Objects and Containers

Weng Kai
2020 Fall

Primitive Data Types

Type	Storage	Min Value	Max Value
`byte`	8 bits	–128	127
`short`	16 bits	–32,768	32,767
`int`	32 bits	–2,147,483,648	2,147,483,647
`long`	64 bits	–9,223,372,036,854,775,808	9,223,372,036,854,775,807
`float`	32 bits	Approximately –3.4E+38 with 7 significant digits	Approximately 3.4E+38 with 7 significant digits
`double`	64 bits	Approximately –1.7E+308 with 15 significant digits	Approximately 1.7E+308 with 15 significant digits

Creating Objects

```java
int num = 42;
String name = new String("James Gosling");
```

- The first declaration creates a variable that holds an integer v

- The second declaration creates a String variable that holds a
  a String object. An object variable doesn't hold an object itself.

num	42

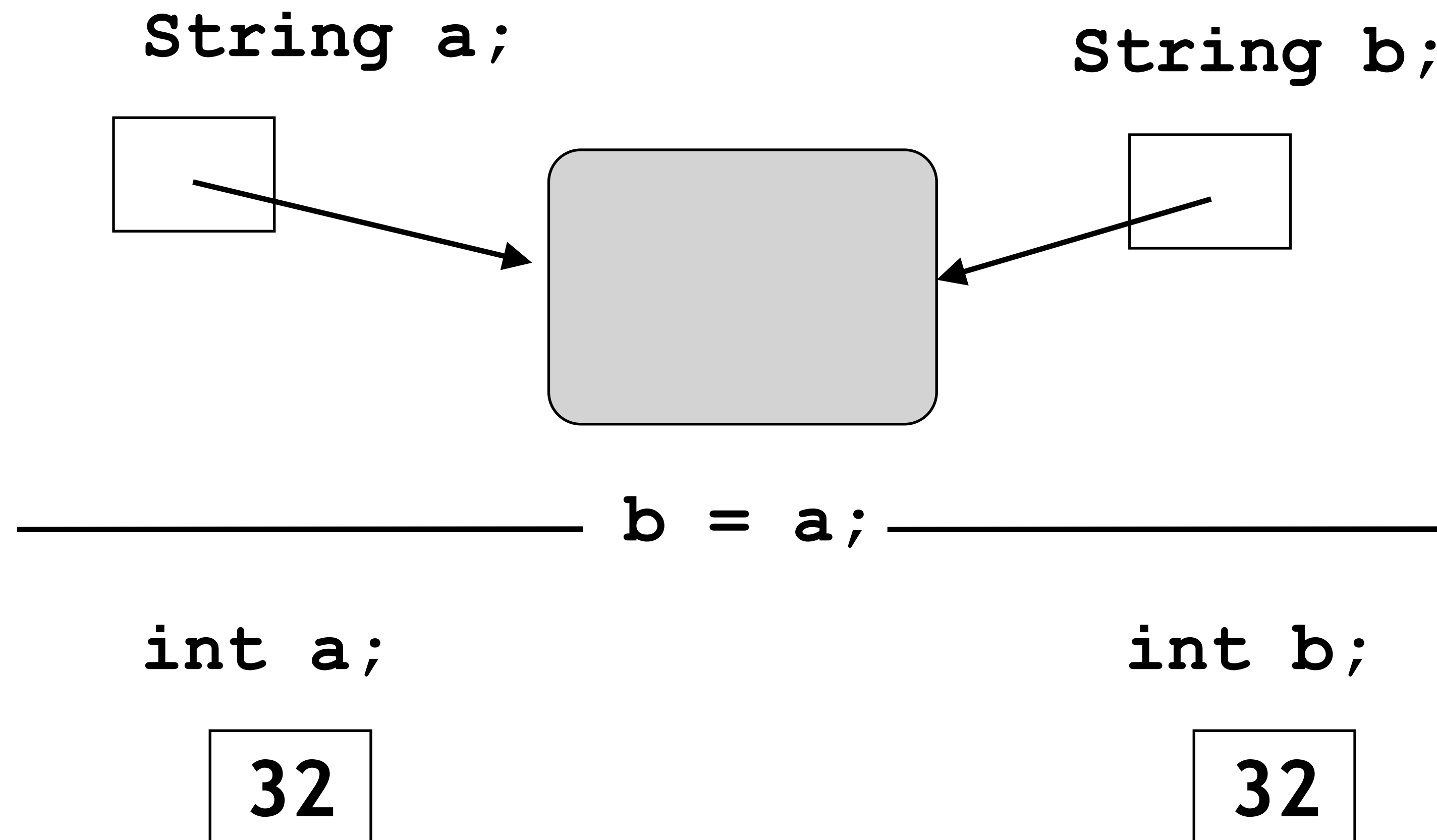| name	

"James Gosling"
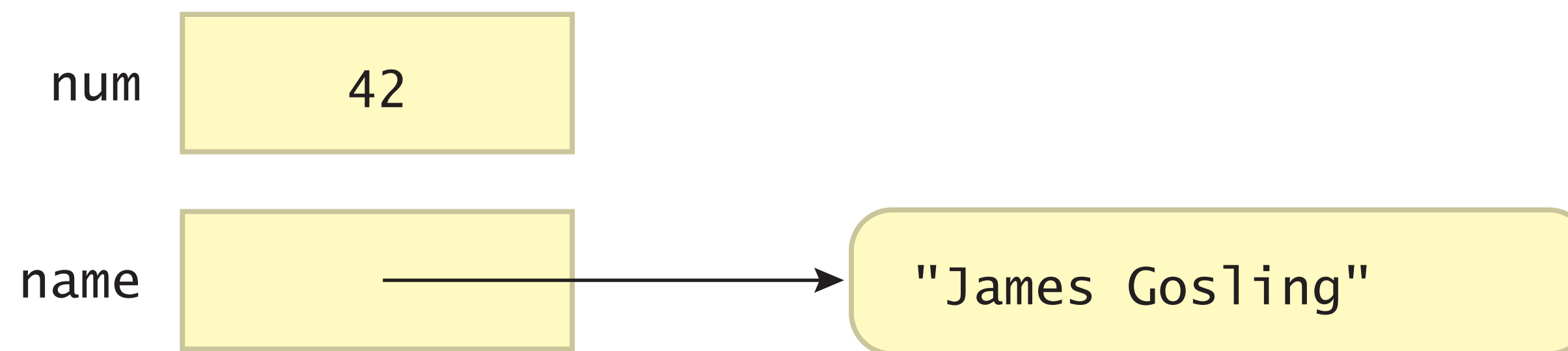
Object Variables

- Object variables are managers to objects.

```
String name1 = "Ada, Countess of Lovelace";
String name2 = "Grace Murray Hopper";
name2 = name1;
```

name1 ——→ "Ada, Countess of Lovelace"

name2 ——→ "Grace Murray Hopper"

name1 ——→ "Ada, Countess of Lovelace"

name2 ——→
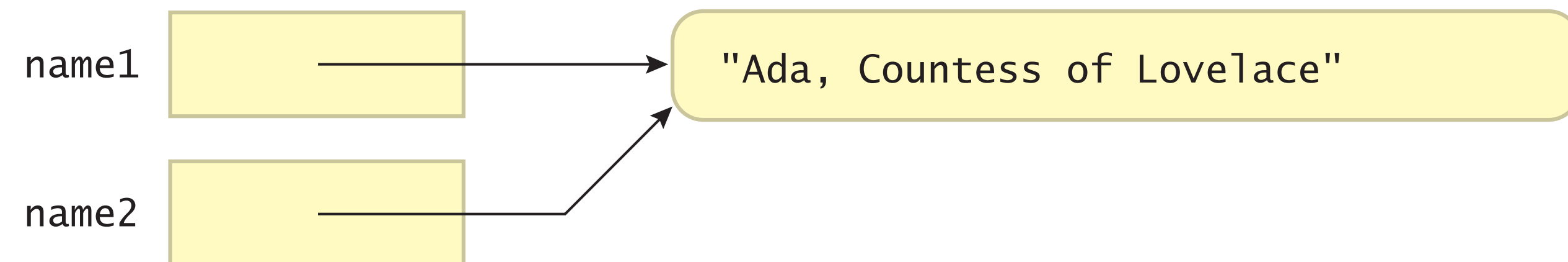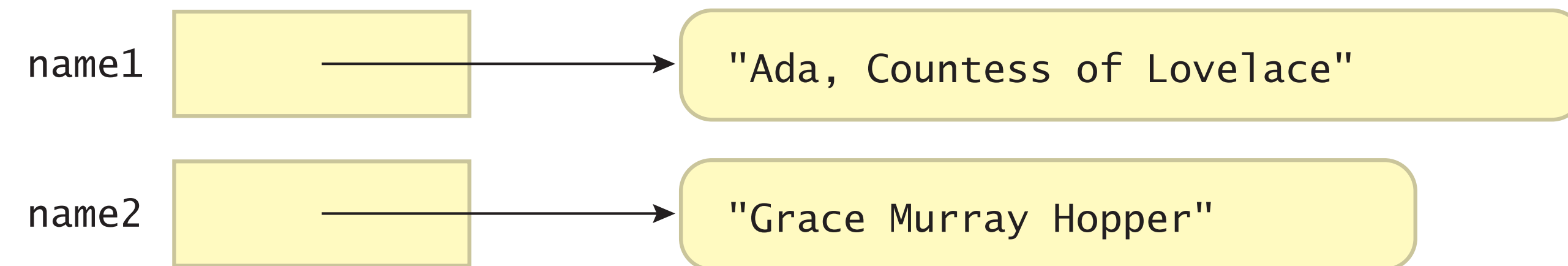
Primitive types vs. object types

