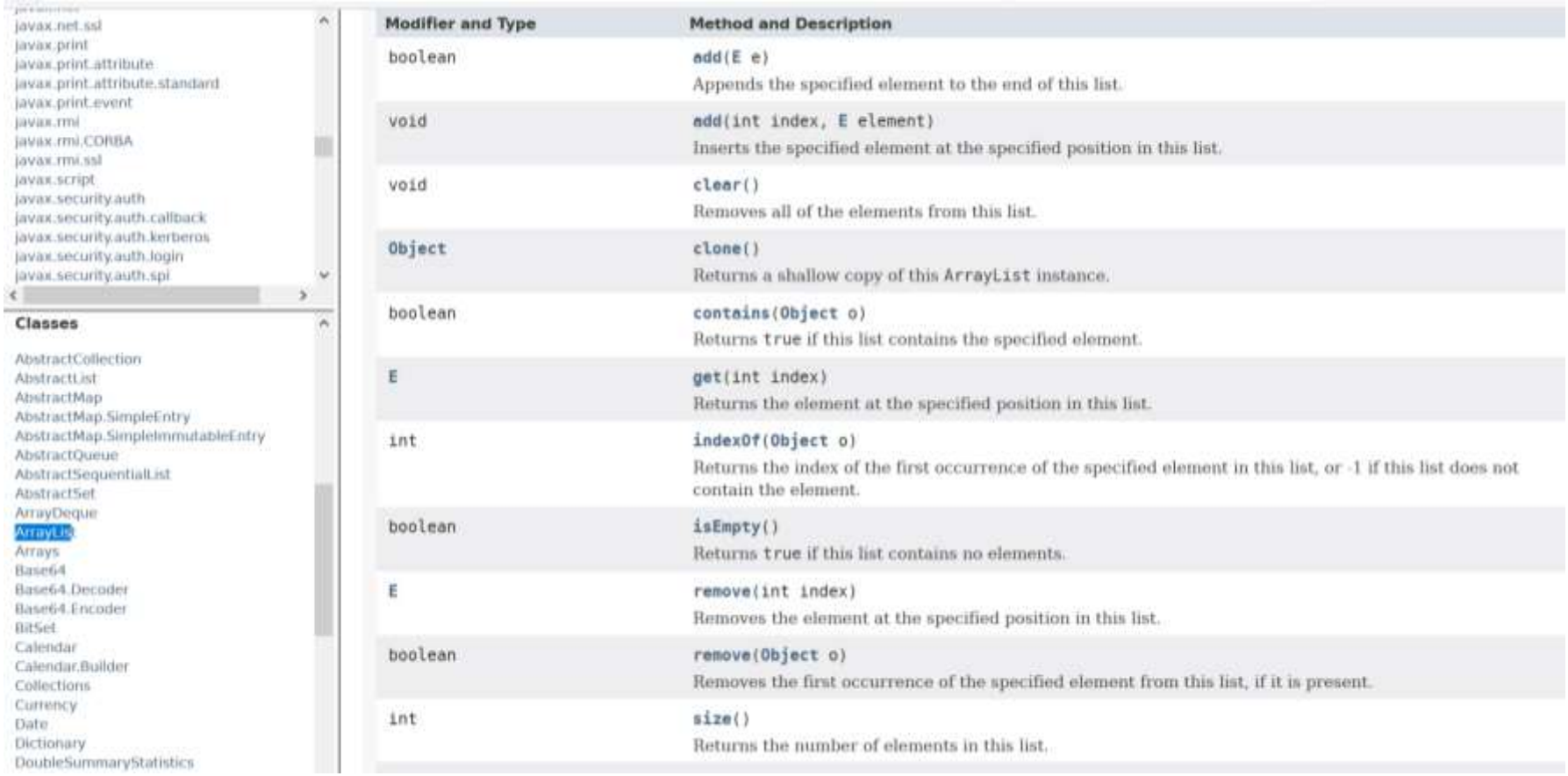
- رابط کاربردی برنامه‌نویسی
- بخشی از JDK
- API جاوا در واقع یک مجموعه بزرگ از کتابخانه‌ها برای انجام کارهای پایه‌ای همانند مجموعه‌ها (collections)، تکرار (looping)، نمایش GUI و... است.
- تمام کلاس‌های موجود در کلاس‌های بسته‌های API جاوا به زبان جاوا نوشته شده‌اند بنابراین می‌توانند توسط JVM اجرا شوند. اما درون JVM قرار ندارند.

# API

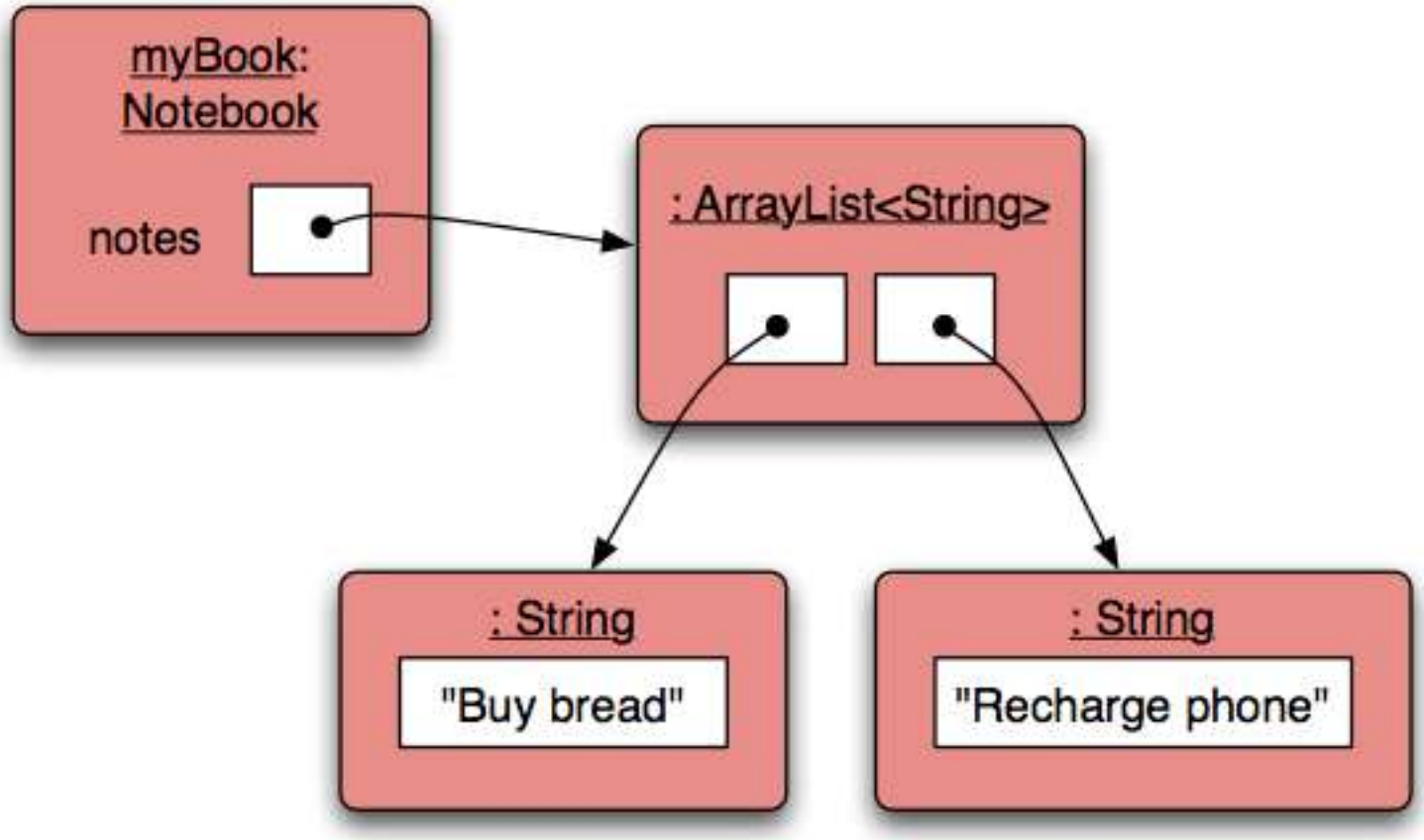
- برنامه‌ها (programs) در جاوا برنامه کاربردی (application) خوانده می‌شوند. شما برنامه‌نویس برنامه کاربردی هستید.

  - See API documents:
    https://docs.oracle.com/javase/8/docs/api/
    - در BlueJ از منوی help نیز قابل دسترس است.

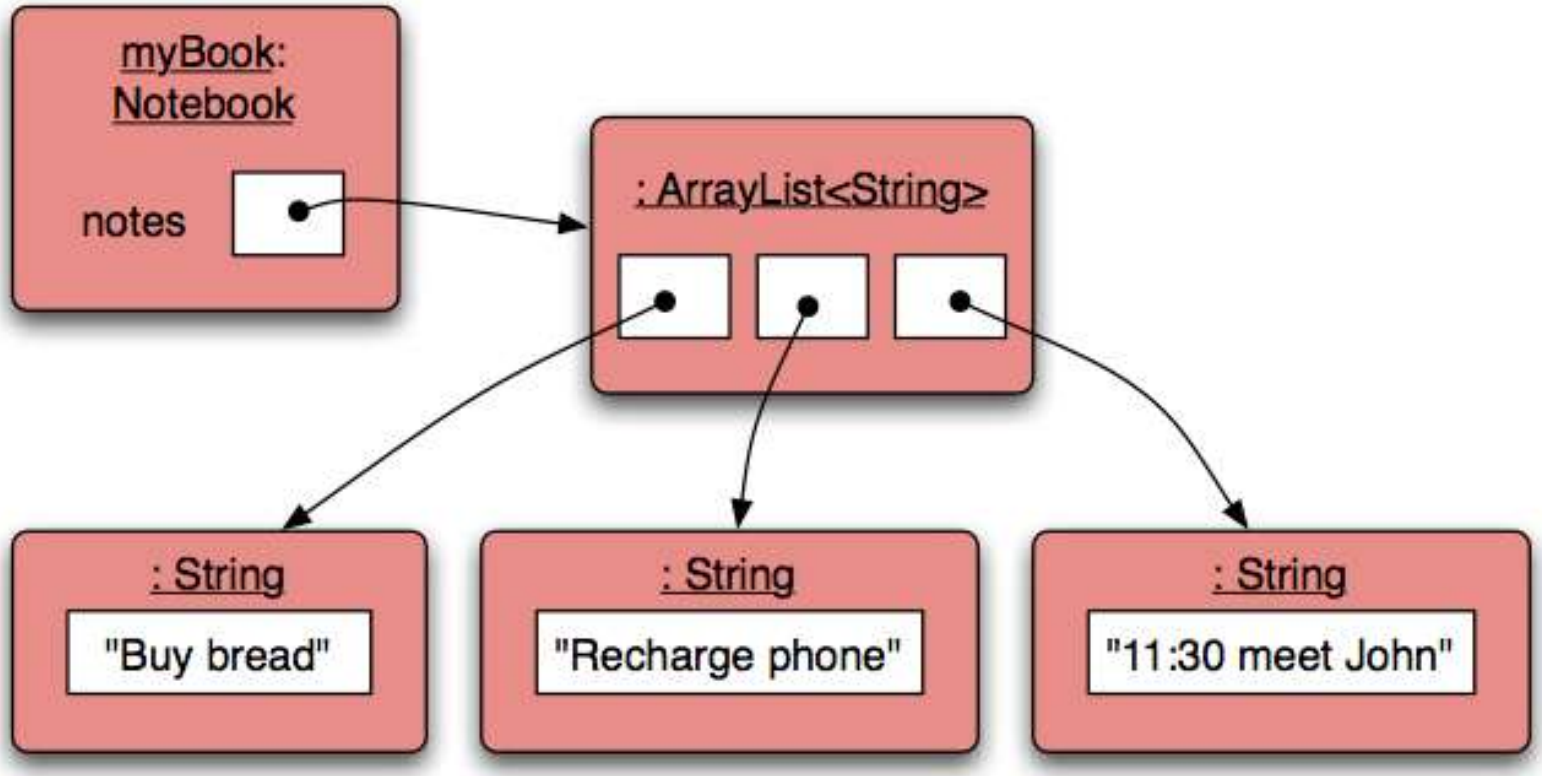
# بخشی از API کلاس ArrayList



# دیاگرام شی notebook



# اضافه کردن یک گره جدید



# ویژگی‌های مجموعه‌ها

- در صورت لزوم می‌توانند ظرفیت خود را افزایش دهند.
- تعداد اجزای خود را نگهداری می‌کند.
  - `size()` accessor
- اشیا را با یک ترتیب مشخص نگهداری می‌کند.
- جزئیات مربوط به انجام این امور مخفی است.
  - آیا این مهم است؟ آیا نشناختن ما مانع از استفاده از آن می‌شود.
- این امر اصول _____ و _____ شی‌گرایی را دنبال می‌کند.
  - جداسازی _____ و _____.

