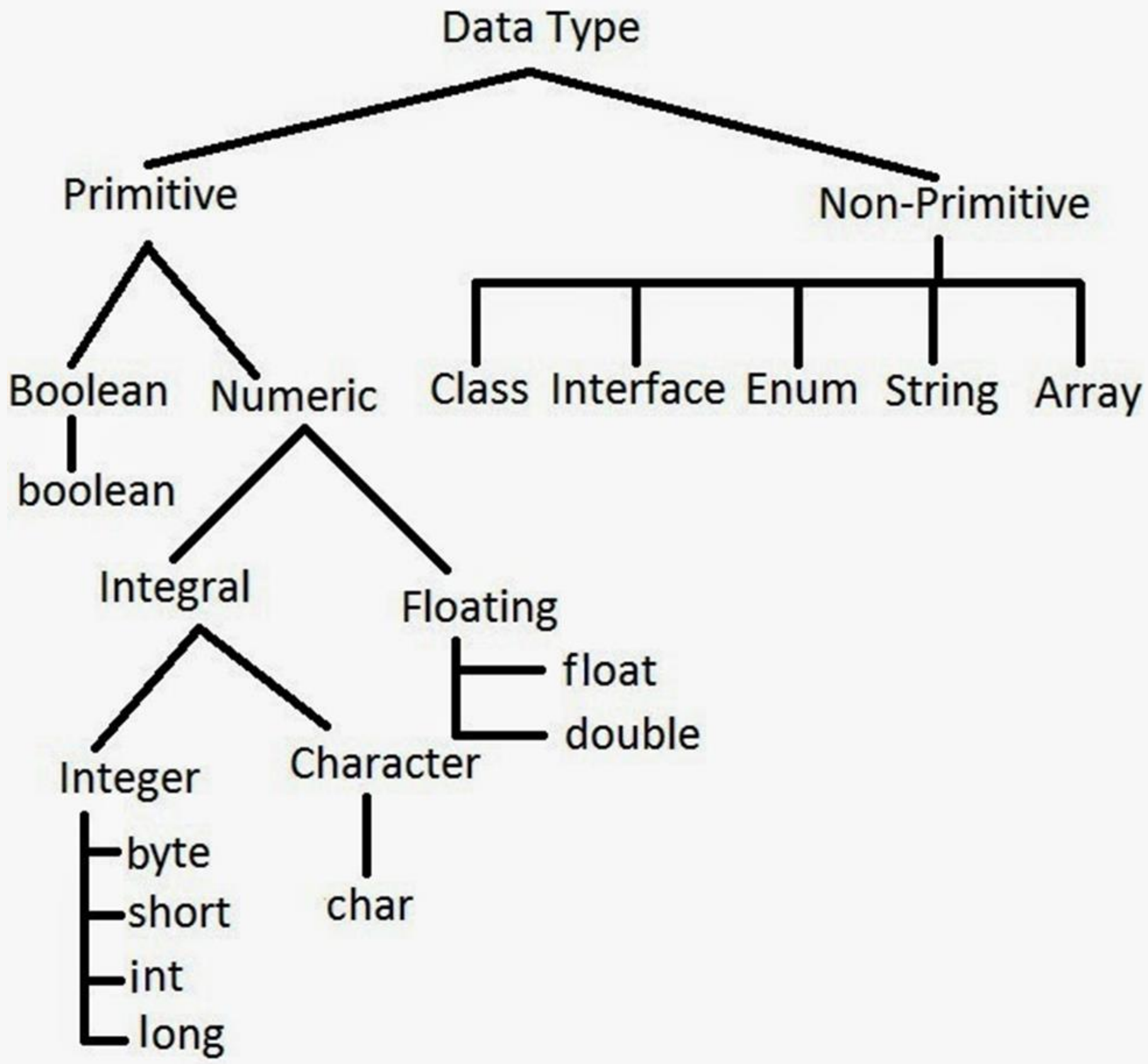
CS 24 AP Computer Science A Review

Week 6: Array and ArrayList

DR. ERIC CHOU
IEEE SENIOR MEMBER

SECTION 1

Reference Data Type



Data Type

Primitive

Non-Primitive

Boolean

Numeric

Class

Interface

Enum

String

Array

boolean

Integral

Floating

float

double

Integer

Character

byte

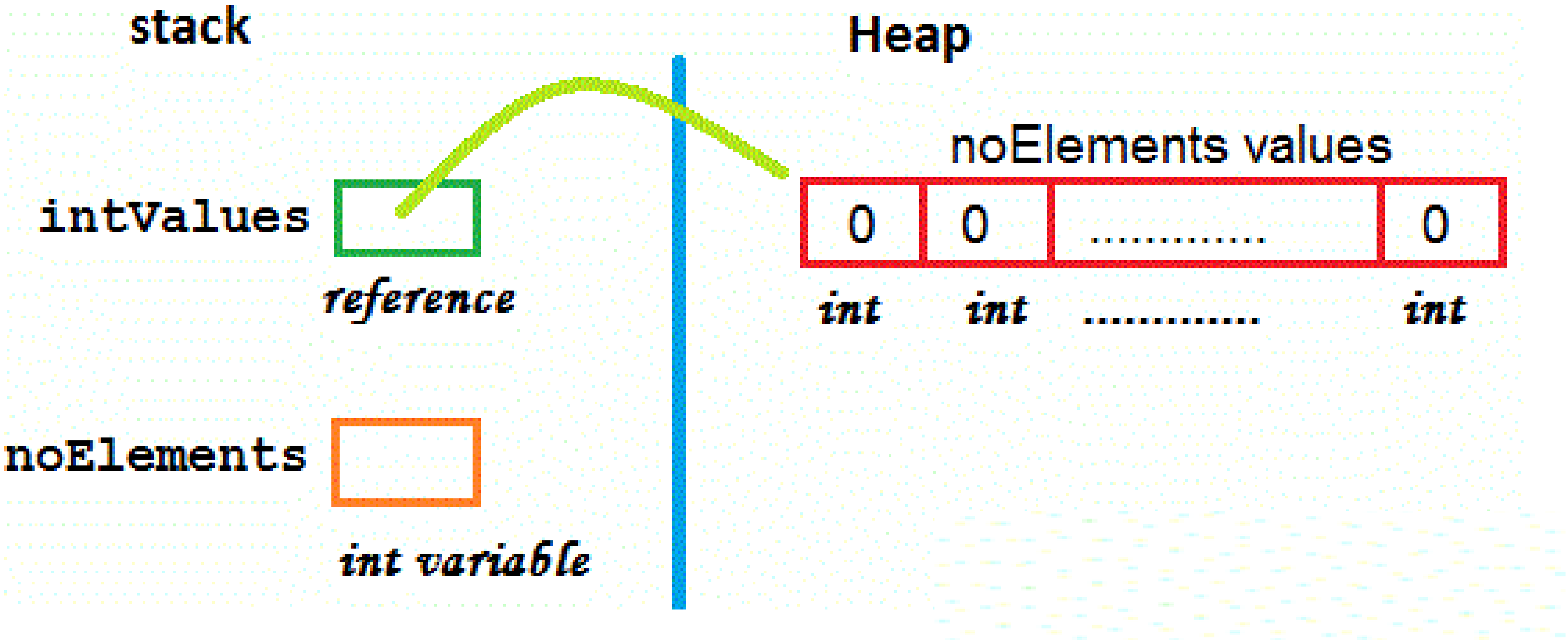
short

int

long

char

Reference Date Type



Three Important Complex Data Structures

There are five basic data structures (ADT) that are tested on the AP Computer Science A Exam:

- String (Covered in Week 2)
- The array (of one-dimensional array)
- The 2-D array
- The ArrayList
- Class (Structure, Data Record)

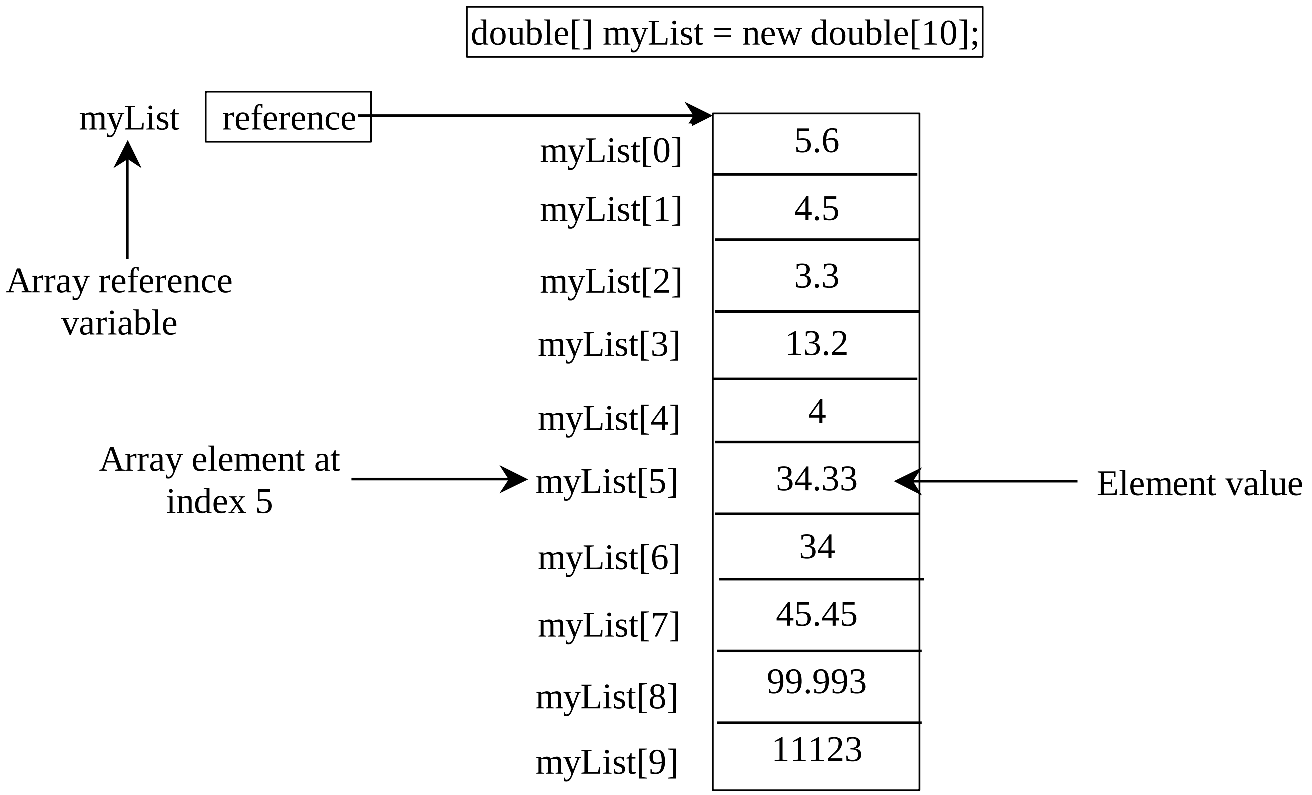
Situation	Recommendation	Justification
A program helps an elevator know what floor it is on.	Array	The number of floors is fixed. The elevator can go to any floor by knowing what number it is.
Facebook keeps track of how many friends you have.	ArrayList	The number of friends you have on Facebook may increase or decrease. You are allowed to add or remove anyone at any time regardless of where they are in the list.
Your program is going to simulate chess, Candy Crush, or 2048	2-D Array	Each of these games can be simulated on either a square or rectangular grid in which the row and column are used to find out what is in each cell.
A cell phone keeps track of text messages	ArrayList	The number of text message on a cell phone can increase or decrease. You can even delete all of the messages.
A program keeps track of classes you have each period of the school day.	Array	The number of class periods in the school day is fixed. Each period is assigned a value