`String a;`

`String b;`

`b = a;`

`int a;`

`int b;`

`32`

`32`

To Compare Two Strings

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

    ...

}
```

Same one?

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

    ...

}
```

Same Content?

• String should be compared by .equals()

:String

"Fred"

person1

:String

"Jill"

person2

person1 == person2 ?
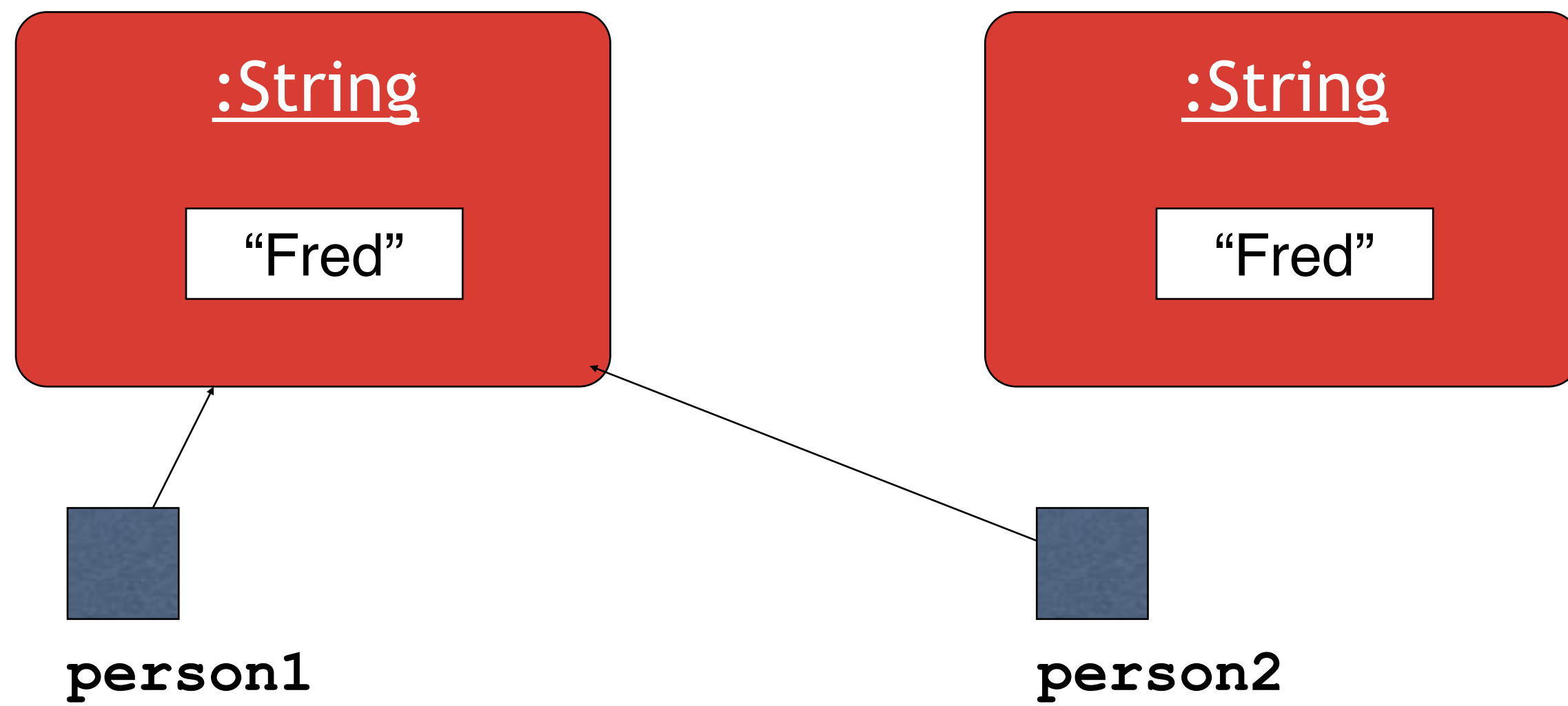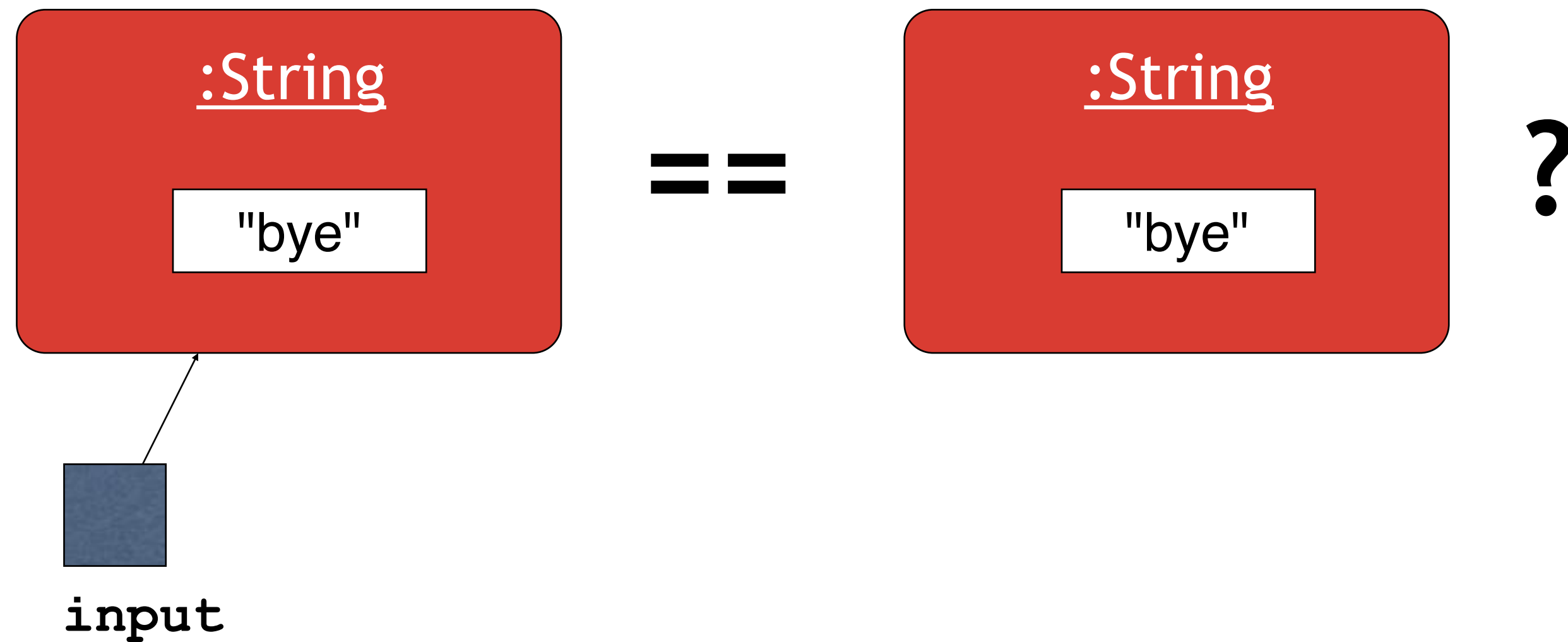
:String

"Fred"

person1

:String

"Fred"

person2

person1 == person2 ?

person1 == person2 ?

```
String input = in.next();
if(input == "bye") {
    ...
}
```

== for the same one

:String

"bye"

==

:String

"bye"

?

input

String Methods

```
int count = name.length();
```

- After an object has been instantiated, we use the dot operator to access its methods.