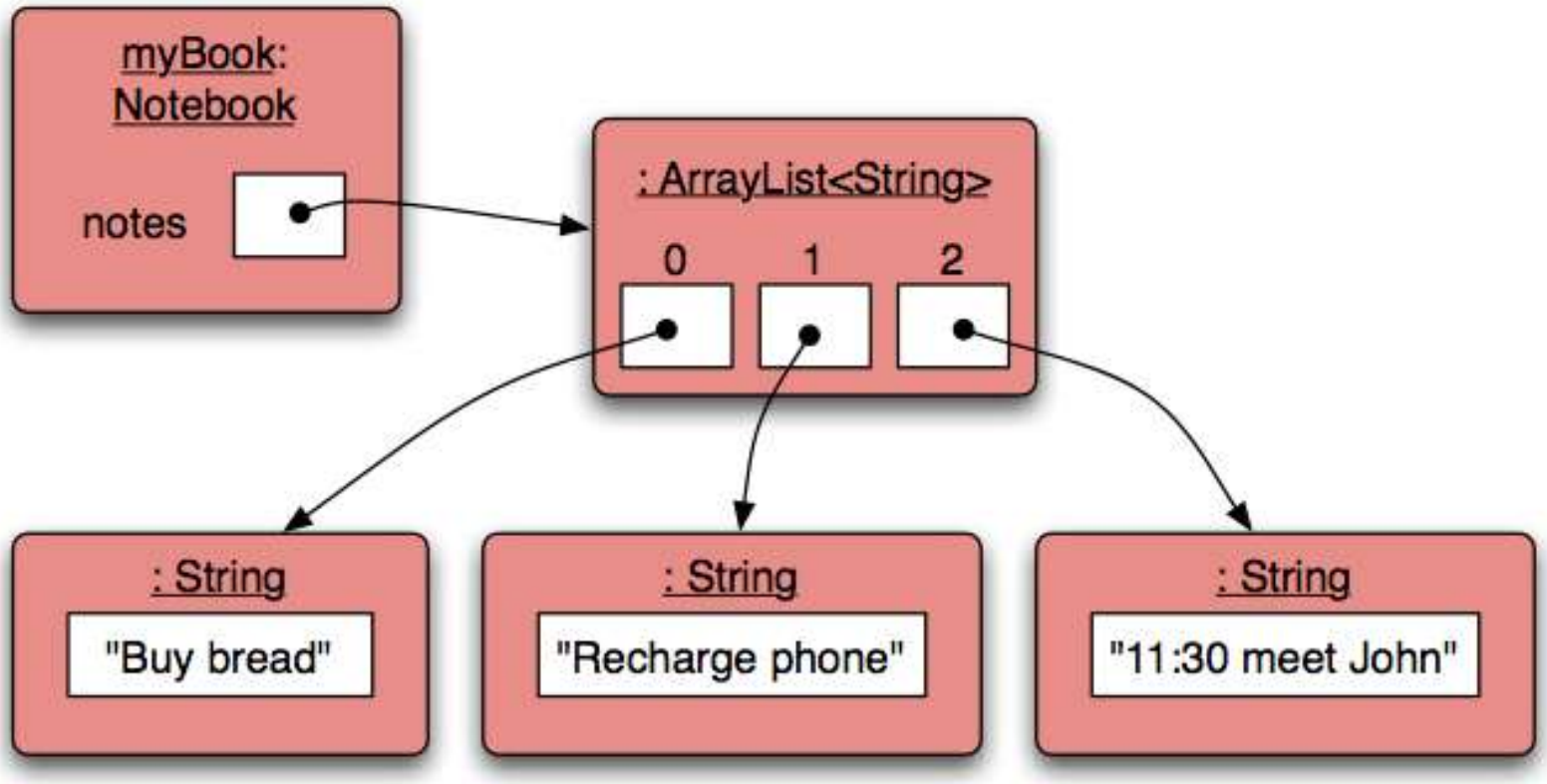
# استفاده از مجموعه

```
public class Notebook
{
    private ArrayList<String> notes;
    ...

    public void storeNote(String note)
    {
        notes.add(note);
    }

    public int numberOfNotes()
    {
        return notes.size();
    }

    ...
}
```

اضافه کردن یک نوشته جدید

تعداد نوشته‌ها را بازمی‌گرداند

# شماره‌گذاری شاخص‌ها (Index numbering)




**شاخص‌ها در جاوا از صفر شروع می‌شوند**

# بازیابی یک شی

```
public void showNote(int noteNumber)
{
    if(noteNumber >= 0 && noteNumber < numberOfNotes()) {
        String note = notes.get(noteNumber);
        System.out.println(note);
    }
    else {
        // This is not a valid note number.
    }
}
```

بررسی معتبر بودن شاخص

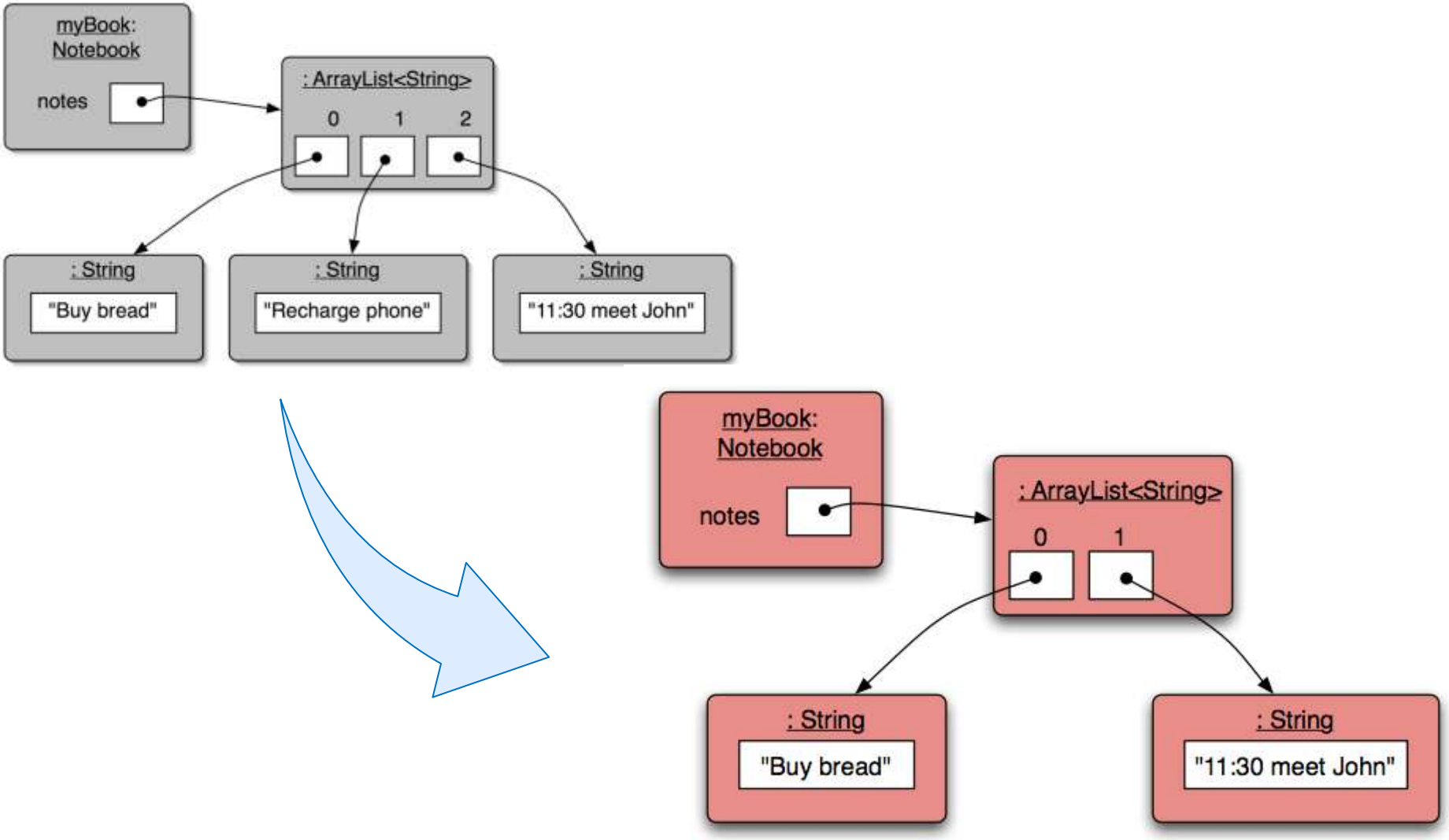
بازیابی و چاپ نوشته

آیا لازم است؟ (به شکل پیغام؟)

# حذف یک شی

```
public void removeNote(int noteNumber)
{
    if(noteNumber >= 0 && noteNumber < numberOfNotes()) {
        notes.remove(noteNumber);
    }
    else {
        // This is not a valid note number.
    }
}
```

# حذف ممکن است شاخص‌ها را تحت تاثیر قرار دهد!



# Notebook یک نمونه کد OO است!

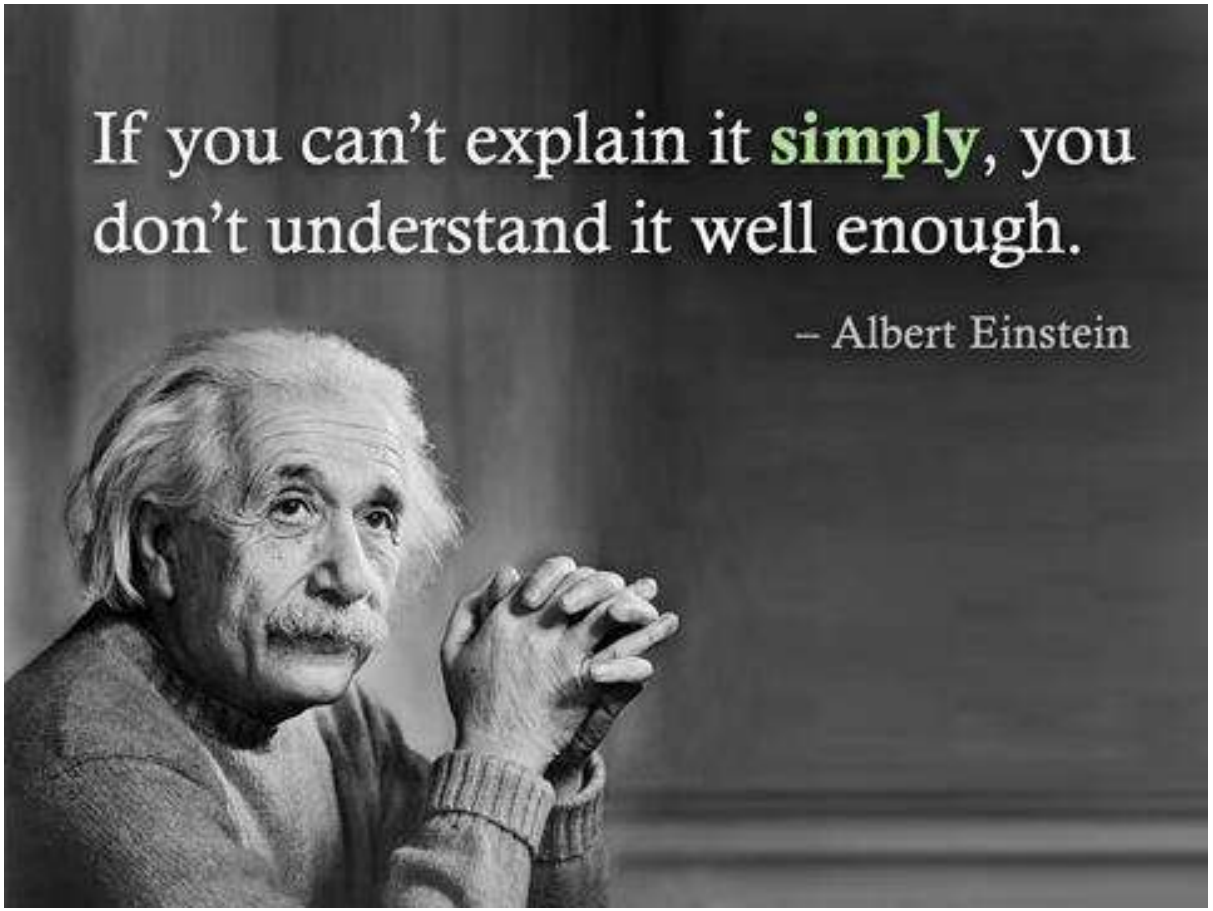
- از یک کلاس کتابخانه برای انجام بخش اعظم کار استفاده می‌کند.
  - پیغام‌هایی به ArrayList ارسال می‌کند.
  - می‌گوییم Notebook امور را به ArrayList محول (*delegate*) می‌کند.
  - از کلاس Notebook برای پنهان کردن کتابخانه کلاس (ArrayList) استفاده می‌کنیم (محصورسازی). می‌توان بعدا ArrayList را جایگزین کنیم.
  - OOP شامل پاس کردن پیغام‌های فراوانی است.
    - در برابر برنامه‌نویسی دستوری (آمرانه) که بیشتر در مورد دستورات گام به گام است.
  - Notebook به ما امکان معتبرسازی را می‌دهد.