SECTION 2

Array

Introducing Arrays

- Array is a data structure that represents a collection of the same types of data.



```
double[] myList = new double[10];
```

myList reference

Array reference variable

myList[0]	5.6
myList[1]	4.5
myList[2]	3.3
myList[3]	13.2
myList[4]	4
myList[5]	34.33
myList[6]	34
myList[7]	45.45
myList[8]	99.993
myList[9]	11123

Array element at index 5

Element value

Pre-defined Array

- **Pre-defined Array:**

```
int[] intArray = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
```

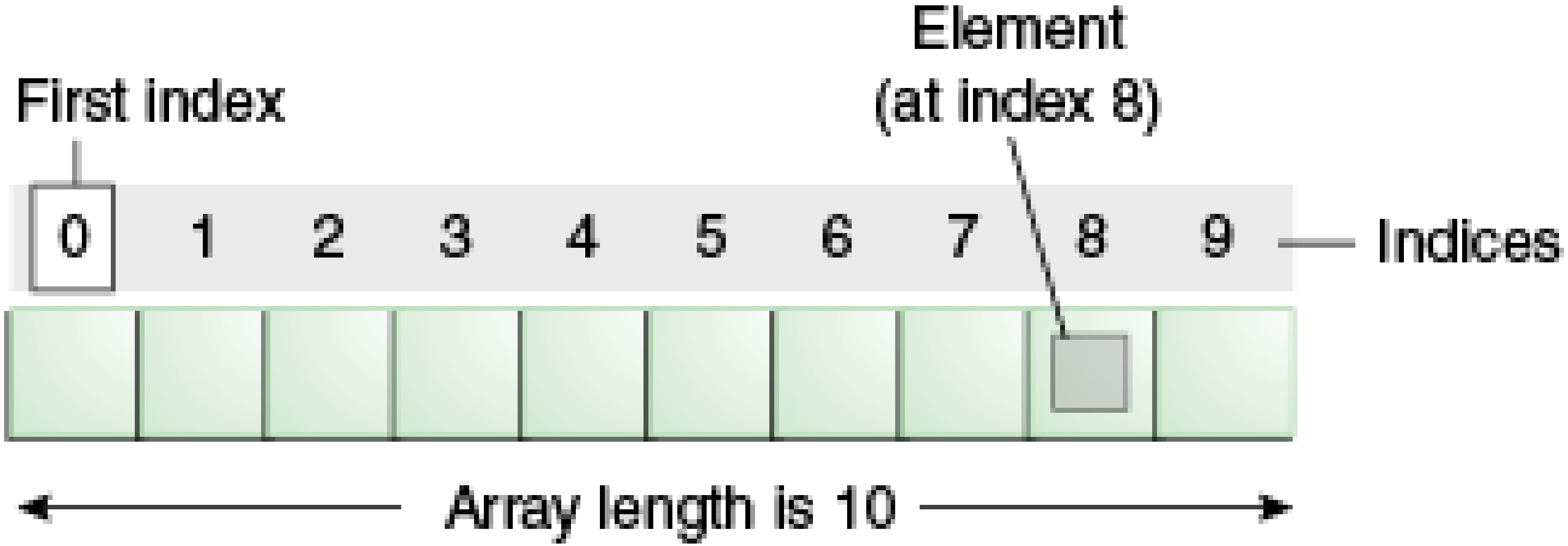
- **Array Declaration with array memory allocation:**

```
int[] intArray2 = new int[10];
```

- **Array declaration and data assignment with anonymous array:**

```
int[] intArray3;
        intArray3 = new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
```

array.length



The **new** Operator

- An object of a class is named or declared by a variable of the class type:

```
ClassName classVar;
```