String Methods (1)

- String (String str)

  - Constructor: creates a new string object with the same characters as str.

- int length ()

  - Returns the number of characters in this string.

- int compareTo (String str)

  - Returns an integer indicating if this string is lexically before (a negative return value), equal to (a zero return value), or lexically after (a positive return value), the string str.

- boolean equals (String str)

  - Returns true if this string contains the same characters as str (including case) and false otherwise.

- boolean equalsIgnoreCase (String str)

  - Returns true if this string contains the same characters as str (without regard to case) and false otherwise.

String Methods (2)

- char charAt (int index)

  - Returns the character at the specified index.

- String concat (String str)

  - Returns a new string consisting of this string concatenated with str.

- String replace (char oldChar, char newChar)

  - Returns a new string that is identical with this string except that every occurrence of oldChar is replaced by newChar.

- String substring (int offset, int endIndex)

  - Returns a new string that is a subset of this string starting at index offset and extending through endIndex-1.

String Methods (3)

- String toLowerCase ()

  - Returns a new string identical to this string except all uppercase letters are converted to their lowercase equivalent.

- String toUpperCase ()

  - Returns a new string identical to this string except all lowercase letters are converted to their uppercase equivalent.

- String trim()

  - Returns a new string without the space before or after the string.

String Object is Immutable

- All methods can not alter the string but generate a new string object

- Assign a new string literal to a string variable is to make the variable manage the new string object

```
String s = "";
for ( int i=0; i<100; i++ ) {
    s += i;
}
```

- This code segment will create and destroy string object 100 times!

StringBuffer

```java
StringBuffer sb = new StringBuffer();
for ( int i=0; i<100; i++ ) {
    sb.append(i);
}
String s = sb.toString();
```

Lab 1

- PTA 7-1, 7-2

array

- Consider the case where a programmer needs to keep track of a number of people within an organization. So far, our initial attempt will be to create a specific variable for each user. This might look like,

```
int name1 = 101;

int name2 = 232;

int name3 = 231;
```

the array way

```
int[] names = new int[4];

names[0] = 101;

names[1] = 232;

names[2] = 231;

names[3] = 0;
```

array

- An array is a special type of collection that can store a fixed number of elements.

```
int[] a;

int a[];
```

array

- Array is a data structure which hold multiple variables of the same data type

  - All elements are in the same type

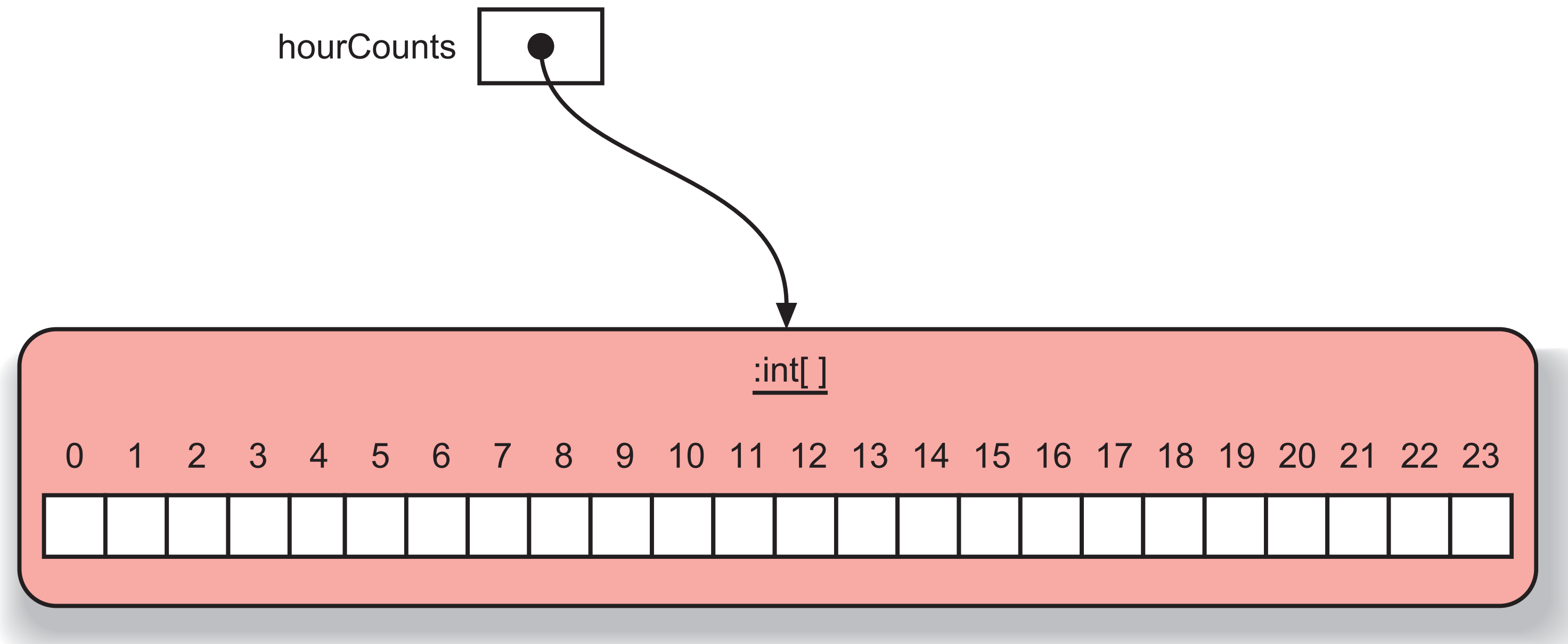| a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

to define an array

- <type>[] <name>= new <type>[number_of_elements];

  - int[] grades= new int[100];

  - float[] averages = new int[20];

- *number_of_elements* must be an integer

create array object

- hourCounts = new int[24];

int[] a= new int[10]

- An array of int

- 10 elements: a[0],a[1],…,a[9]

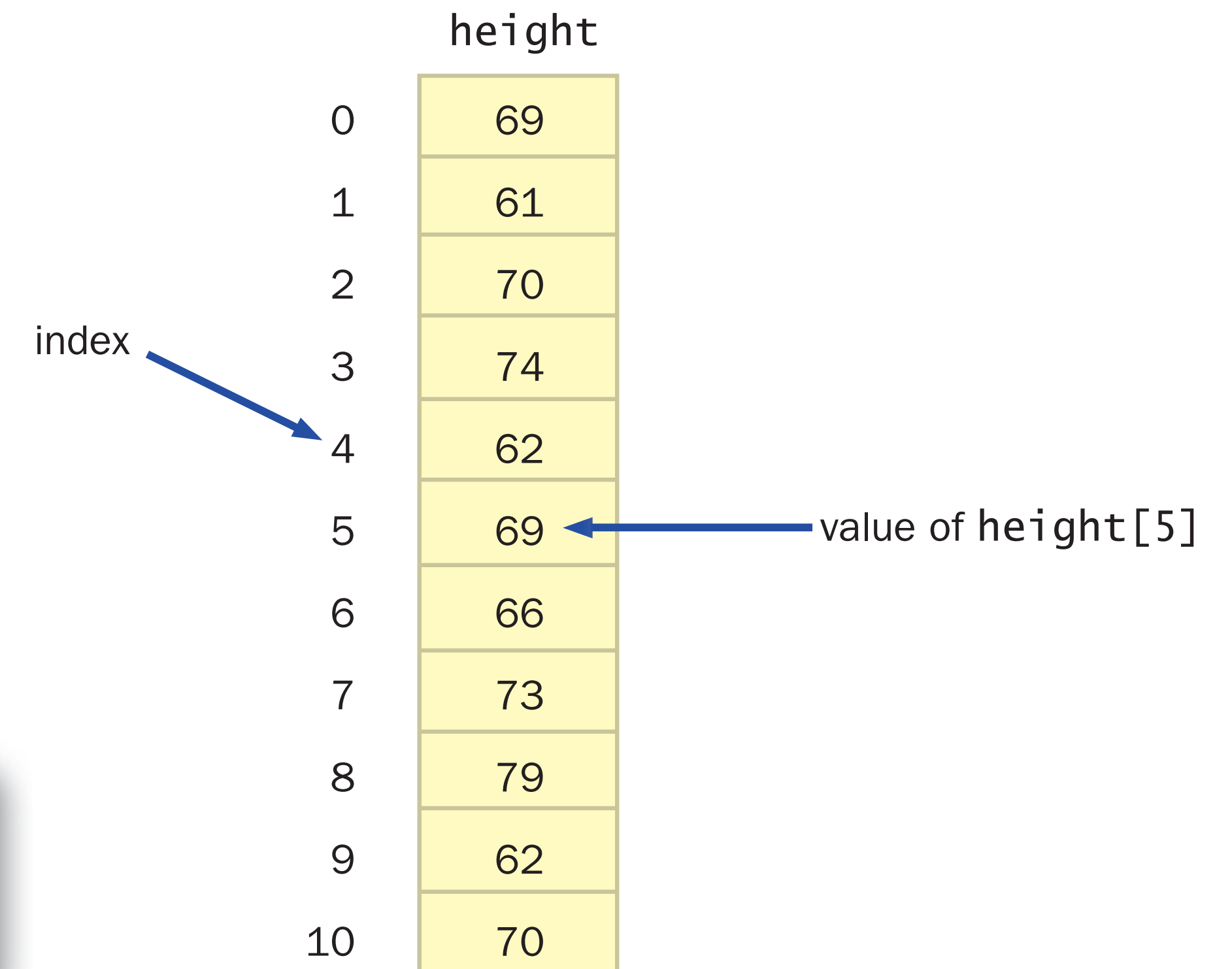| a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