# مرور

- مجموعه‌ها (collections) به ما کمک می‌کنند که تعداد دلخواه از اشیا را ذخیره کنیم.
- کتابخانه کلاس معمولا شامل تعدادی کلاس مجموعه امتحان و تست شده است.
- کتابخانه کلاس معمولا بسته خواند می‌شوند.
- بسته‌ها بخشی از API جاوا هستند.
- ما از کلاس کتابخانه‌ای ArrayList از بسته java.util استفاده کردیم.

# مرور ArrayList

- المان‌ها ممکن است اضافه یا حذف شوند.
- هر المان دارای یک شاخص است.
- شاخص‌ها ممکن است با حذف یک المان تغییر کنند.
- متدهای اصلی ArrayList:
  - add, get, remove and size
- ArrayList یک نوع پارامتری شده (parameterized type) یا جنسی (generic) است.


If you can't explain it simply, you don't understand it well enough.
– Albert Einstein

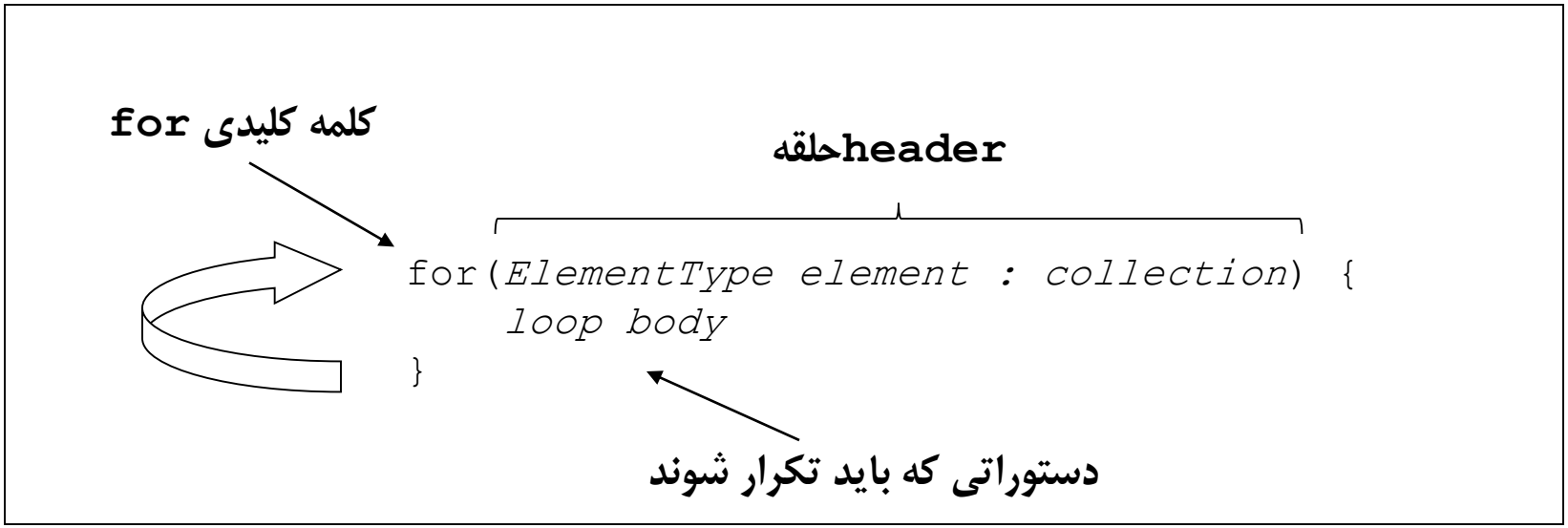
# تکرار

- گاهی اوقات لازم است یک عمل را روی تعداد دلخواهی از المان‌ها انجام دهیم.
  - مثلا: تمام نوشته‌های دورن `Notebook` را چاپ کن.
    - چند نوشته وجود دارد؟
- اکثر زبان‌های برنامه‌نویسی شامل دستورات تکرار برای میسر کردن این امور هستند.
- جاوا دارای دستورات تکرار متعددی است.
- حلقه‌ها در جاوا همانند `C` هستند:
  - `for, while, do/while`
- جاوا همچنین دارای حلقه‌های ارتقاع یافته‌تری نیز هست.

# مفهوم تکرار

- گاهی می‌خواهیم یک عمل را چند بار تکرار کنیم.
- حلقه‌ها به ما امکان می‌دهند تا دفعات تکرار این عمل‌ها را کنترل کنیم.
- در کار با مجموعه‌ها معمولا می‌خواهیم یک عمل یکبار روی تمام المان‌های درون مجموعه انجام شود.
  - for-each

# شبه کد حلقه for-each




**برای هریک از المان‌های درون مجموعه، دستورات درون بدنه حلقه را انجام بده**

**Read "for" as "for each" and ":" as "in"**

# یک مثال در جاوا

```
public void listNotes()
{
    for(String note : notes) {
        System.out.println(note);
    }
}
```

به ازای هر note درون notes، آنرا چاپ کن

# پروسه انتخاب

- دستورات می‌تواند به صورت تو در تو شامل انتخاب بیشتری باشد:

```
public void findNote(String searchString)
{
    for(String note : notes) {
        if(note.contains(searchString)) {
            System.out.println(note);
        }
    }
}
```

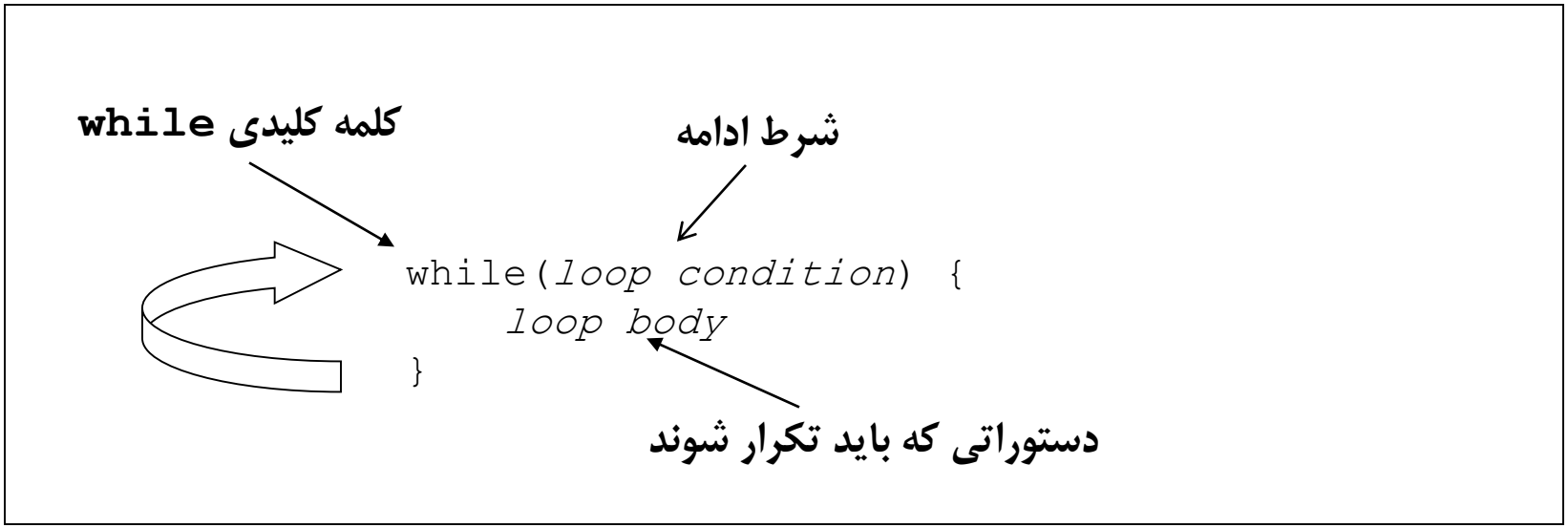
# خوبی‌ها و بدی‌های `for-each`

- نوشتن آن ساده است.
- خاتمه به طور طبیعی اتفاق می‌افتد (بعد از پیمایش کل مجموعه).
- مجموعه مورد پیمایش را نمی‌توان عوض کرد
- هیچ شاخصی ایجاد نکردیم.
  - همه مجموعه‌ها مبتنی بر شاخص نیستند.
- نمی‌توانیم از طریق آن بایستیم!
  - مثلا پیدا کردن اولین المانی که با شرط جور بود
- تکرار مشخص (`definite iteration`) را فراهم می‌آورد – تکرار محدود (`bounded iteration`).

# حلقه `while`

- حلقه `for-each`، بدنه حلقه را به ازای هر المان درون مجموعه تکرار می‌کند.
- گاهی به تغییراتی بیشتر از این نیاز داریم.
- می‌توانیم از یک شرط بولین (`boolean`) برای تصمیم گیری در مورد اینکه به تکرار ادامه دهیم یا نه استفاده کنیم.
- حلقه `while` این کنترل را برای ما فراهم می‌کند.

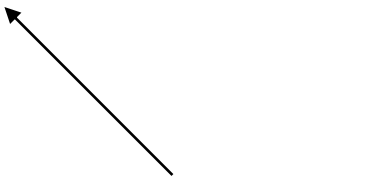
# شبه کد حلقه while




**تا زمانی که می‌خواهیم ادامه دهیم، دستورات درون بدنه حلقه را انجام بده**

# یک مثال در جاوا

```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int index = 0;
    while(index < notes.size()) {
        System.out.println(notes.get(index));
        index++;
    }
}
```




**تا زمانی که مقدار شاخص از اندازه مجموعه کوچکتر است، المان را چاپ کرده شاخص را زیاد کن**

# مقایسه `for-each` و `while`

- `for-each`
  - نوشتن ساده‌تر
  - مطمئن‌تر: پایان یافتن حلقه تضمین شده است.

- `while`
  - لازم نیست حتما کل مجموعه را پردازش کنیم.
  - حتما نباید با یک مجموعه مورد استفاده قرار گیرد
  - *مواظب باشید!* خطر ایجاد حلقه بینهایت

# جستجو در یک مجموعه

```
int index = 0;
boolean found = false;
while(index < notes.size() && !found) {
    String note = notes.get(index);
    if(note.contains(searchString)) {
        // We don't need to keep looking.
        found = true;
    }
    else {
        index++;
    }
}
// Either we found it, or we searched the whole
// collection.
```

# while بدون مجموعه!

```
// Print all even numbers from 0 to 30.
int index = 0;
while(index <= 30) {
    System.out.println(index);
    index = index + 2;
}
```

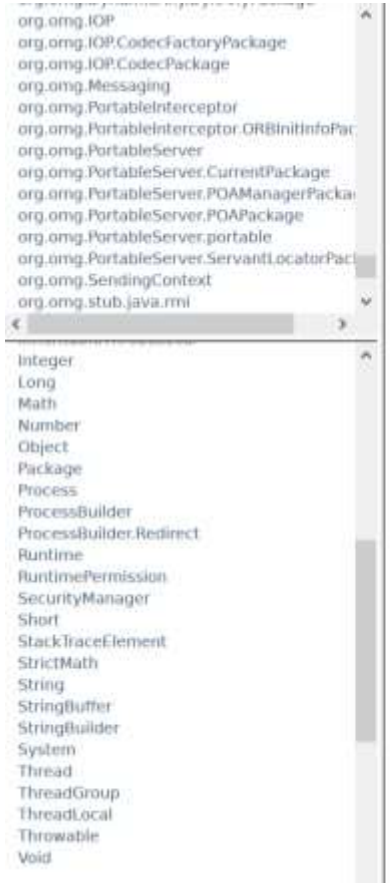
# while بدون مجموعه!

```
public void print()
{
    int i = 1;
    while(i <= 10) {
        System.out.println(i);
        i++;
    }
}
```

```
public void print()
{
    int i = 0;
    while(++i <= 10) {
        System.out.println(i);
    }
}
```

```
public void print()
{
    int i = 0;
    while(i++ < 10) {
        System.out.println(i);
    }
}
```