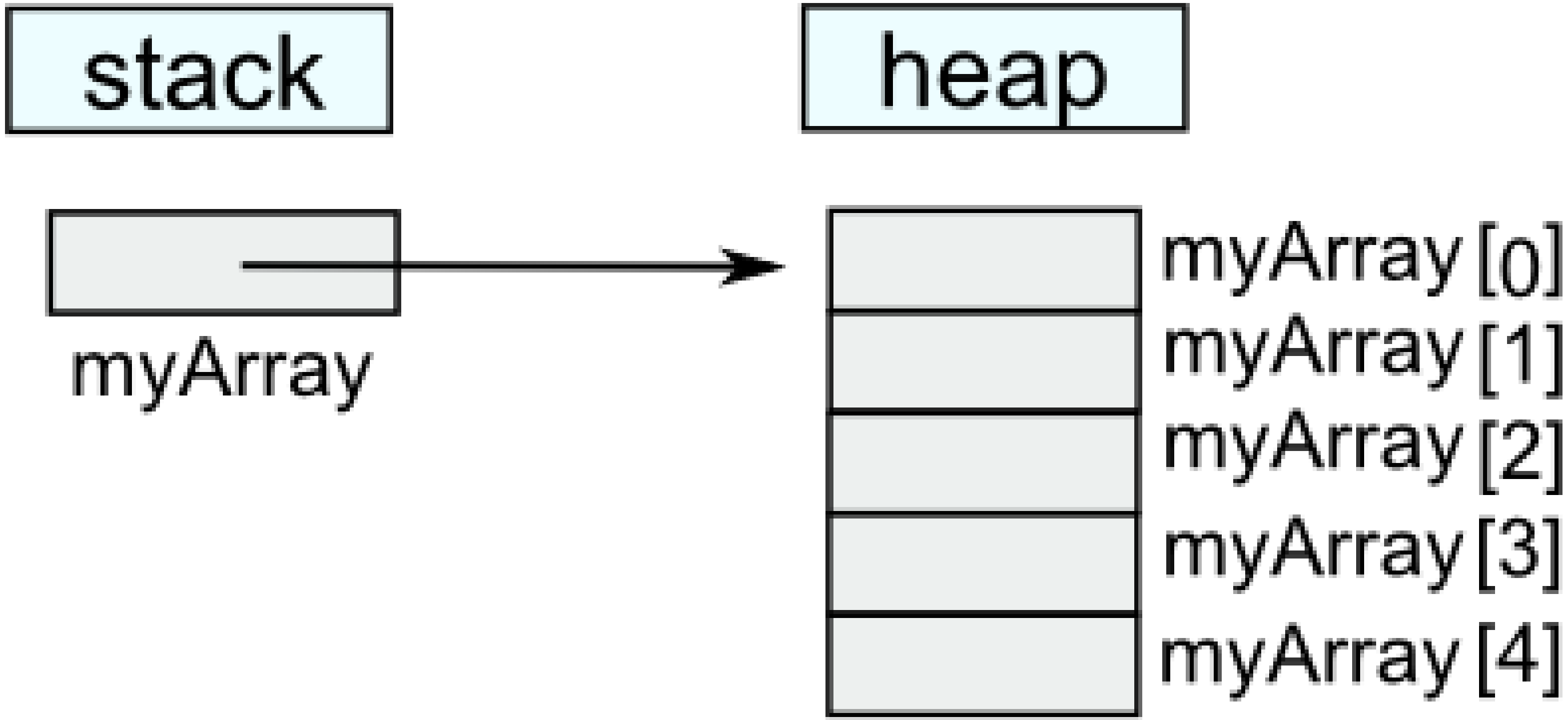
- The new operator must when be used to create the object and associate it with its variable:

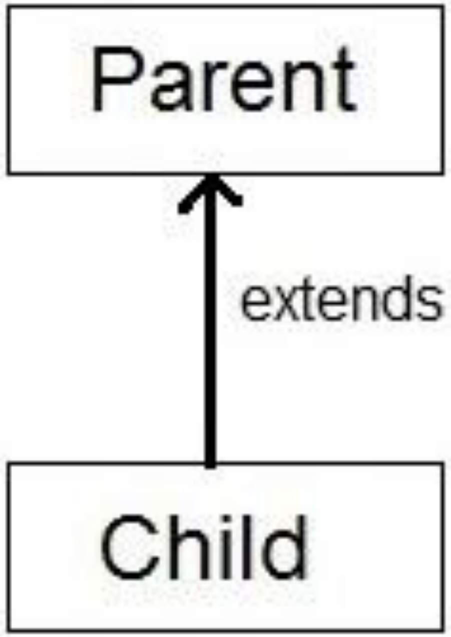
```
classVar = new ClassName();
```

- These can be combined as follows:

```
className classVar = new  ClassName();
```