- Each element is an int variable

- To be right or left value:

  - a[2] = a[1]+6;

element of an array

- each element is a variable of the type

- index or subscript starts from 0

  - grades[0]

  - grades[99]

  - average[5]

KEY CONCEPT

An array of size N is indexed from 0 to N–1.

height

index	value
0	69
1	61
2	70
3	74
4	62
5	69
6	66
7	73
8	79
9	62
10	70

index

value of height[5]

Question

height

index	height
0	69
1	61
2	70
3	74
4	62
5	69
6	66
7	73
8	79
9	62
10	70

index

value of `height[5]`

- Based on the array shown at left, what are each of the following?

1. height[1]

2. height[2] + height[5]

3. height[2 + 5]

4. the value stored at index 8

5. the fourth value

6. height.length

length

- every array has an internal member length which tells the size of the array

  - the number of elements

Valid Subscript Values

- The compiler does check to see if you use a valid subscript, wherever it is used as right or left value.

- It causes problems once the program runs, possibly corrupting data or code, possibly causing the program to abort.

- It is Java's responsibility to make sure that your program uses only valid subscript values, in the run-time.

KEY CONCEPT

Bounds checking ensures that an index used to refer to an array element is in range.

```
for ( i=0; i<100; ++i )

    sum += grade[i];

                            better to be:

for ( i=0; i<grade.length; ++i )

    sum += grade[i];
```

```java
public class ArrayLength {
    public static void main(String[] args) {
        int[] a = new int[(int)(Math.random()*10)];
        for ( int i=0; i<a.length; ++i )
            a[i] = (int)(Math.random()*100);
        for ( int i=0; i<a.length; ++i )
            System.out.println(a[i]);
    }
}
```

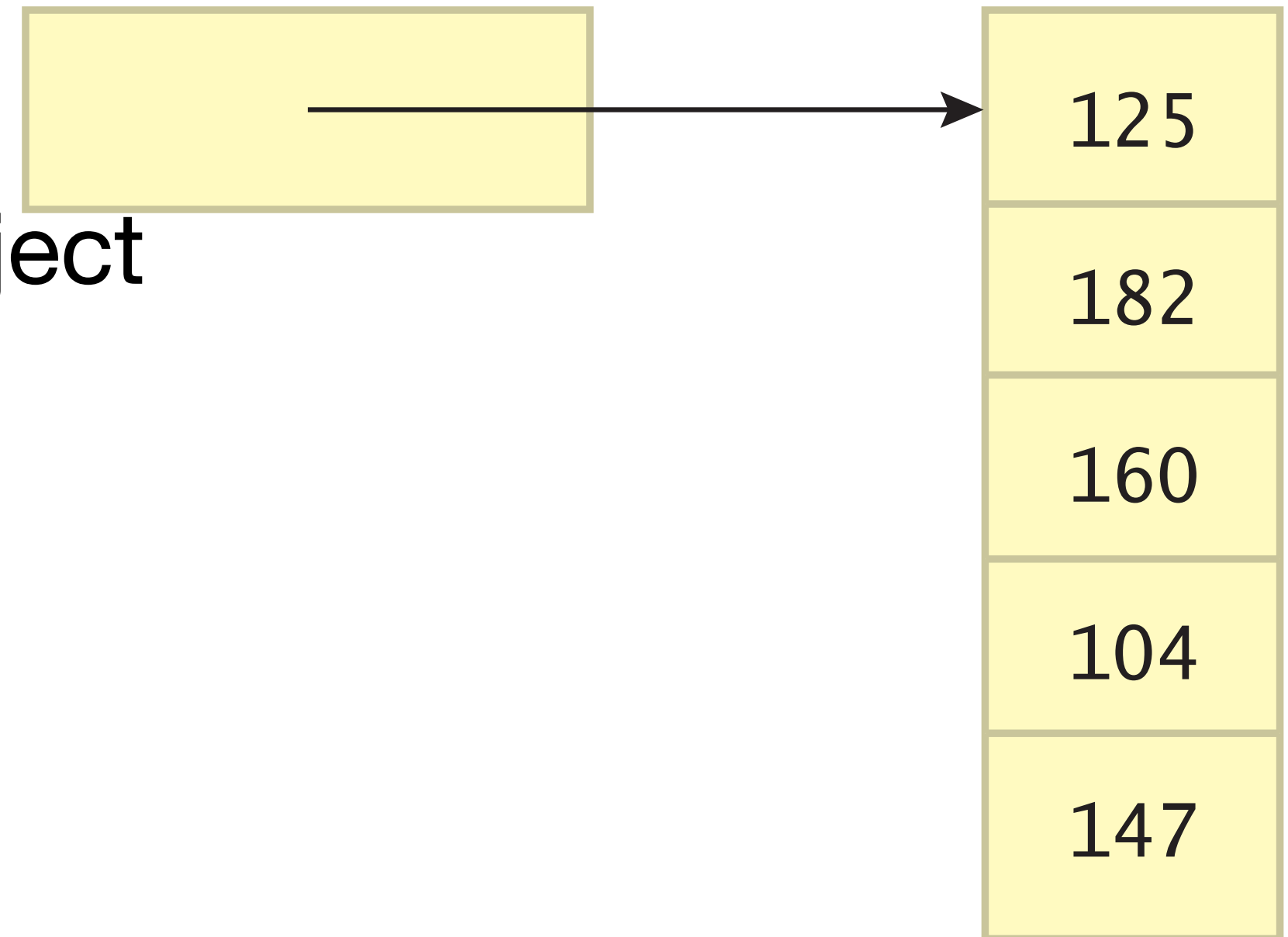# initializer Lists

- int[] a = {1,2,3,4,5};

for-each

```
for ( <type> <variable>: <array> ) {

    variable...

}
```

```java
public class ForArray {
  public static void main(String[] args) {
    int[] array = new int[(int)(Math.random()*10)+1];
    for ( int i=0; i<array.length; ++i )
      array[i] = (int)(Math.random()*100);
    for ( int value: array )
      System.out.println(value);
  }
}
```