# Class String API




| Modifier and Type | Method and Description |
|---|---|
| char | **charAt**(int index)<br>Returns the char value at the specified index. |
| int | **compareTo**(String anotherString)<br>Compares two strings lexicographically. |
| int | **compareToIgnoreCase**(String str)<br>Compares two strings lexicographically, ignoring case differences. |
| String | **concat**(String str)<br>Concatenates the specified string to the end of this string. |
| boolean | **contains**(CharSequence s)<br>Returns true if and only if this string contains the specified sequence of char values. |
| boolean | **endsWith**(String suffix)<br>Tests if this string ends with the specified suffix. |
| boolean | **equals**(Object anObject)<br>Compares this string to the specified object. |
| int | **indexOf**(int ch)<br>Returns the index within this string of the first occurrence of the specified character. |
| int | **indexOf**(int ch, int fromIndex)<br>Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index. |
| int | **indexOf**(String str)<br>Returns the index within this string of the first occurrence of the specified substring. |
| int | **indexOf**(String str, int fromIndex)<br>Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. |

# تساوی رشته‌ها

```
if(input == "bye") {
    ...
}
```

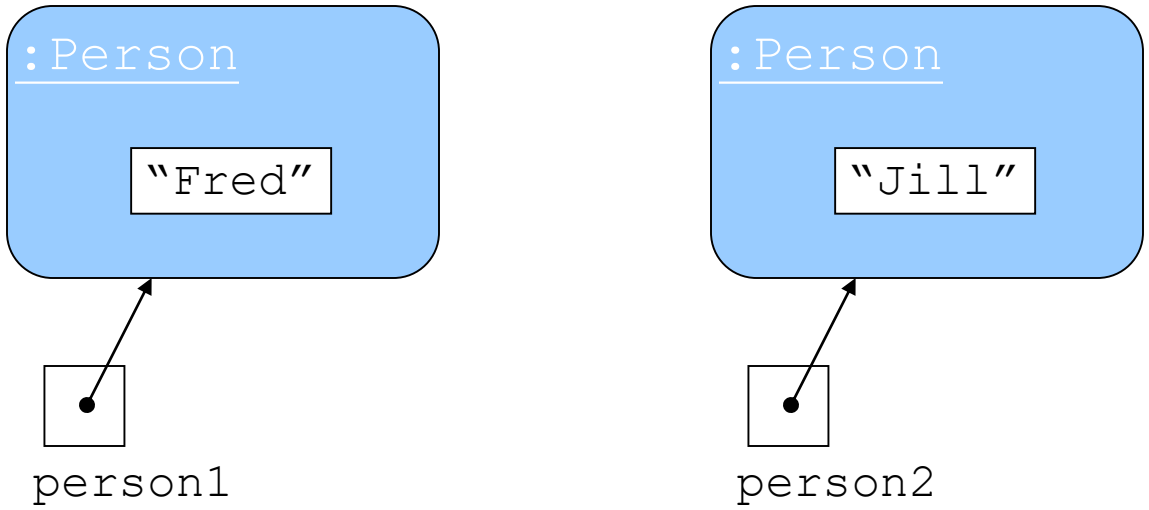
یکسانی هویت را بررسی می‌کند

```
if(input.equals("bye")) {
    ...
}
```

یکسانی مقدار را بررسی می‌کند

- مقایسه‌ی بین دو رشته باید با متد `equals` انجام پذیرد. == یکسان بودن مرجع دو رشته را بررسی می‌کند

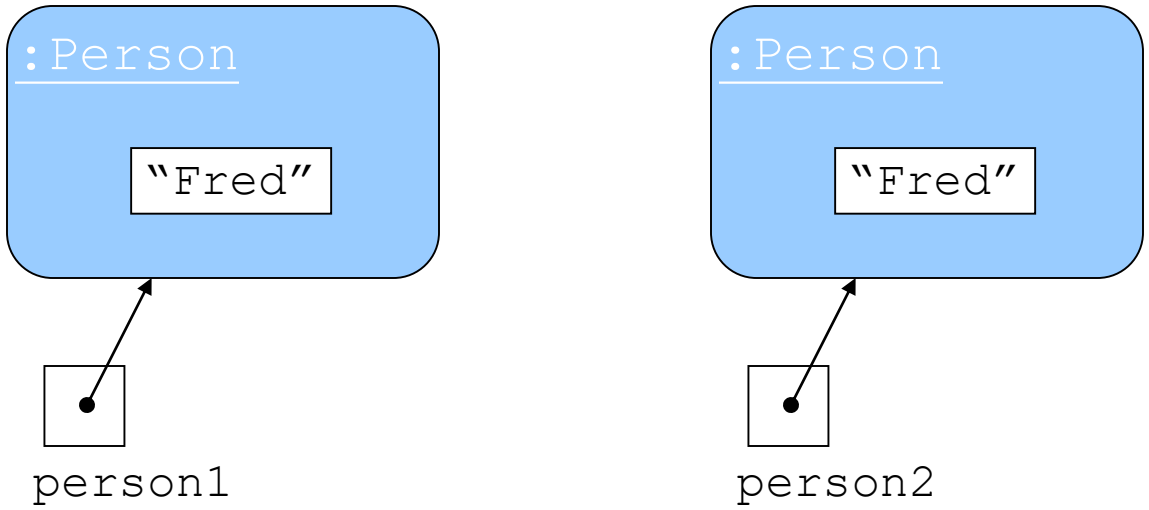
# تساوی رشته‌ها




```
person1 == person2  ?
```

**خیر!**

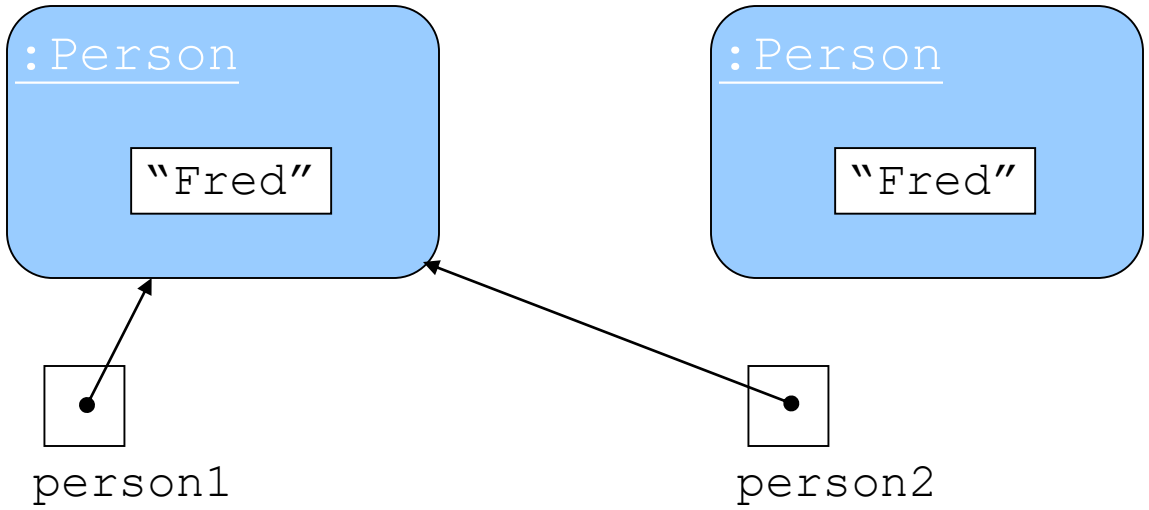
# تساوی رشته‌ها




```
person1 == person2  ?
```

**خیر!**

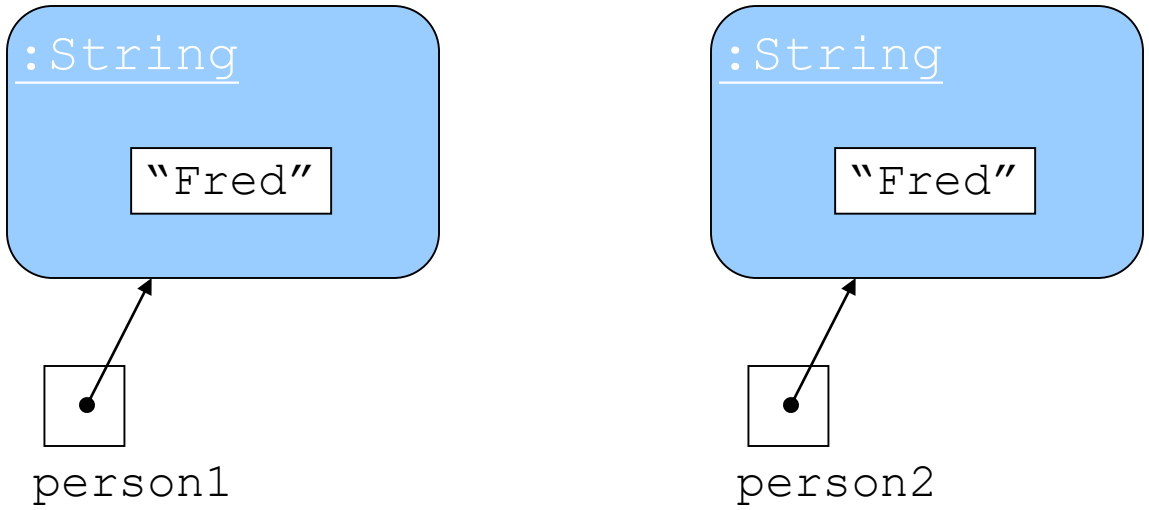
# تساوی رشته‌ها




```
person1 == person2  ?
```

**بلی!**

# تساوی رشته‌ها




```
person1 == person2  ?
```

**بلی!**

# تساوی رشته‌ها

```
String person1 = "Fred";
person1 == "Fred"
```

بلی!

```
String person2 = "Fred";
person1 == person2;
```

بلی!

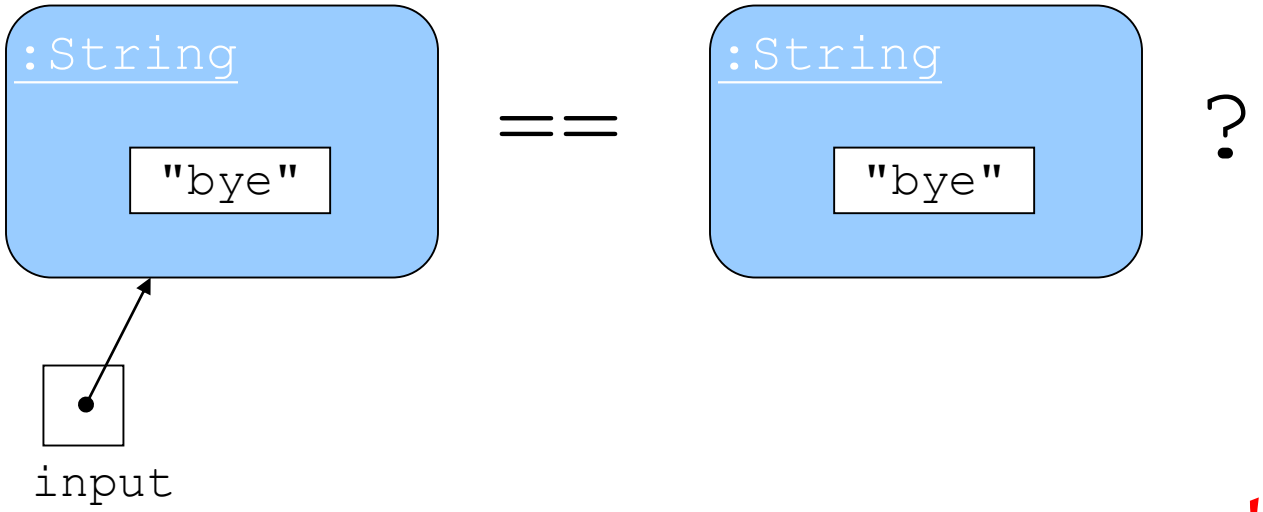
# تساوی رشته‌ها

- کامپایلر رشته‌های یکسان در کد را در یکدیگر ادغام می‌کند.
  - نتیجه‌ی این کار یکسان شدن مرجع رشته‌های با مقدار یکسان است.
- این امر در مورد رشته‌هایی که از بیرون از کد برنامه می‌آید امکان‌پذیر نمی‌باشد.
  - مثلا ورودی‌های کاربر
  - می‌توانیم برای بررسی تساوی مقدار (و نه مرجع) از متد `equals` استفاده کنیم.