new Operator





```
Parent p = new Child( );
```

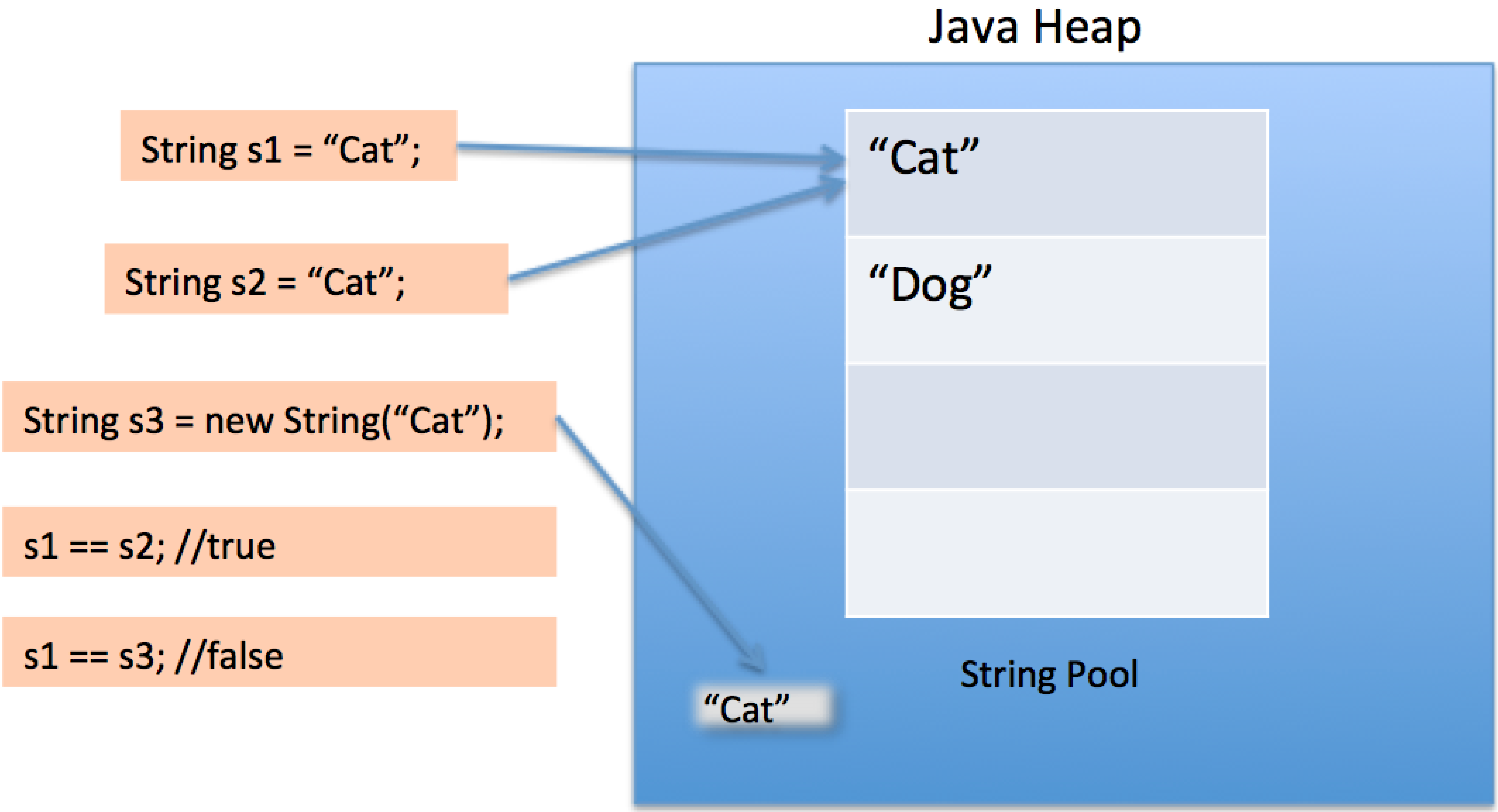
Upcasting

Child c ~~= new~~ Parent( );

Compile time error

```
Child c = ( Child ) new Parent( );
```

Downcasting but throws ClassCastException at runtime.



Java Heap

```
String s1 = “Cat”;

String s2 = “Cat”;

String s3 = new String(“Cat”);

s1 == s2; //true

s1 == s3; //false
```

“Cat”