Array Variable

weight

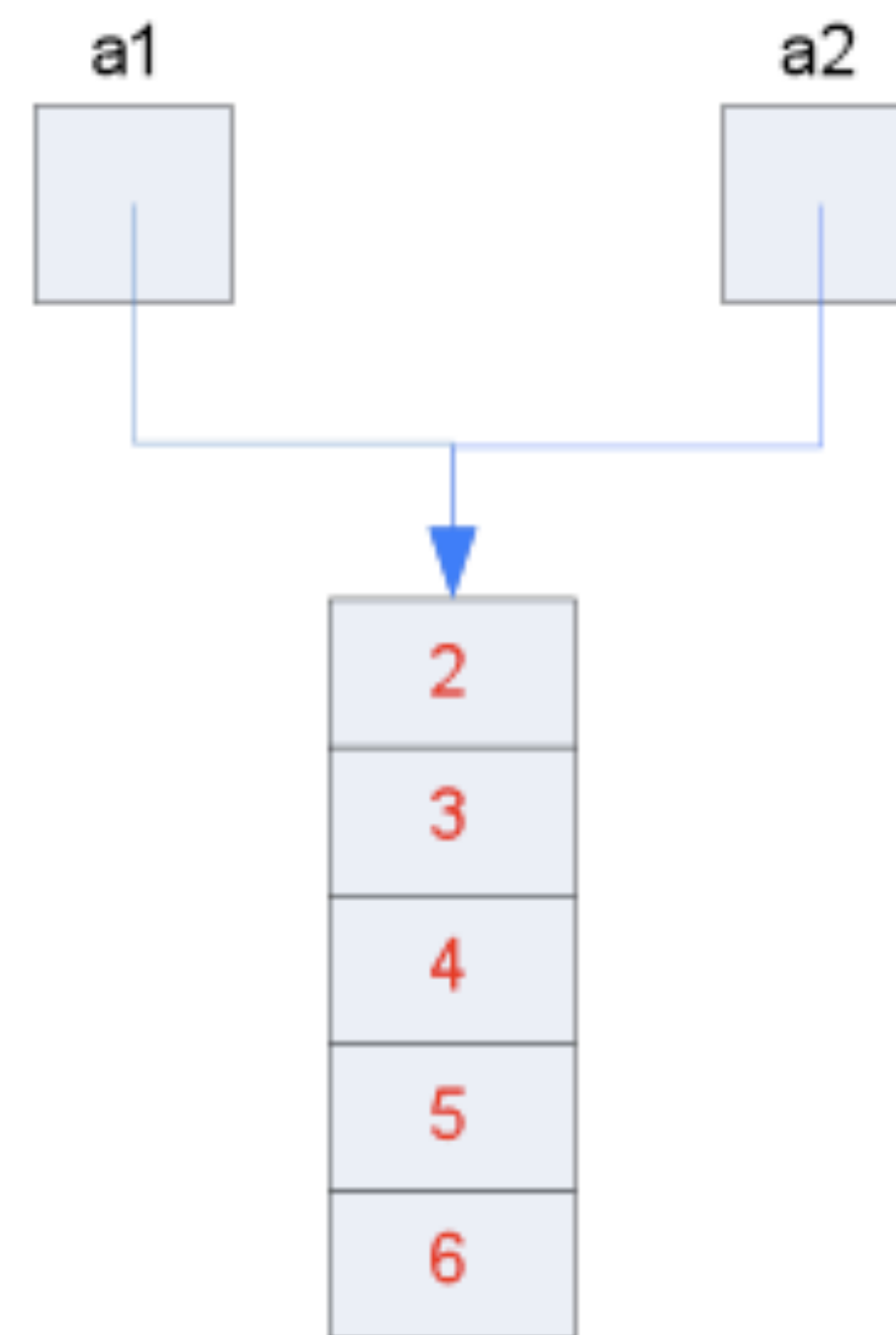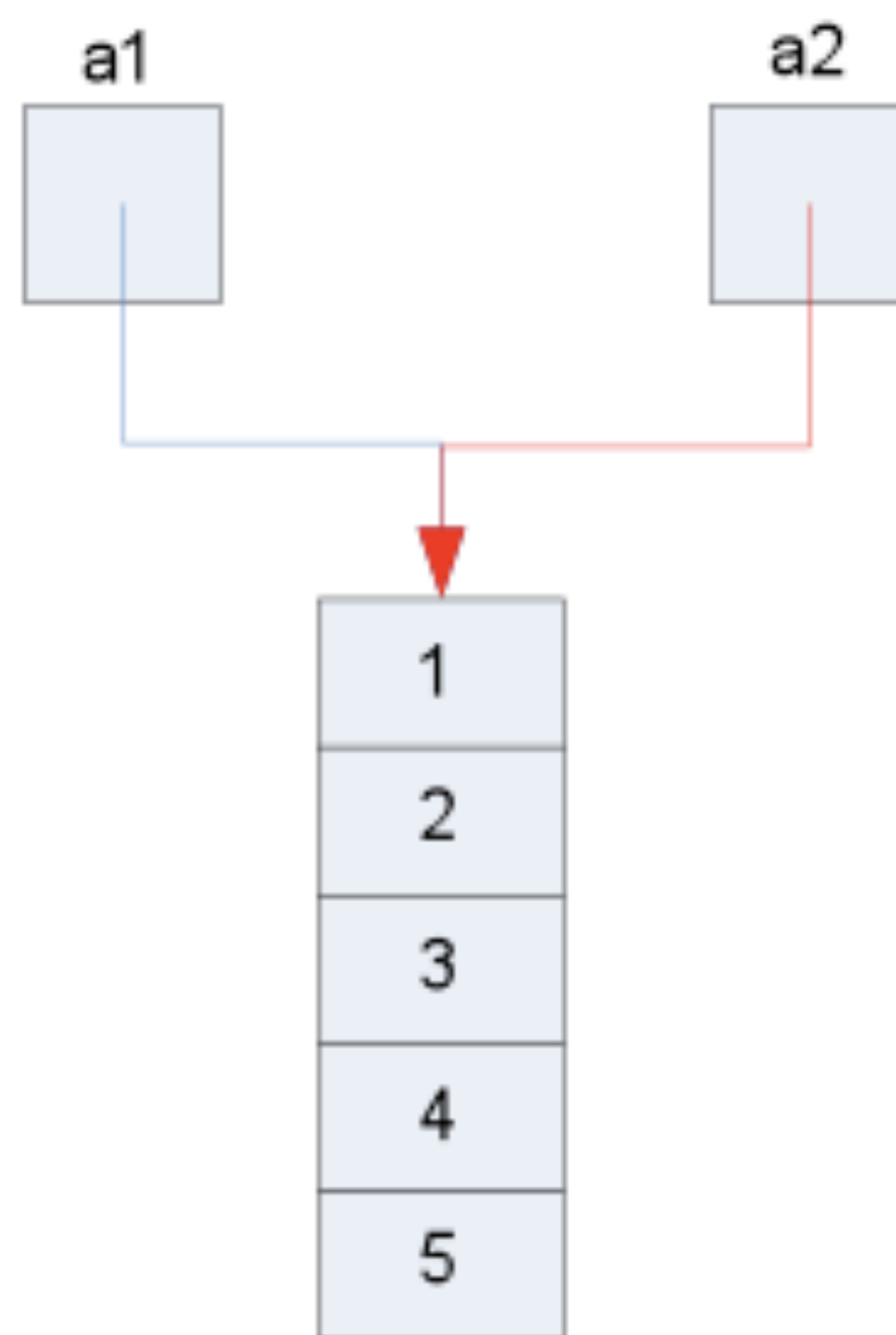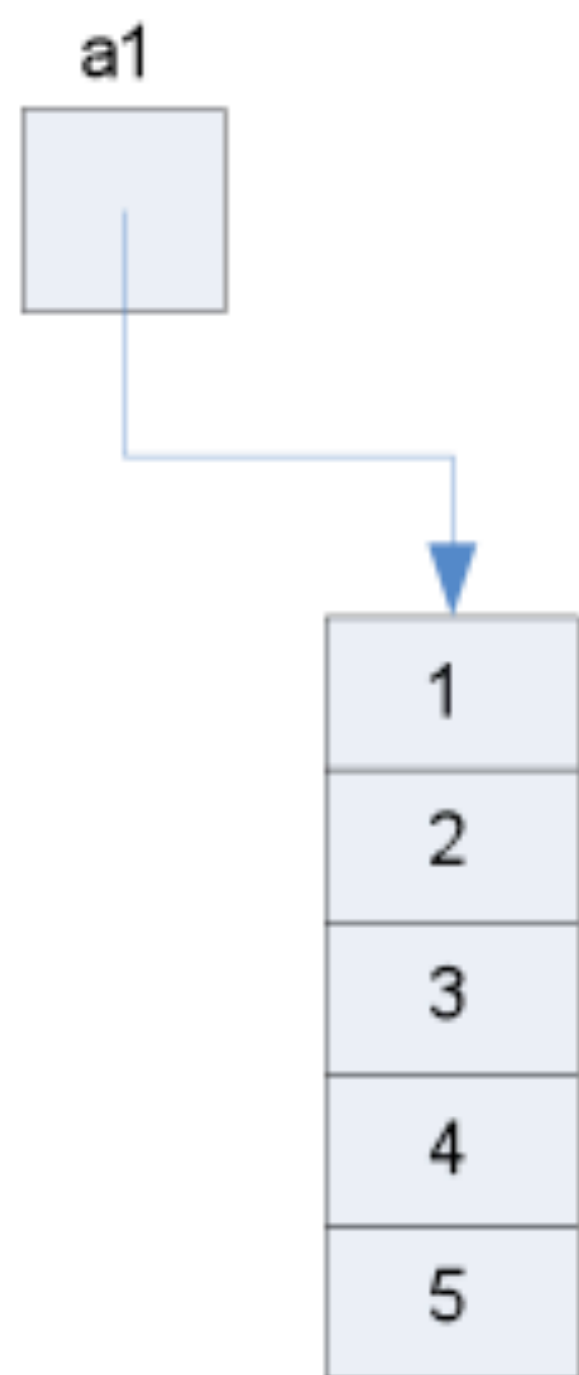- Array variable is the manager to an array object