# تساوی رشته‌ها

```
String input = reader.getInput();
if(input == "bye") {
    ...
}
```

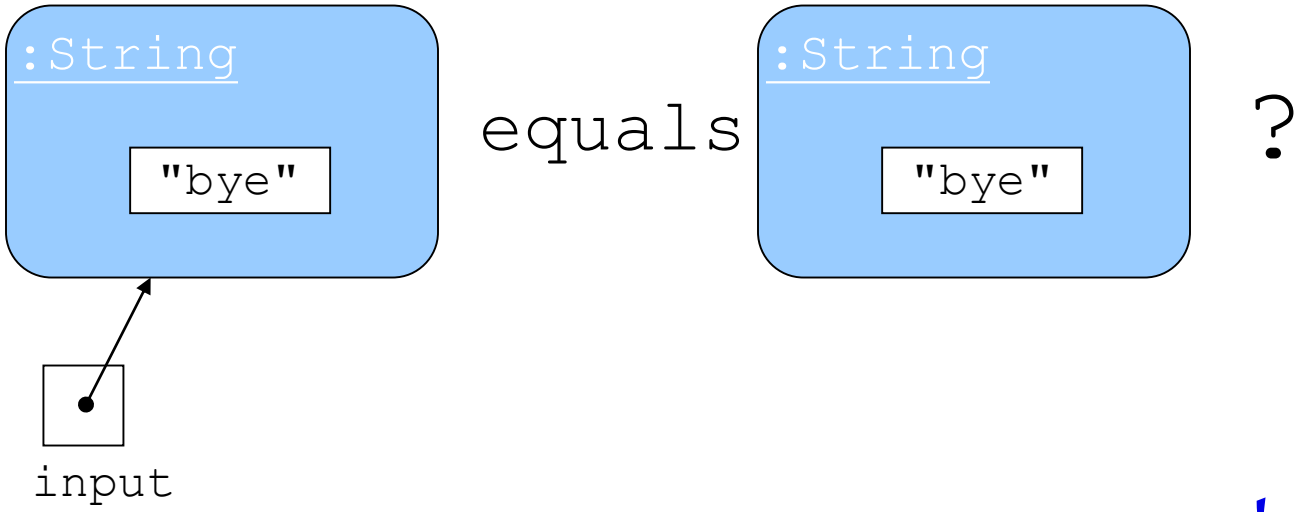
دریافت ورودی از کاربر




**خیر!**

# تساوی رشته‌ها

```
String input = reader.getInput();
if(input.equals("bye")) {
    ...
}
```




بلی!

# Iterator و iterator()

- مجموعه‌ها دارای یک متد iterator() هستند.
- این متد یک شی Iterator بازمی‌گرداند
- Iterator<E> دارای سه متد است:
  - boolean hasNext()
  - E next()
  - void remove()

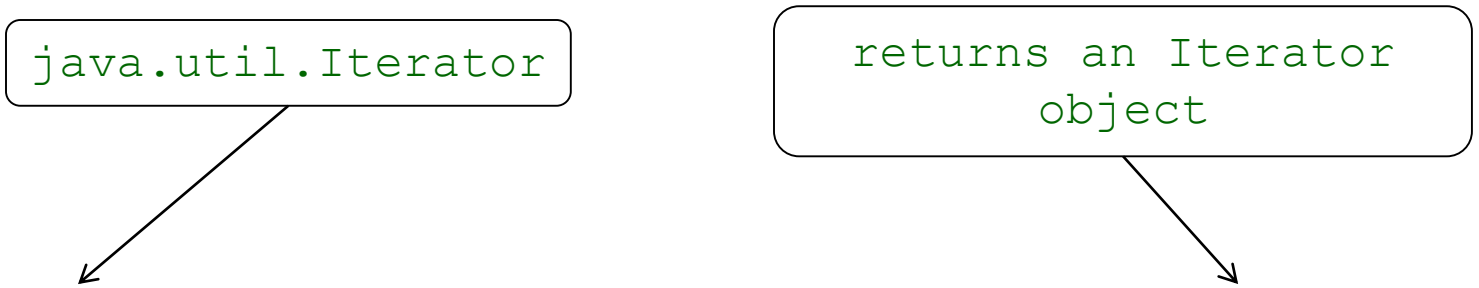
# Iterator و iterator()

- برای پیمایش مجموعه‌ها استفاده می‌شود.
- در بسته زیر قرار دارد

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

- اگر remove() را وقتی فراخوانی کنید که next() تا آن موقع اصلا فراخوانی نشده باشد، با خطا مواجه می‌شوید.

# استفاده از شی تکرار (Iterator)




```
Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}
```

# ایجاد Iterator

- شی Iterator با فراخوانی متد iterator() مجموعه ایجاد می‌شود.

```
Iterator it = notes.iterator();
```

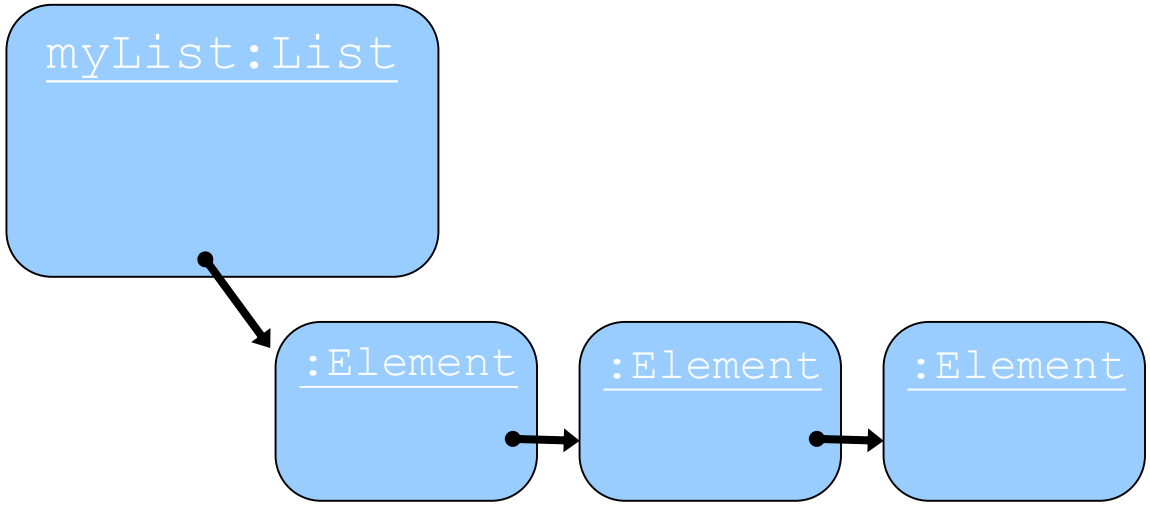
نه با نحو (syntax) مرسوم ایجاد شی

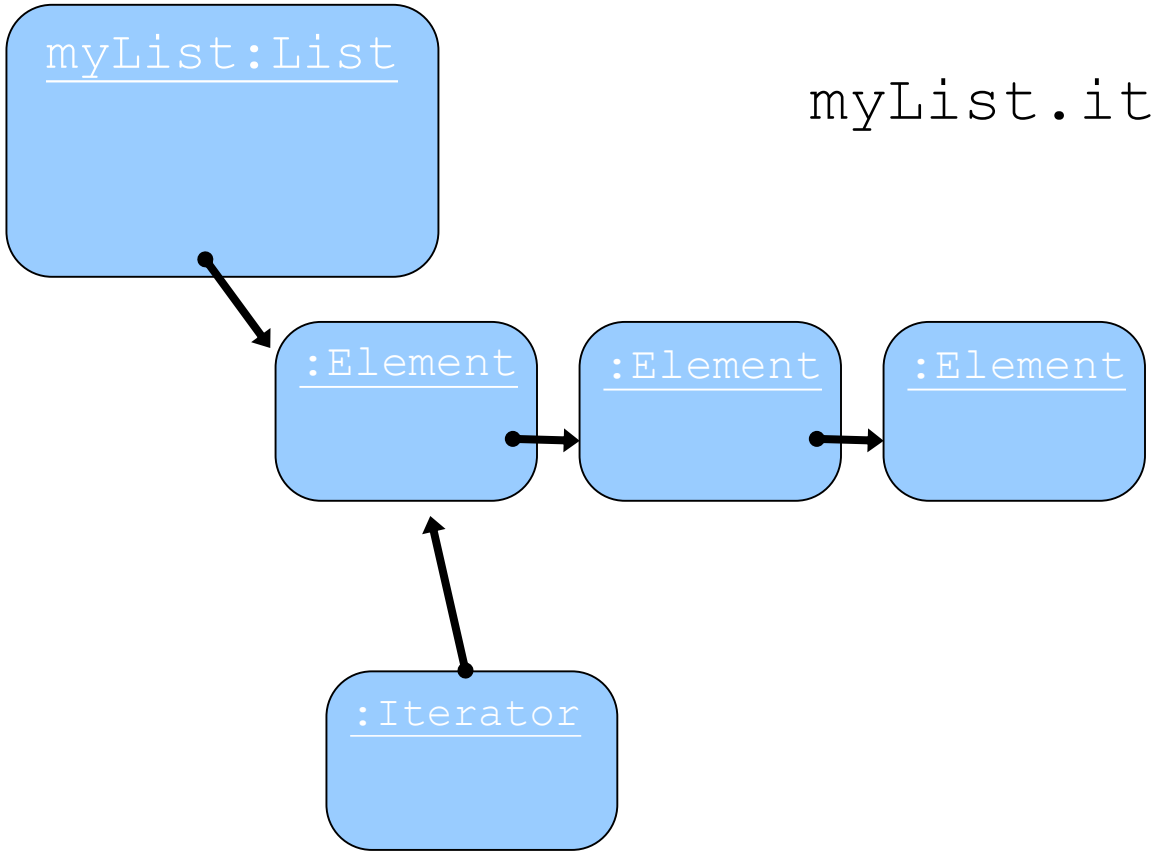
```
Iterator it = new Iterator();
```

- از مجموعه‌ها برای ایجاد ایجاد Iterator استفاده می‌کنیم بنابراین:
  - Iterator از شاخص‌های مجموعه خبر دارد.
  - هر مجموعه می‌تواند Iterator مربوط به خود را داشته باشد، اما همه آنها رابطهای هم‌شکل دارند.

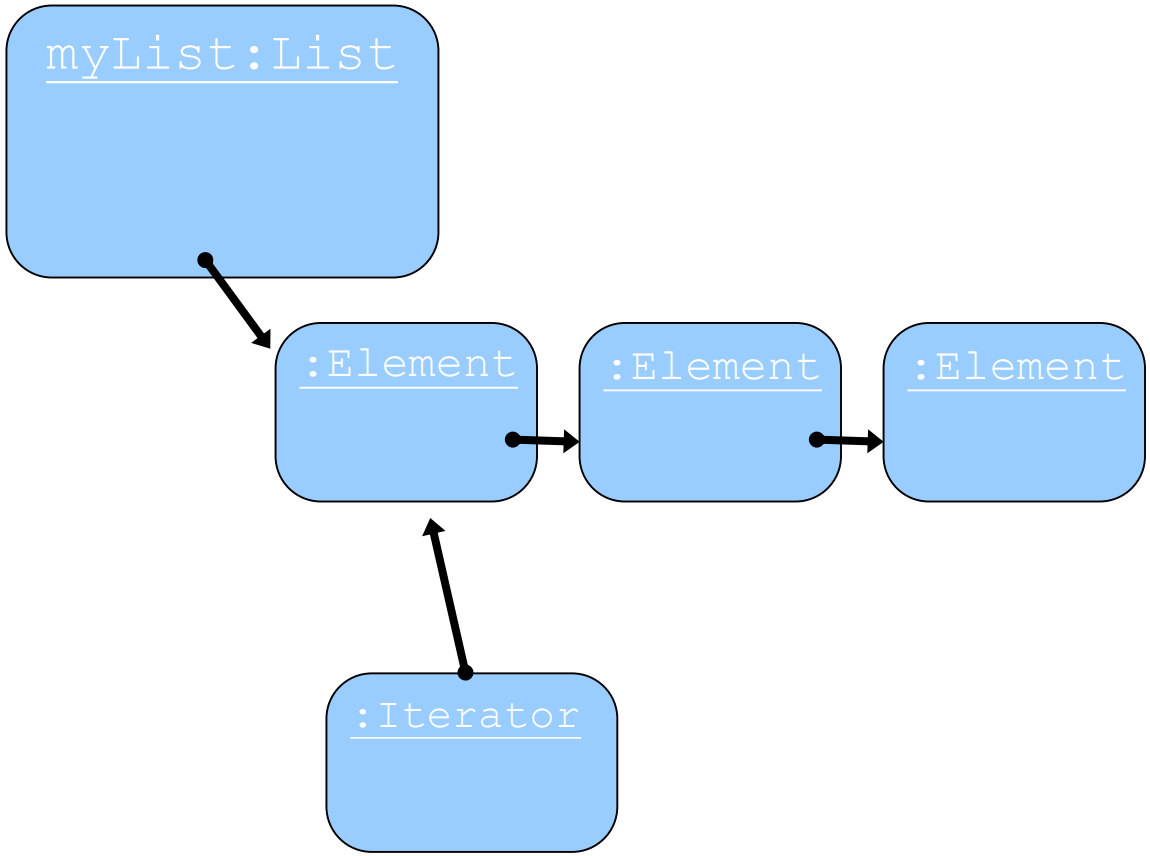
# استفاده از شی تکرار (Iterator)

```
public void listNotes()
{
    Iterator<String> it = notes.iterator();
    while(it.hasNext()) {
        System.out.println(it.next());
    }
}
```