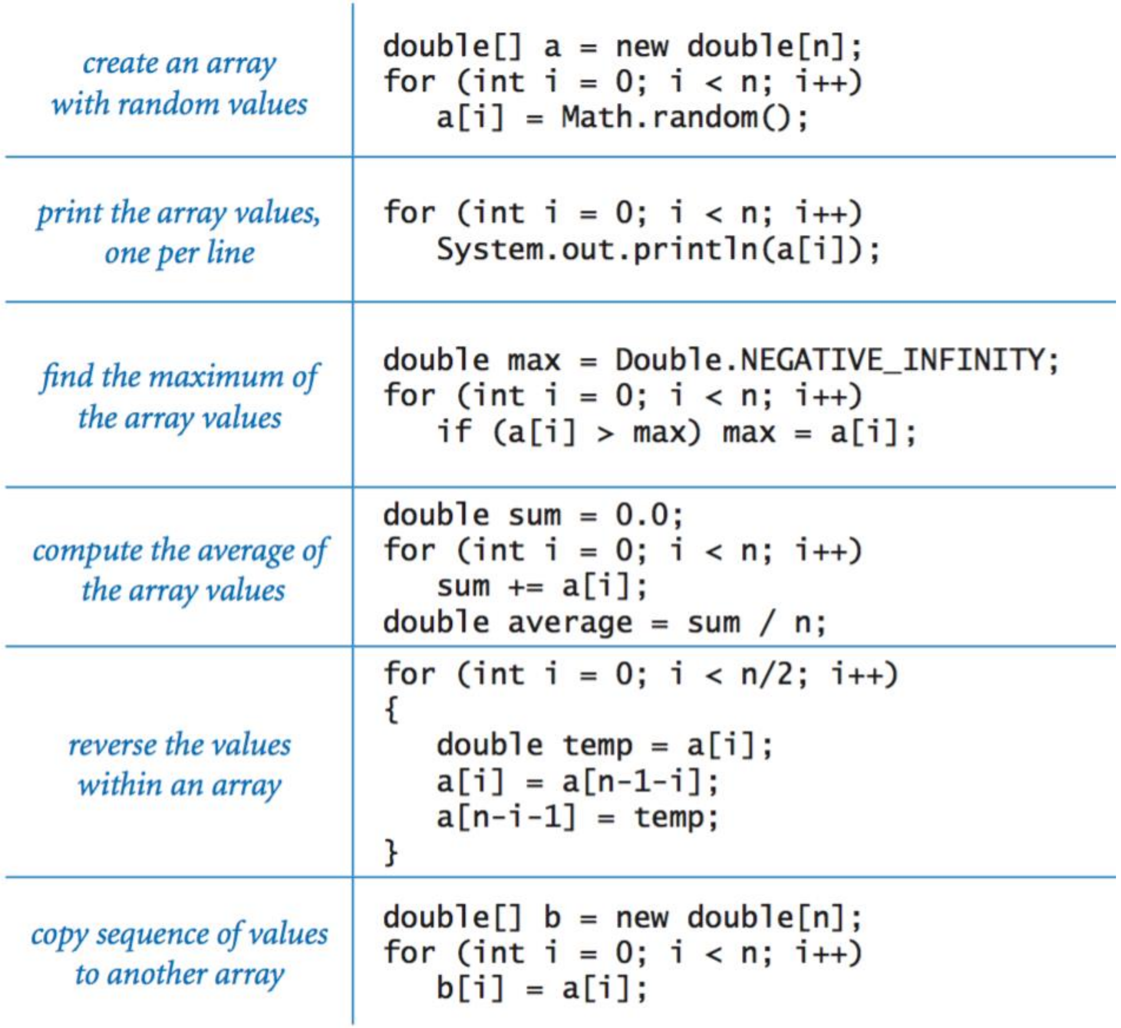
“Dog”

String Pool

“Cat”

Basic Array Class

BASIC APPLICATION AND INDEXING



create an array with random values	`double[] a = new double[n];`
	`for (int i = 0; i < n; i++)`
	`   a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < n; i++)`
	`   System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < n; i++)`
	`   if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < n; i++)`
	`   sum += a[i];`
	`double average = sum / n;`

reverse the values within an array	`for (int i = 0; i < n/2; i++)`
	`{`
	`   double temp = a[i];`
	`   a[i] = a[n-1-i];`
	`   a[n-i-1] = temp;`
	`}`

copy sequence of values to another array	`double[] b = new double[n];`
	`for (int i = 0; i < n; i++)`
	`   b[i] = a[i];`

Array as Stack
Demo Program: StackOfIntegers.java

Go BlueJ!!!

- `size` is also used as the top pointer for the stack.

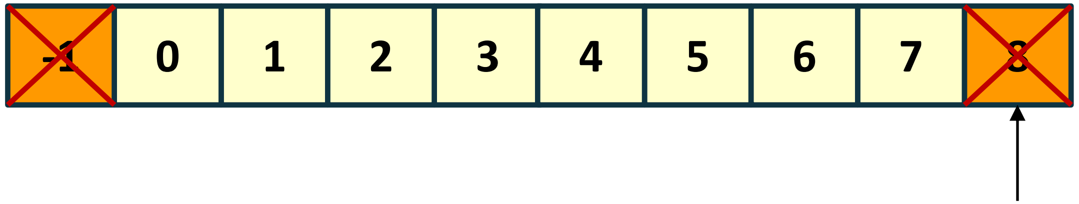
Discrete Space

(last included, end not included)

Count = lastIndex-firstIndex+1 ;

Count = endIndex-startIndex;

Array with 8 elements (indexed from 0 to 7)



~~-1~~	0	1	2	3	4	5	6	7	~~8~~

8-0 (=7-0+1) = 7- (-1)

Number of elements is 8-0 or 7 – (-1).

Use 7 – (-1) (Last element – the element before first element)

Or, 8-0 (The empty element – the first element)