| |
|---|
| 125 |
| 182 |
| 160 |
| 104 |
| 147 |

assignment of two arrays
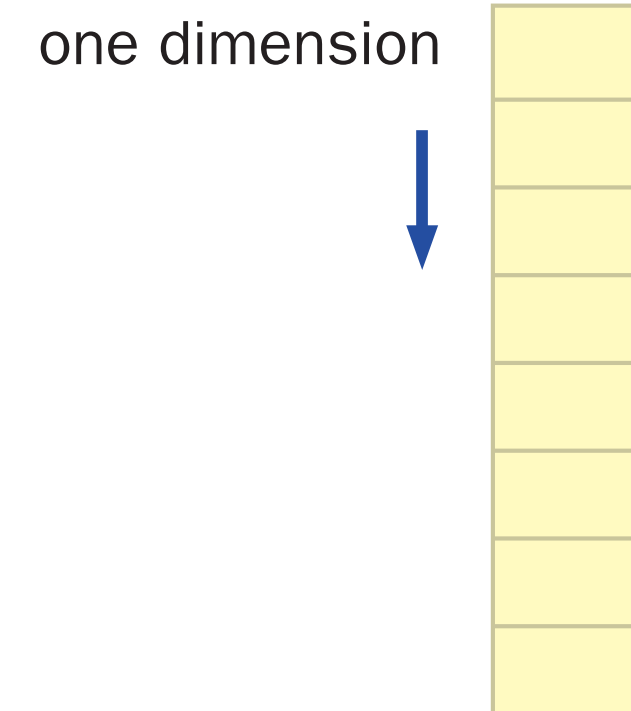
- int[] a = new int[10];

- a[0] = 5;

- int[] b = a;

- b[0] = 16;

- System.out.println(a[0]);

```java
public class ArrayAssignment {
  public static void main(String[] args) {
    int[] a1 = {1,2,3,4,5};
    int[] a2 = a1;
    for ( int i=0; i<a2.length; ++i )
      a2[i] ++;
    for ( int v : a1 )
      System.out.println(v);
  }
}
```

matrix

- array of arrays

  - int[][] a = new int[3][4];

    - defines a two dimensional array

    - a is an array of int[3];

one dimension

two dimensions
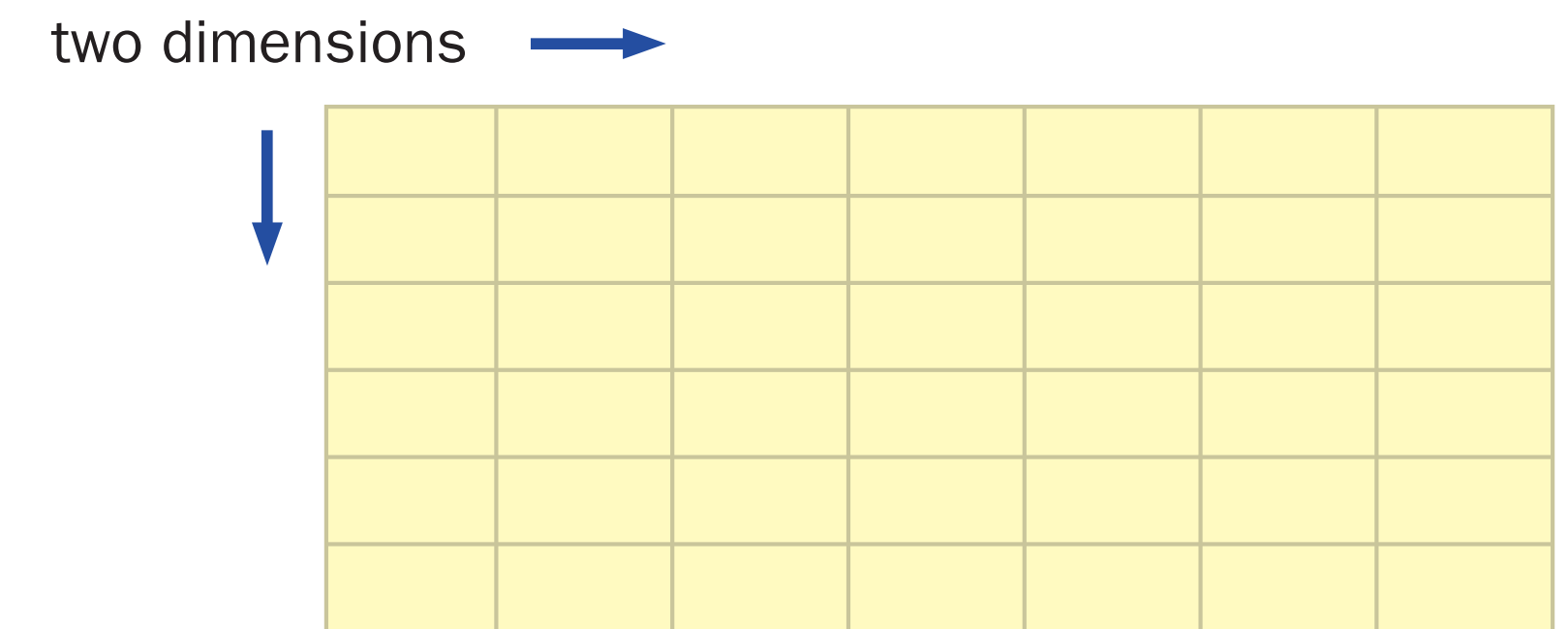
```java
public class TwoDArray
{
    //----------------------------------------------------------
    // Creates a 2D array of integers, fills it with increasing
    // integer values, then prints them out.
    //----------------------------------------------------------
    public static void main(String[] args)
    {
        int[][] table = new int[5][10];
        // Load the table with values
        for (int row=0; row < table.length; row++)
            for (int col=0; col < table[row].length; col++)
                table[row][col] = row * 10 + col;
        // Print the table
        for (int row=0; row < table.length; row++)
        {
            for (int col=0; col < table[row].length; col++)
                System.out.print(table[row][col] + "\t");
            System.out.println();
        }
    }
}
```

Lab 2

- PTA 7-3, 7-4

Container

A personal notebook

- It allows notes to be stored.

- It has no limit on the number of notes it can store.

- It will show individual notes.

- It will tell us how many notes it is currently storing.

Collection

- Collection objects are objects that can store an arbitrary number of other objects.

library classes

- Libraries typically contain many hundreds or thousands of different classes.

- Java calls its libraries packages.