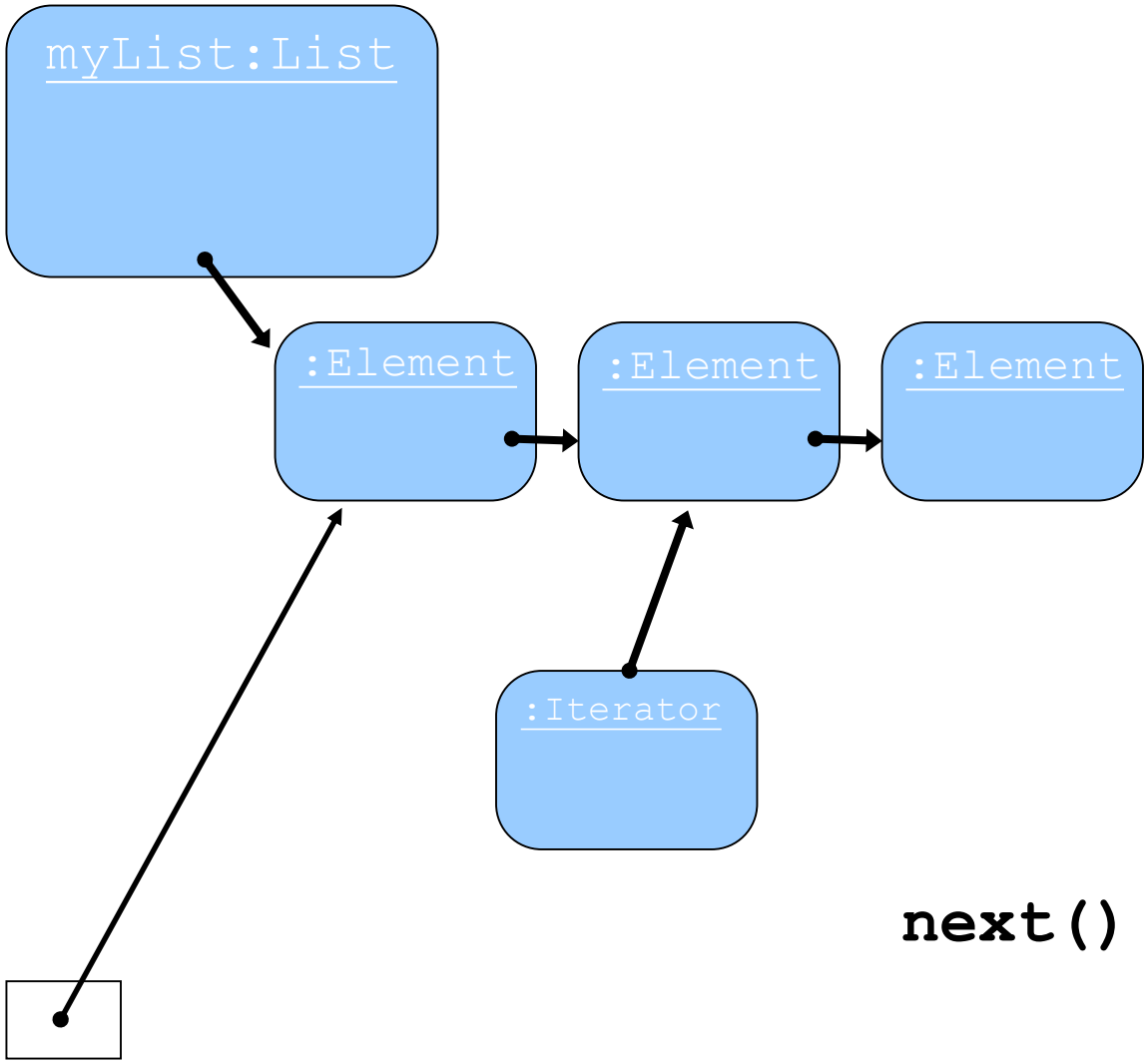

myList:List
:Element
:Element
:Element



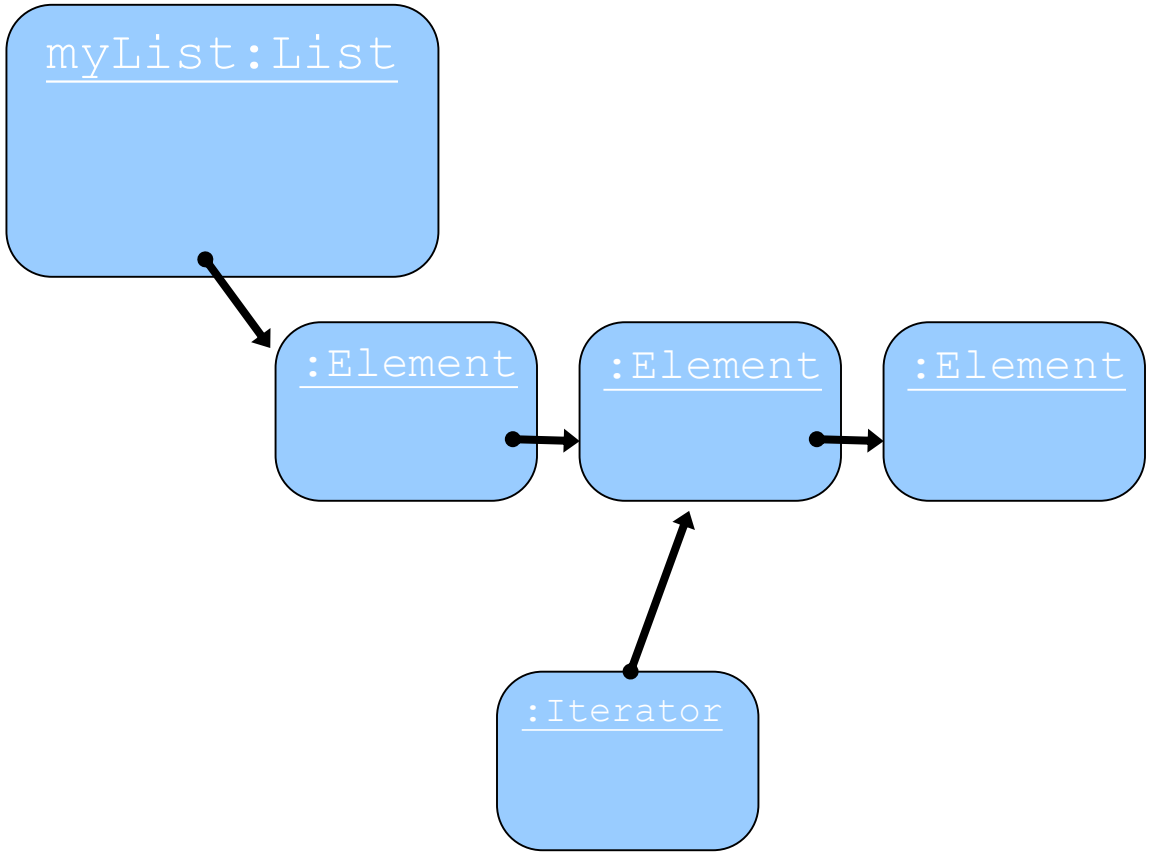

```
myList.iterator()
```



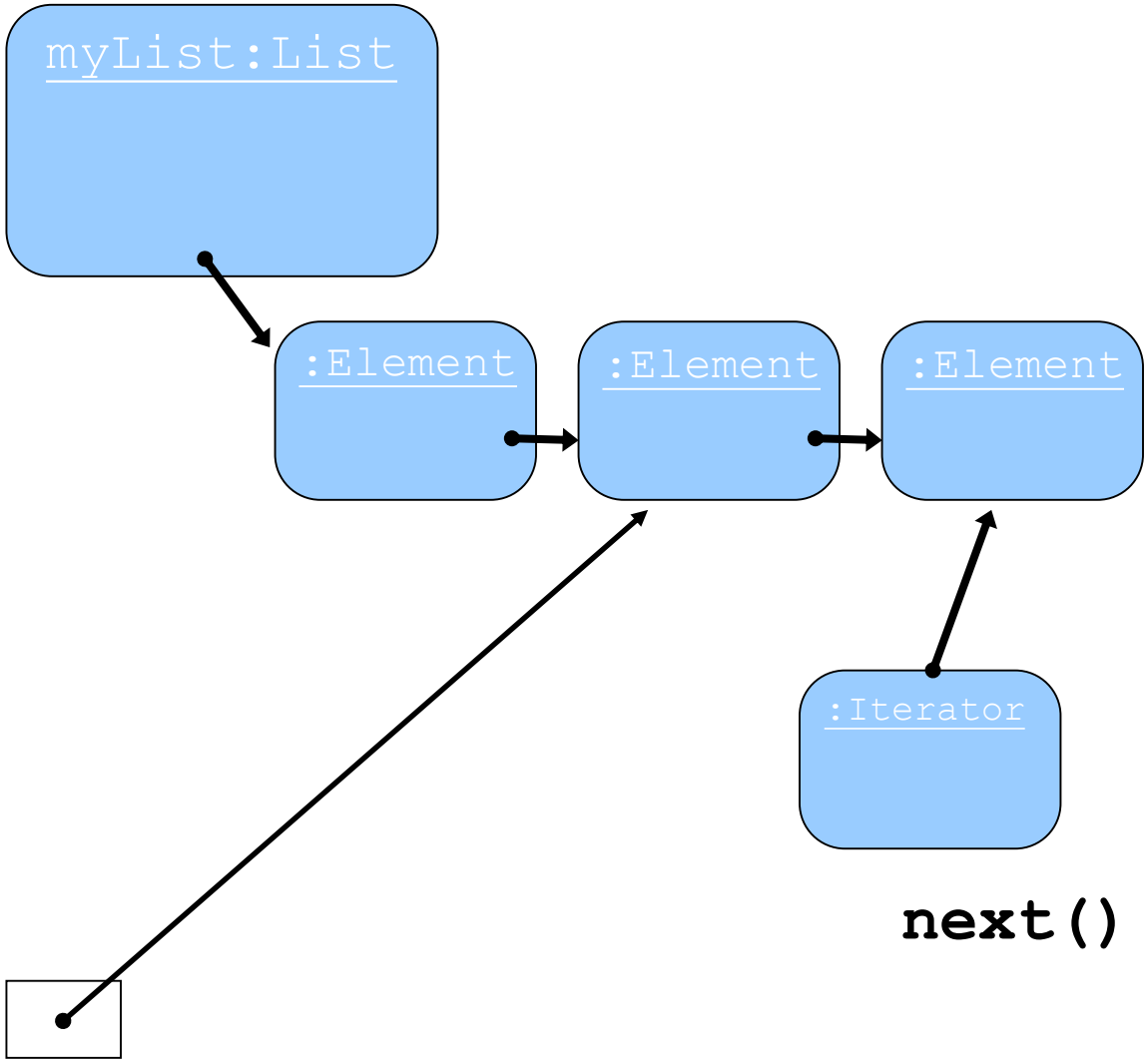

**hasNext()?** ✔

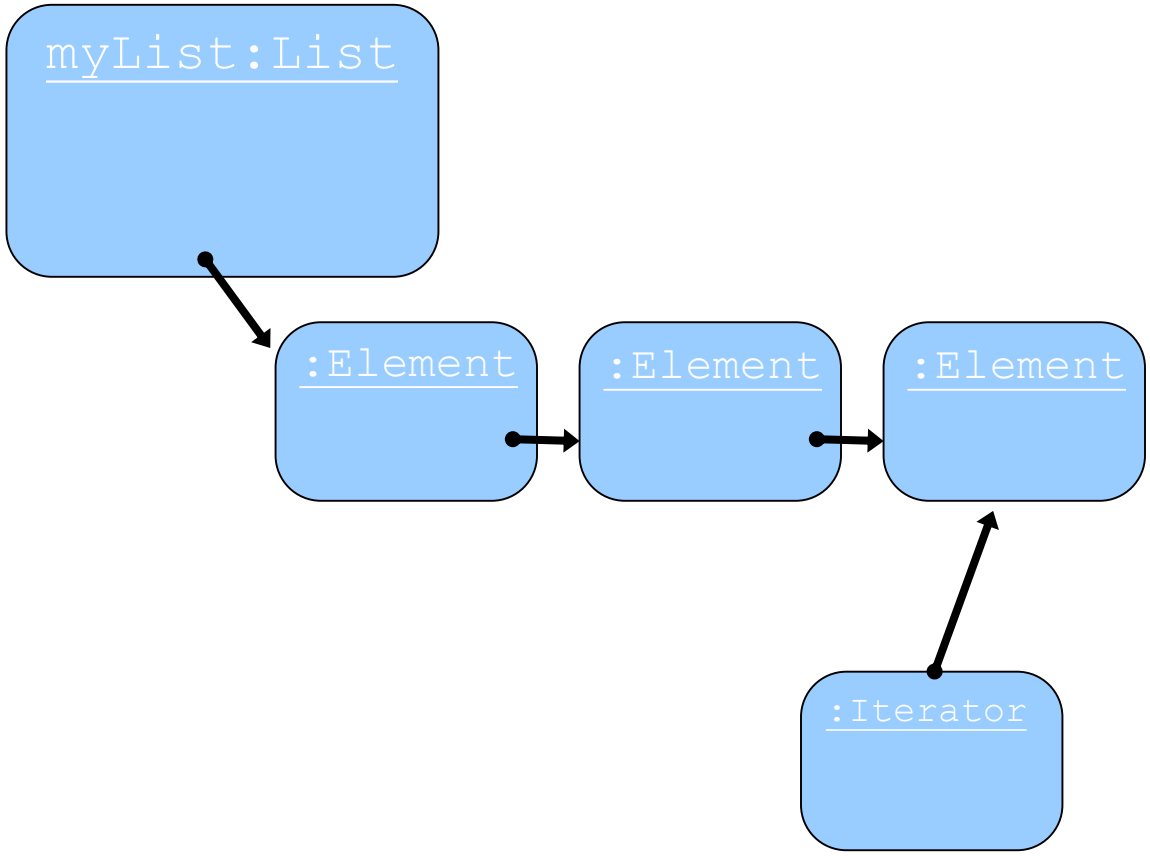



```
Element e = iterator.next();
```

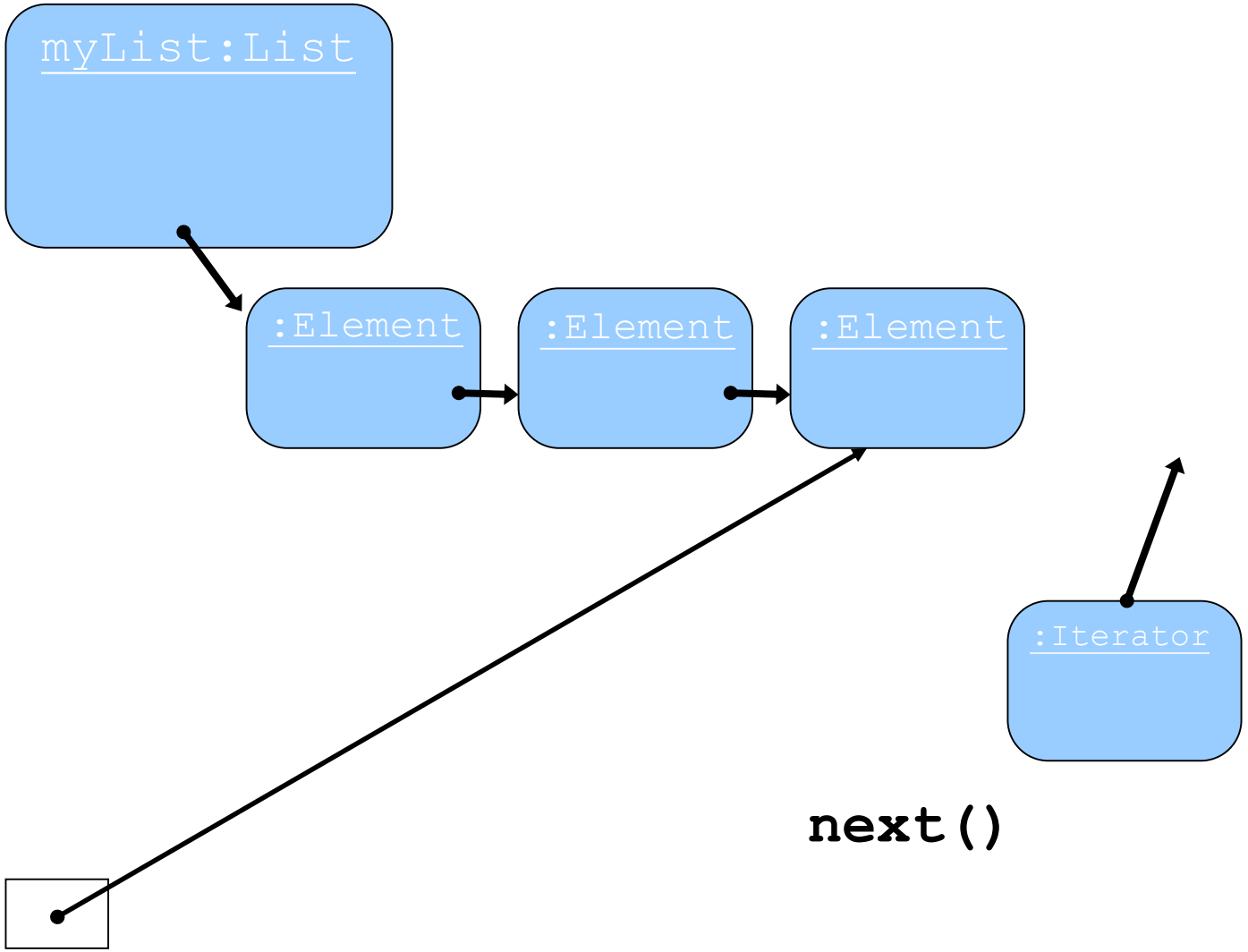


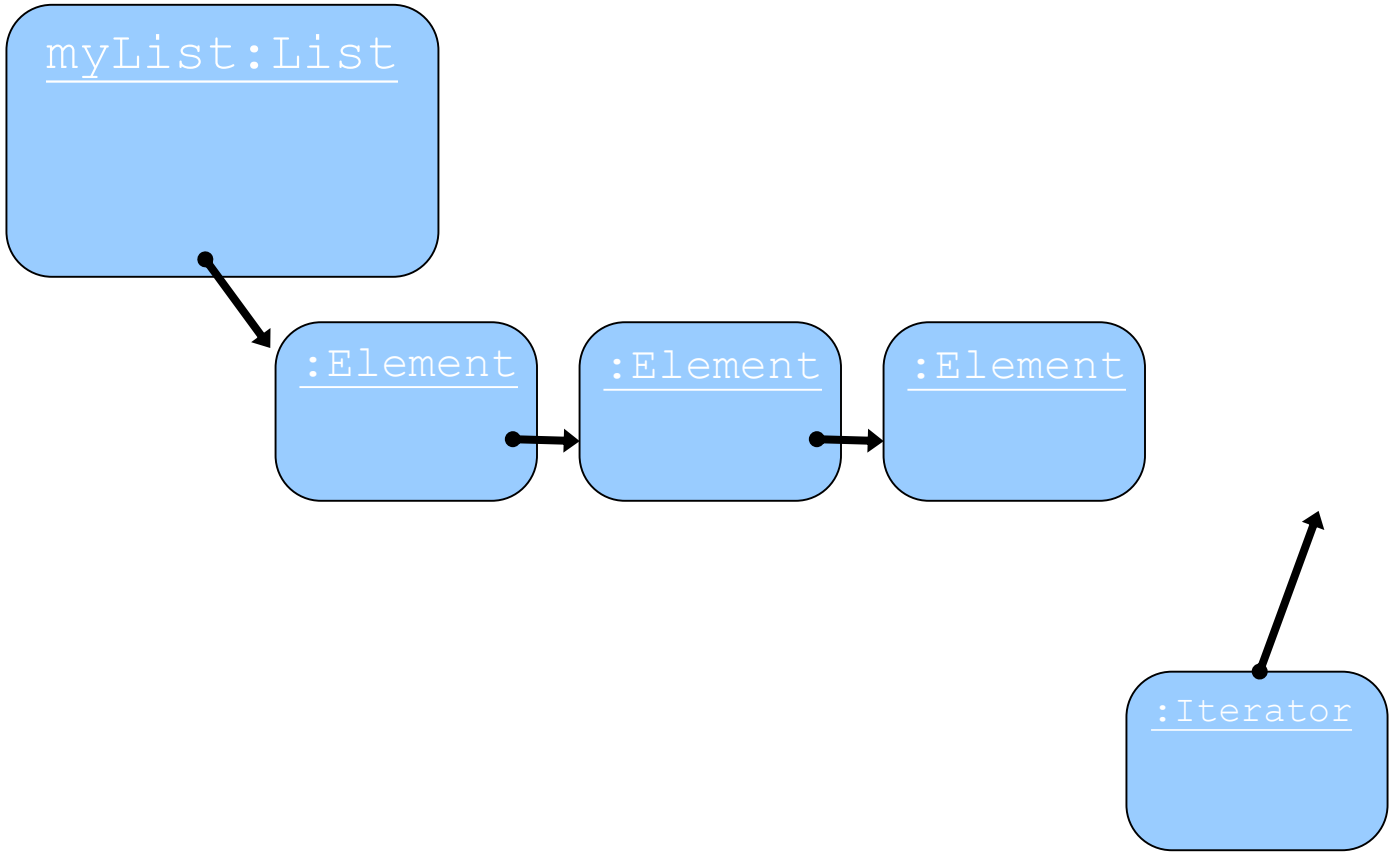

**hasNext()?** ✔


myList:List
:Element
:Element
:Element
:Iterator
next()




**hasNext()?** ✔


myList:List
:Element
:Element
:Element
:Iterator
next()




**hasNext()?** ✗

# حذف از یک مجموعه

```
public void removeNoteByContent(String searchString)
{
      Iterator<String> it = notes.iterator();