Base and local index

n-i	-8	-7	-6	-5	-4	-3	-2	-1	0
$b+i$	4	5	6	7	8	9	10	11	~~12~~
i	0	1	2	3	4	5	6	7	8

Index = Base + local_Index

= $b+i$

Index = Length - local_neg_Index

= n-i

Base = b = 4, Length = n = $a.length$

Indexing for ASCII code

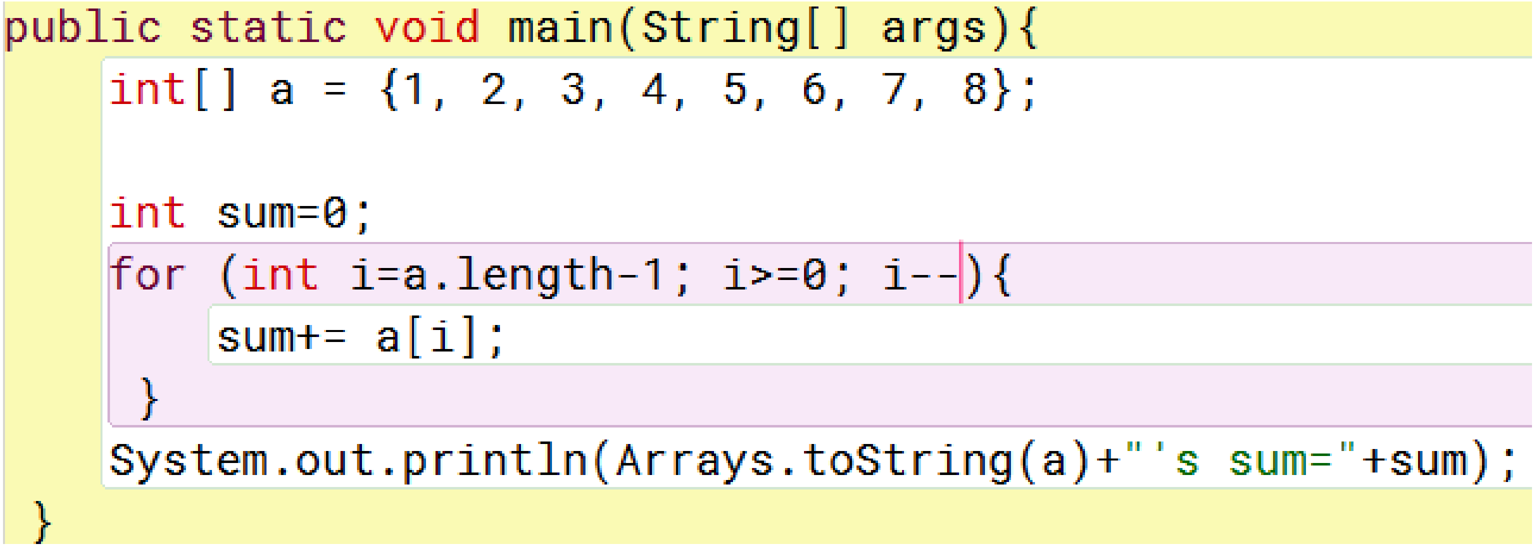
- Total number of ASCII letters: ‘Z’-’A’+1 (alphabet size)
- Indexing for a letter (‘B’): ‘A’ + (‘B’-’A’)
- Traversing through the whole alphabet: ‘A’ + i;   i is from 0 to 25
- Letter indexing is also used for histogram: a[‘X’-’A’] to store the number of occurrence for the letter ‘X’

Sum up an array of 8 elements with index from the last element to the first.

- Write a program to sum up

int a = {1, 2, 3, 4, 5, 6, 7, 8};

- starting from 8 down to 1 with proper indexing.

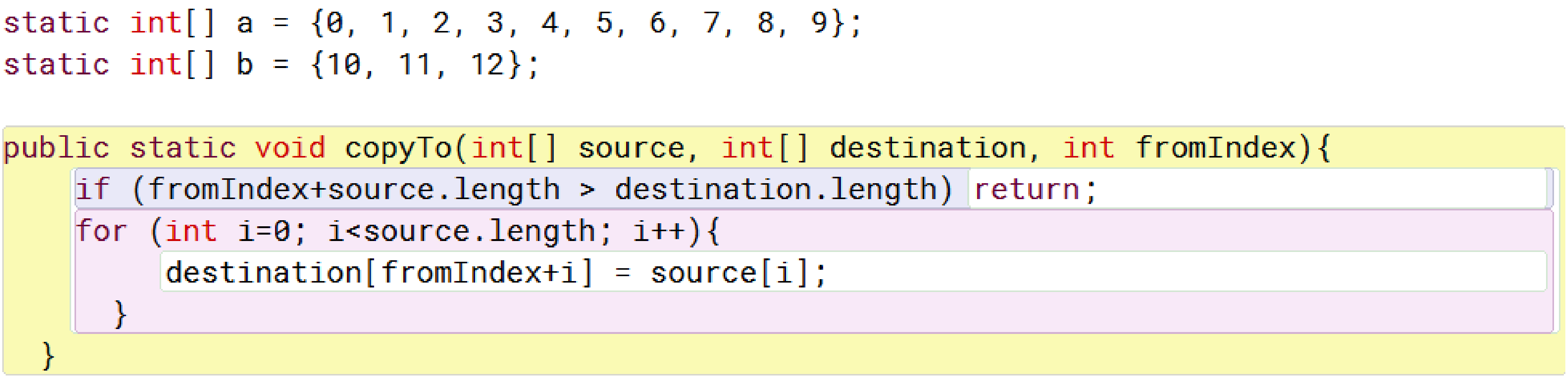


```
public static void main(String[] args){
    int[] a = {1, 2, 3, 4, 5, 6, 7, 8};

    int sum=0;
    for (int i=a.length-1; i>=0; i--){
        sum+= a[i];
     }
    System.out.println(Arrays.toString(a)+"'s sum="+sum);
 }
```