  - import

- Notebook class uses ArrayList class in the java.util package

NoteBook.java

```java
import java.util.ArrayList;

public class Notebook
{
   private ArrayList<String> notes;

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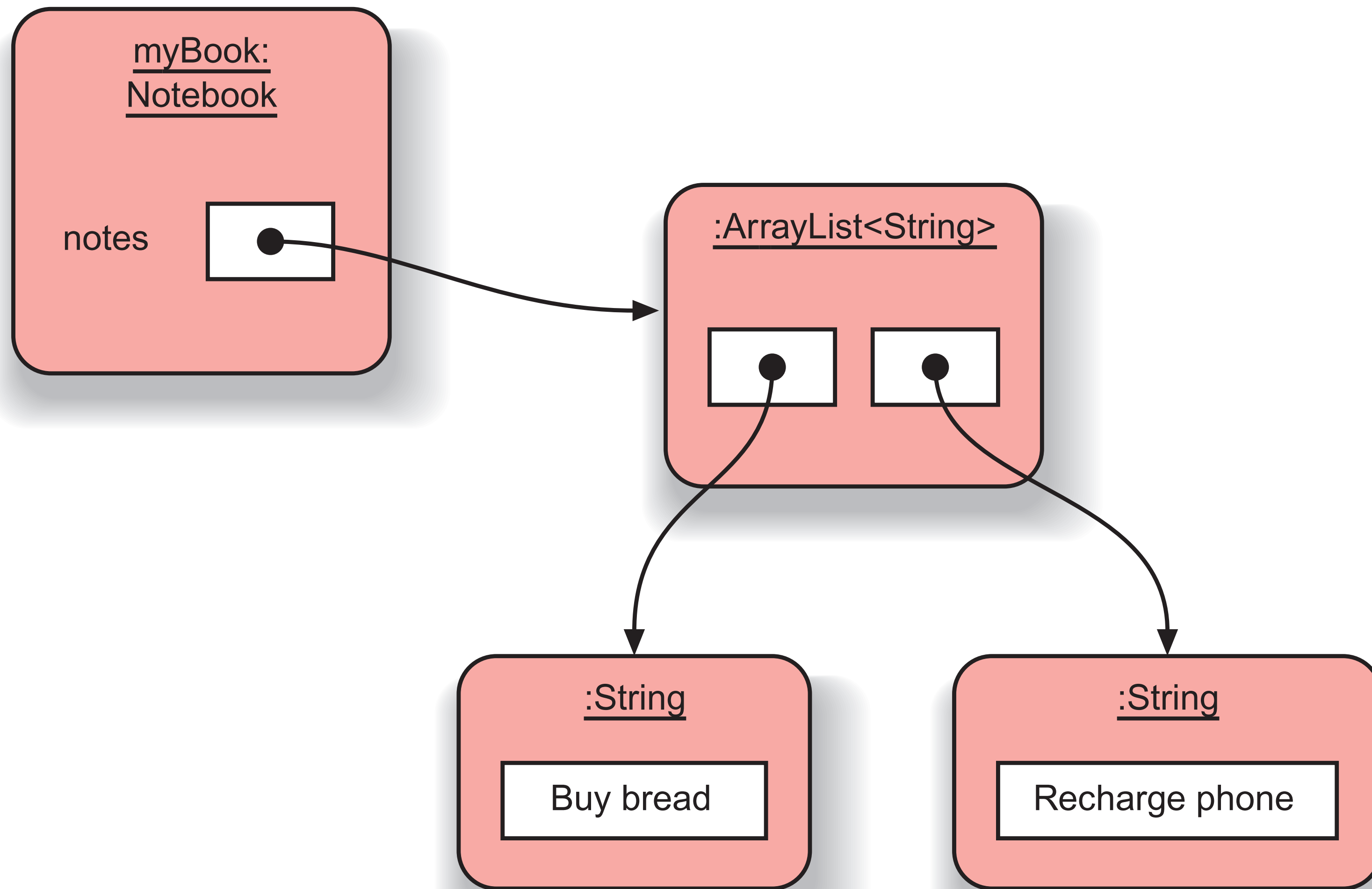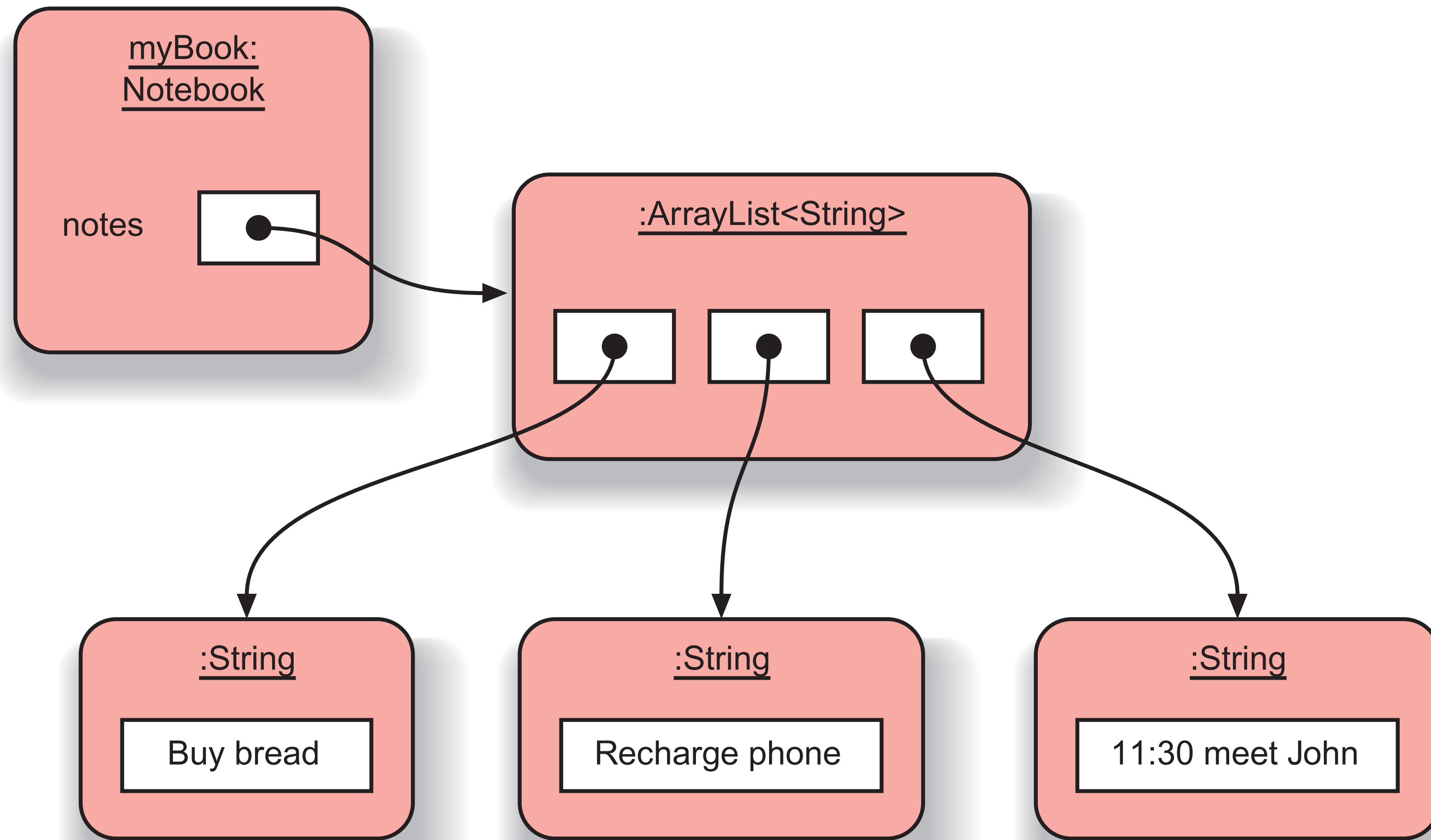
generic classes

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

- Have to specify two types: the type of the collection itself (here: ArrayList) and the type of the elements that we plan to store in the collection (here: String)

# Object structure

Object structure

ArrayList

- It is able to increase its internal capacity as required: as more items are added, it simply makes enough room for them.

- It keeps its own private count of how many items it is currently storing. Its size method returns the number of objects currently stored in it.

- It maintains the order of items you insert into it. You can later retrieve them in the same order.
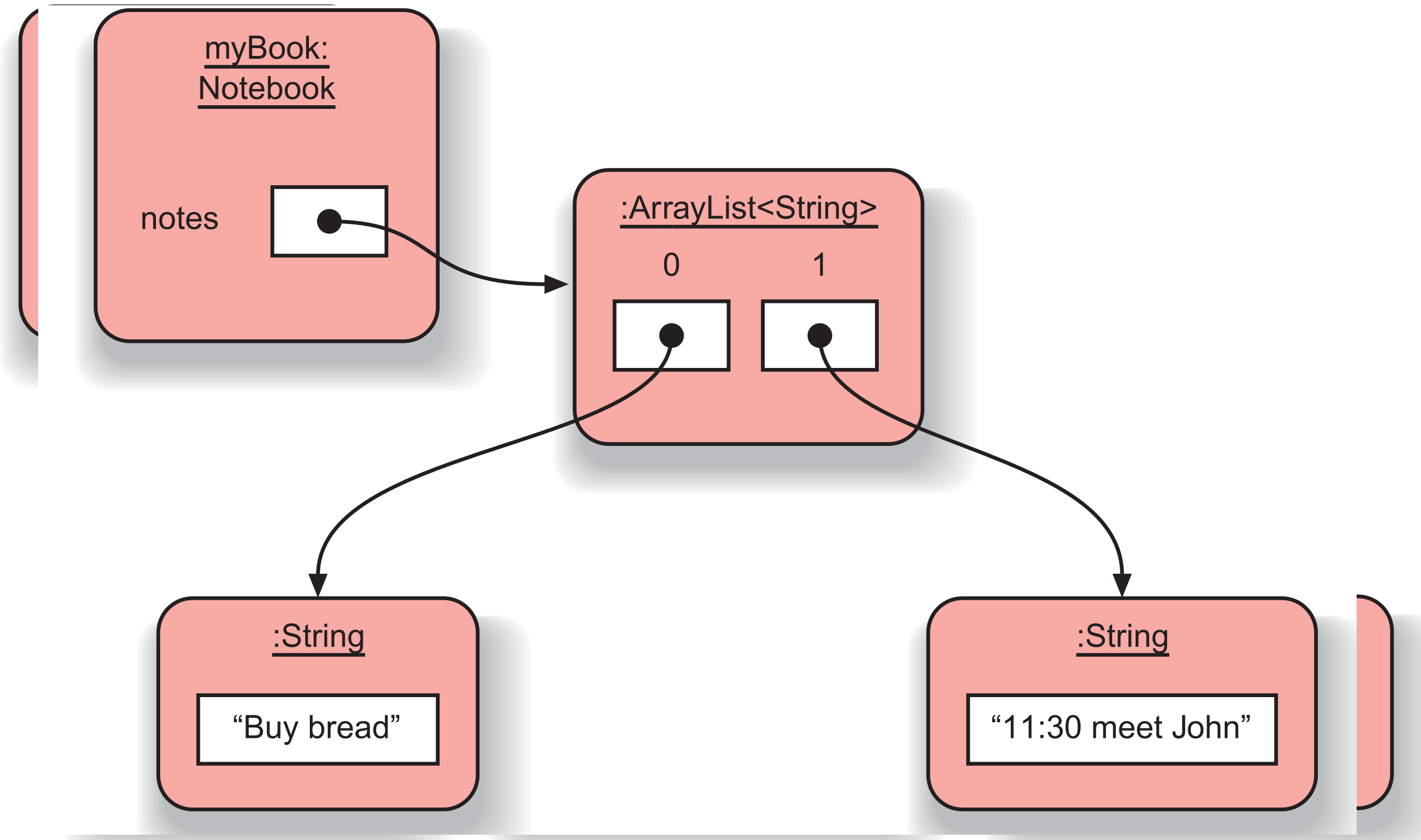
put in

- `public boolean add(E o);`

- `public void add(int index, E element);`

index
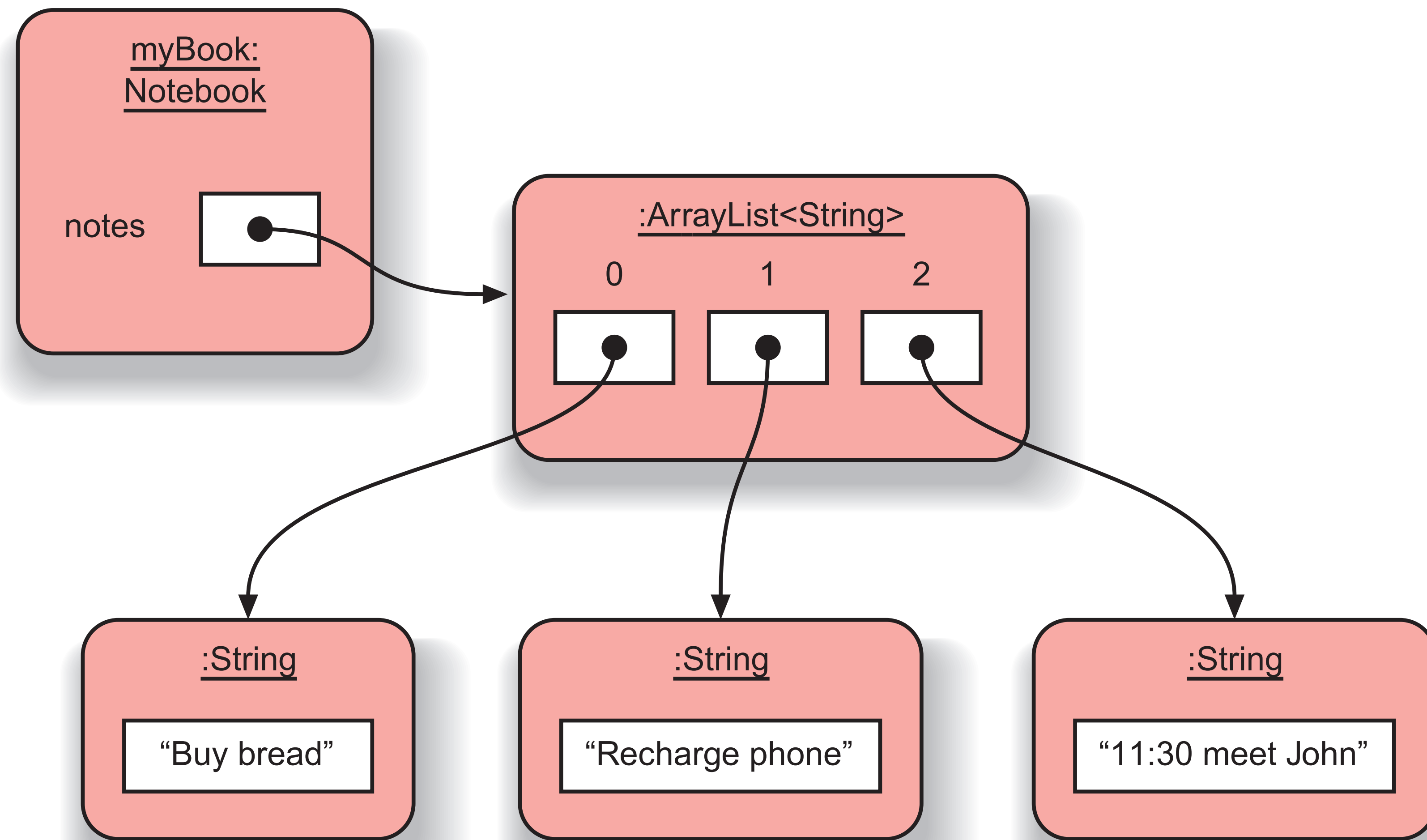
Removing

- `public boolean remove(Object o);`

- `public E remove(int index);`

Processing a whole collection

- for-each loop

for (ElementType element : collection) {

    loop body

}

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

54

while loop