      while(it.hasNext()) {
            String note = it.next();
            if(note.contains(searchString)) {
                  it.remove();
            }
      }
}
```

# مقایسه شاخص (index) با Iterator

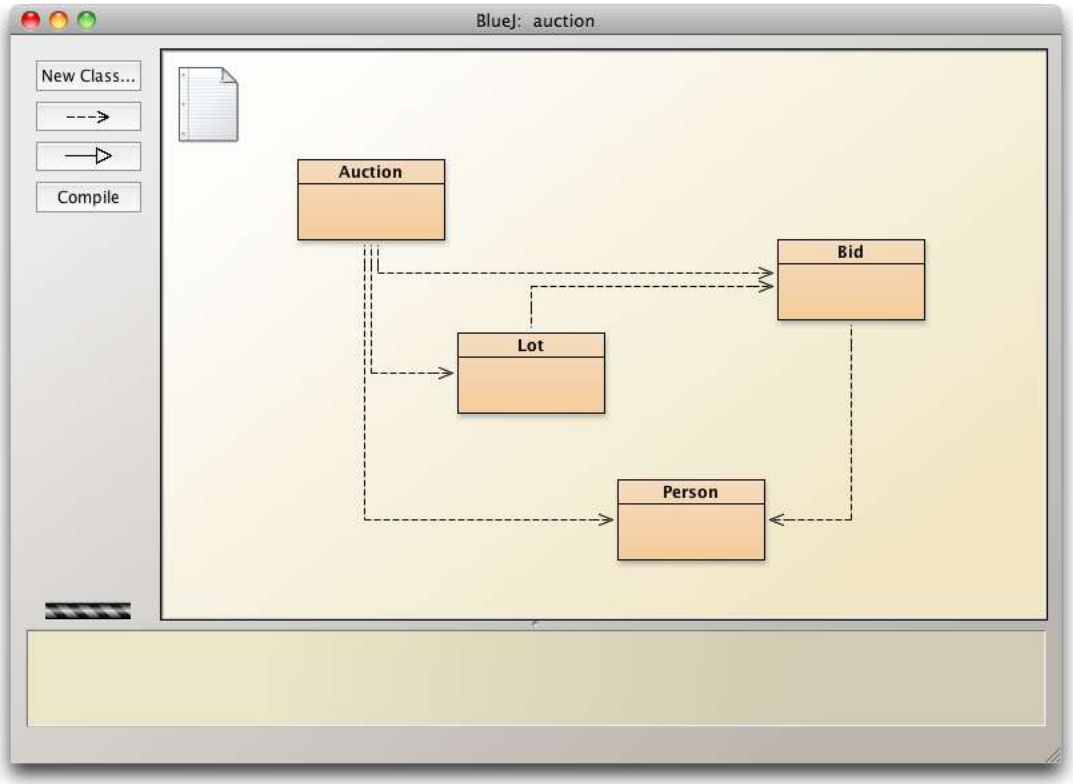
- روش‌های پیمایش یک مجموعه
  - حلقه for-each
    - استفاده زمانی که می‌خواهیم تمام المان‌ها را بررسی کنیم
  - حلقه while
    - استفاده زمانی که بخواهیم وسط راه بایستیم!
    - تکرار زمانی که مجموعه وجود ندارد.
  - شی Iterator
    - استفاده زمانی که بخواهیم وسط راه بایستیم!
    - اغلب زمانی استفاده می‌شود که استفاده از شاخص کارآمد نیست یا غیر ممکن است.
      - پیمایش و حذف برخی المان‌های لیست
- تکرار یکی از الگوهای مهم برنامه‌نویسی است.

# مرور

- دستور تکرار، امکان تکرار یک بلوک از دستورات را برای دفعاتی ممکن می‌سازد.
- حلقه `for-each` امکان پیمایش تمام المان‌های یک مجموعه را می‌دهد.
- حلقه `while` امکان کنترل تکرار را به وسیله یک عبارت `boolean` فراهم می‌سازد.
- تمام کلاس‌های مجموعه یک امکان استفاده از یک شی `Iterator` برای دسترسی به المان‌های مجموعه را دارند.

# پروژه حراج (auction)

- این پروژه تصویر بهتری از مجموعه‌ها و `iteration` ارائه می‌کند.
  - استفاده از `null`
  - اشیا بدون نام
  - فراخوانی زنجیره‌ای متد



# null

- برای نوع داده‌ای شی استفاده می‌شود.
- به معنی «هیچ شی‌ای» است.
- می‌توان `null` بودن مقدار یک شی را بررسی کرد .

– **`if(highestBid == null)`** ...

– مشخص می‌کند **bid**ی وجود دارد یا نه!

# null


**The worst mistake of computer science**

# اشیای بدون نام (Anonymous objects)

- اشیا گاهی به محض ایجاد جای دیگری مورد استفاده قرار می‌گیرند

```
Lot furtherLot = new Lot(…);
lots.add(furtherLot);
```

- واقعا نیازی به `furtherLot` نداریم:

```
lots.add(new Lot(…));
```

# فراخوانی زنجیره‌ای متد

- متدها گاها یک شی را بازمی‌گردانند
- گاها سریعا برای شی بازگردانده شده متدی را فراخوانی می‌کنیم:

```
Bid bid = lot.getHighestBid();
Person bidder = bid.getBidder();
```

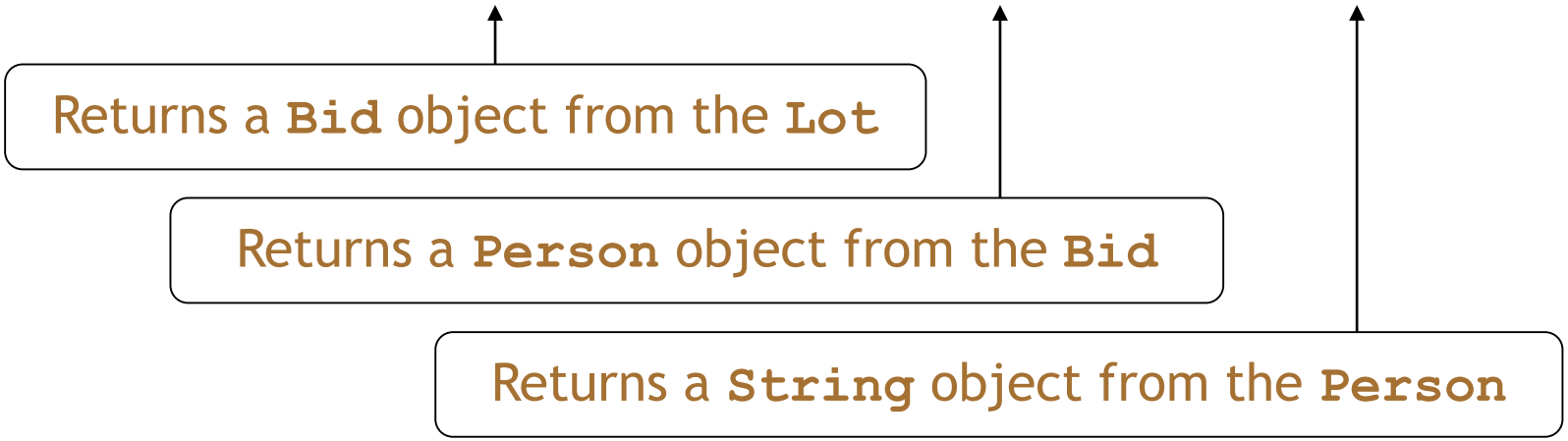
- می‌توانیم از مفهوم اشیا بدون نام و فراخوانی زنجیره‌ای متدها استفاده کنیم:

```
Person bidder =
lot.getHighestBid().getBidder()
```

# فراخوانی زنجیره‌ای متد

- هر متد در زنجیره برای شی‌ای فراخوانی می‌شود که متد قبلی آنرا بازگردانده است.

```
String name =
    lot.getHighestBid().getBidder().getName();
```



# مجموعه‌های با اندازه ثابت

- گاهی اوقات حداکثر اندازه مجموعه می‌تواند از پیش مشخص شود.
- زبان‌های برنامه‌نویسی معمولا یک نوع خاص از مجموعه‌ها با اندازه مشخص را پیشنهاد می‌دهند: آرایه‌ها (`Arrays`)
- آرایه‌ها در جاوا می‌توانند نوع‌های شی (`object type`)یا اولیه (`primitive type`) را در خود ذخیره کنند.
- آرایه‌ها از نحو (`syntax`) ویژه خود بهره می‌برند.

# پروژه weblog-analyzer

- وب سرور اطلاعات همه دسترسی‌ها را نگهداری می‌کند.
- عملیات آنالیزی را پیشتیانی می‌کند:
  - صفحات پربازدید
  - دوره‌های زمانی شلوغ
  - چه میزان داده منتقل شده است
  - مراجع خراب شده

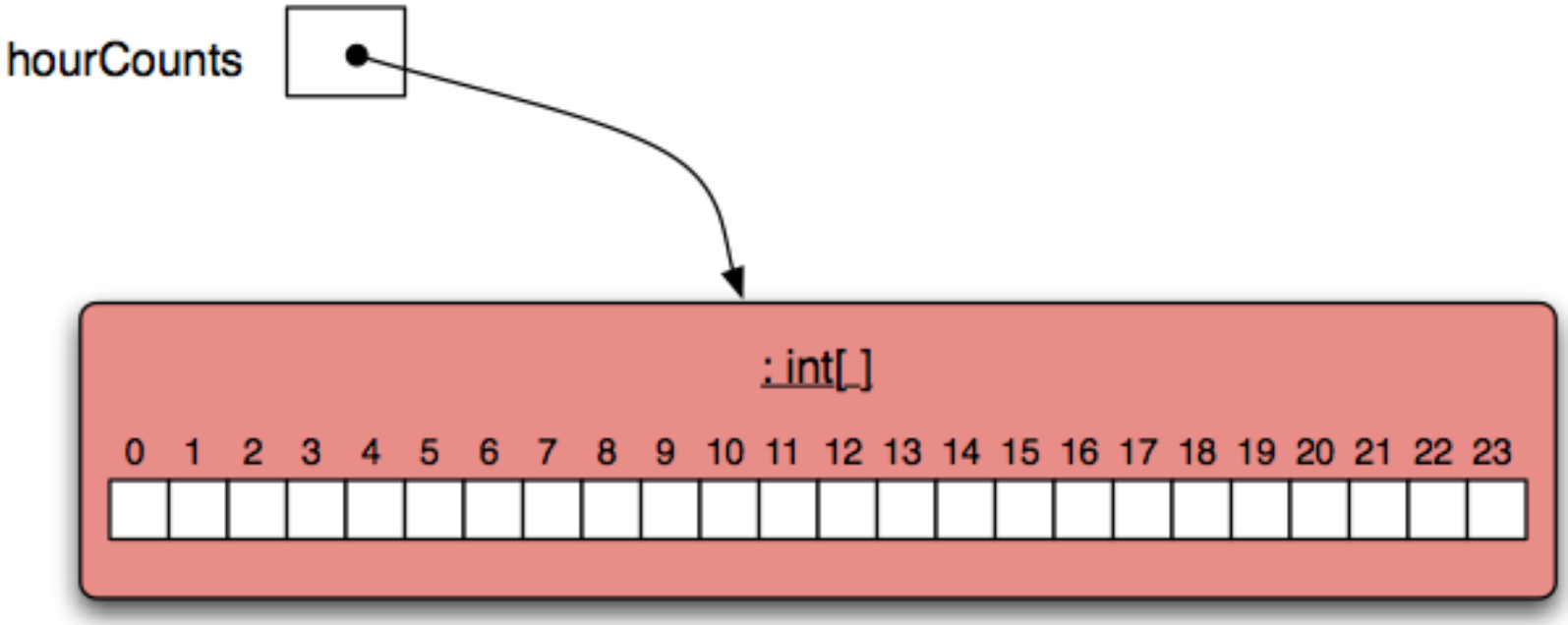
# ایجاد یک شی آرایه

```
public class LogAnalyzer
{
    private int[] hourCounts;
    private LogfileReader reader;

    public LogAnalyzer()
    {
        hourCounts = new int[24];
        reader = new LogfileReader();
    }
    ...
}
```

اعلان متغیر آرایه

ایجاد شی آرایه

# آرایه hourCounts



# استفاده از آرایه

- نماد **براکت** برای دسترسی به یک عنصر از آرایه مورد استفاده قرار می‌گیرد.
  - `hourCounts[...]`
- عناصر همانند متغیرهای معمولی استفاده می‌شوند.
  - در سمت چپ عملگر انتساب
    - `hourCounts[hour] = ...;`
  - در یک عبارت
    - `adjusted = hourCounts[hour] - 3;`
    - `hourCounts[hour]++;`

# استفاده از آرایه

```
int[] hourCounts;
hourCounts = new int[24];
hourcounts[1] = 0;
hourcounts[2] = 2;
...
```

اعلان، تعریف، مقداردهی

```
String[] names = {“Steve”, “John”};
```

مقداردهی هنگام اعلان (نیازی به مشخص کردن اندازه نیست)

```
hourcounts[2]++;
System.out.println(hourcounts[2]);
hourcounts[1] = 0;
```

استفاده

```
int n = hourcounts.length;
```

بدست آوردن طول آرایه (خصوصیت)

# حلقه `for`

- حلقه `for` معمولا برای تکرار به دفعات مشخصی مورد استفاده قرار می‌گیرد.
- اغلب با یک متغیر مورد استفاده قرار می‌گیرد که یک در هر تکرار به یک مقدار ثابت تغییر می‌کند.

# شبه کد حلقه for

**شکل کلی حلقه for**

```
for(initialization; condition; post-body action) {
    statements to be repeated
}
```

**حلقه while مشابه**

```
initialization;
while(condition) {
    statements to be repeated
    post-body action
}
```

# یک مثال در جاوا

```
for(int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

```
int hour = 0;
while(hour < hourCounts.length) {
    System.out.println(hour + ": " + hourCounts[hour]);
    hour++;
}
```

# یک مثال در جاوا

```
for(int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

```
int hour;
for(hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

# for با قدم‌های بزرگتر

```
// Print multiples of 3 that are below 40.
for(int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}
```

# پیمایش آرایه با for-each

```
for(int index=0; index < myArray.length; index++)
  System.out.println(myArray[index]);
```

```
for (int element : myArray)
   System.out.println(element);
```

# پیمایش آرایه با for-each

```
public int totalQuantity(Product[] products){
    int total = 0;
    for(Product p : products) {
        total += p.getQuantity();
    }
    return total;
}
```

# مرور

- آرایه‌ها می‌توانند زمانی که یک مجموعه با طول ثابت مورد نظر است مورد استفاده قرار گیرند.
- آرایه‌ها دارای نحو ویژه خود هستند.
- استفاده از حلقه `for` برای پیمایش.

# این درس

**مفاهیم**

- collections
- loops
- iterators
- arrays

**جاوا**

- ArrayList
- Iterator
- Array
- while
- for
- Index
- import