```
int index = 0;
while(index < notes.size()) {
    System.out.println(notes.get(index));
    index++;
}
```

`public E get(int index);`

while loop

```
int index = 0;
boolean found = false;
while (index < notes.size() && !found) {
    String note = notes.get(index);
    if (note.contains(searchString)) {
        found = true;
    } else {
        index++;
    }
}
```

Iterator

- An iterator is an object that provides functionality to iterate over all elements of a collection.

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

How an iterator acts

1. Ask a container to hand you an **Iterator** using a method called **iterator( )**. This **Iterator** will be ready to return the first element in the sequence on your first call to its **next( )** method.

2. Get the next object in the sequence with **next( )**.

3. See if there *are* any more objects in the sequence with **hasNext( )**.

4. Remove the last element returned by the iterator with **remove( )**.

Maps

- Maps are collections that contain pairs of values.

- Pairs consist of a <u>key</u> and a <u>value</u>.

- Lookup works by supplying a key, and retrieving a value.

- An example: a telephone book.

Using maps

- A map with Strings as keys and values

:HashMap

"Charles Nguyen"	"(531) 9392 4587"
"Lisa Jones"	"(402) 4536 4674"
"William H. Smith"	"(998) 5488 0123"

Using maps

```java
HashMap <String, String> phoneBook = new HashMap<String, String>();

phoneBook.put("Charles Nguyen", "(531) 9392 4587");

phoneBook.put("Lisa Jones", "(402) 4536 4674");

phoneBook.put("William H. Smith", "(998) 5488 0123");

String phoneNumber = phoneBook.get("Lisa Jones");

System.out.println(phoneNumber);
```

Using sets

```
import java.util.HashSet;
import java.util.Iterator;
...
HashSet<String> mySet = new HashSet<String>();
mySet.add("one");
mySet.add("two");
mySet.add("three");
Iterator<String> it = mySet.iterator();
while(it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}
```

Compare
this to
ArrayList
code!

Lab 3

- A score data processing program reads two types of text lines

  - \<sid\> \<name\>, as 3190101234,Kim

  - \<sid\> \<course initials\> \<mark\>, as 3190101234,Java,95

- And prints a table of the marks as:

  - sid, name, \<course name1\>, \<course name2\>…,average

  - 3190101234, Kim, 95, ,95

  - 3190101235, John, , 86, 86

Cheat Sheet for Lab 3

- line.split(",");

- TreeMap/TreeSet sorted map key/sorted set

- if (score == null) {