CopyTo(src, dest, fromIndex) Method

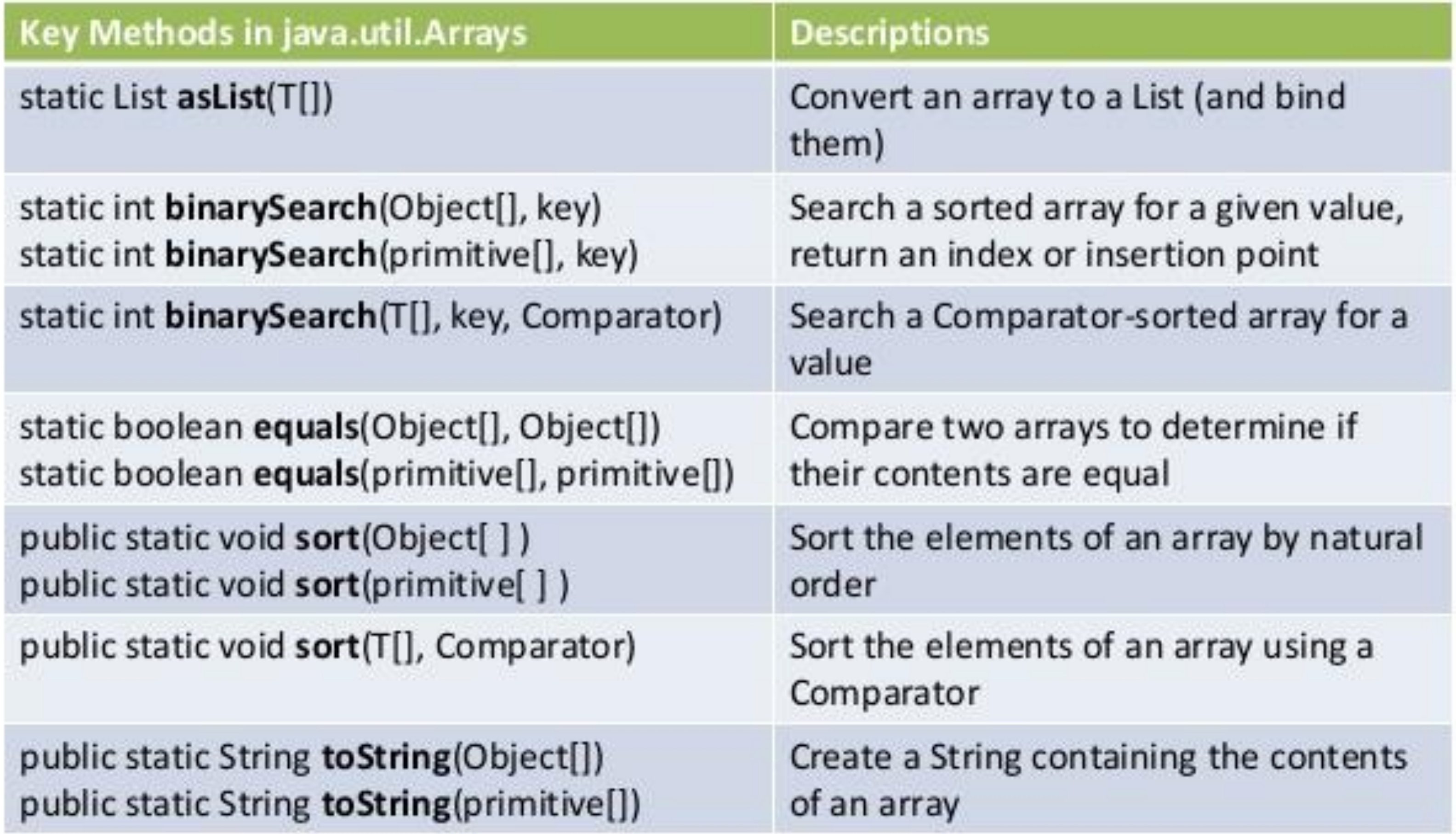
Demo Program: CopyTo.java



```java
static int[] a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
static int[] b = {10, 11, 12};

public static void copyTo(int[] source, int[] destination, int fromIndex){
    if (fromIndex+source.length > destination.length) return;
    for (int i=0; i<source.length; i++){
        destination[fromIndex+i] = source[i];
    }
}
```

Key Methods in Arrays



Key Methods in java.util.Arrays	Descriptions
static List **asList**(T[])	Convert an array to a List (and bind them)
static int **binarySearch**(Object[], key) static int **binarySearch**(primitive[], key)	Search a sorted array for a given value, return an index or insertion point
static int **binarySearch**(T[], key, Comparator)	Search a Comparator-sorted array for a value
static boolean **equals**(Object[], Object[]) static boolean **equals**(primitive[], primitive[])	Compare two arrays to determine if their contents are equal
public static void **sort**(Object[ ]) public static void **sort**(primitive[ ])	Sort the elements of an array by natural order
public static void **sort**(T[], Comparator)	Sort the elements of an array using a Comparator
public static String **toString**(Object[]) public static String **toString**(primitive[])	Create a String containing the contents of an array

(Non-AP: Use with cautions)

For Each Loop

ITERABLE ITEMS ONLY

Enhanced for-loop

Demo Program: ForEach1.java

General Form for the Enhanced for loop (the for-each loop):

```
datatype[] arrayName = /* array filled in some way */
for (datatype temporaryVarible: arrayName){
   // instructions that sue temporaryVariable
}
```

What data structure can be accessed by for-each loop?

Any container class that implements both Iterable Interface and Iterator Interface can be accessed by for-each loop.

Iterable Interface:

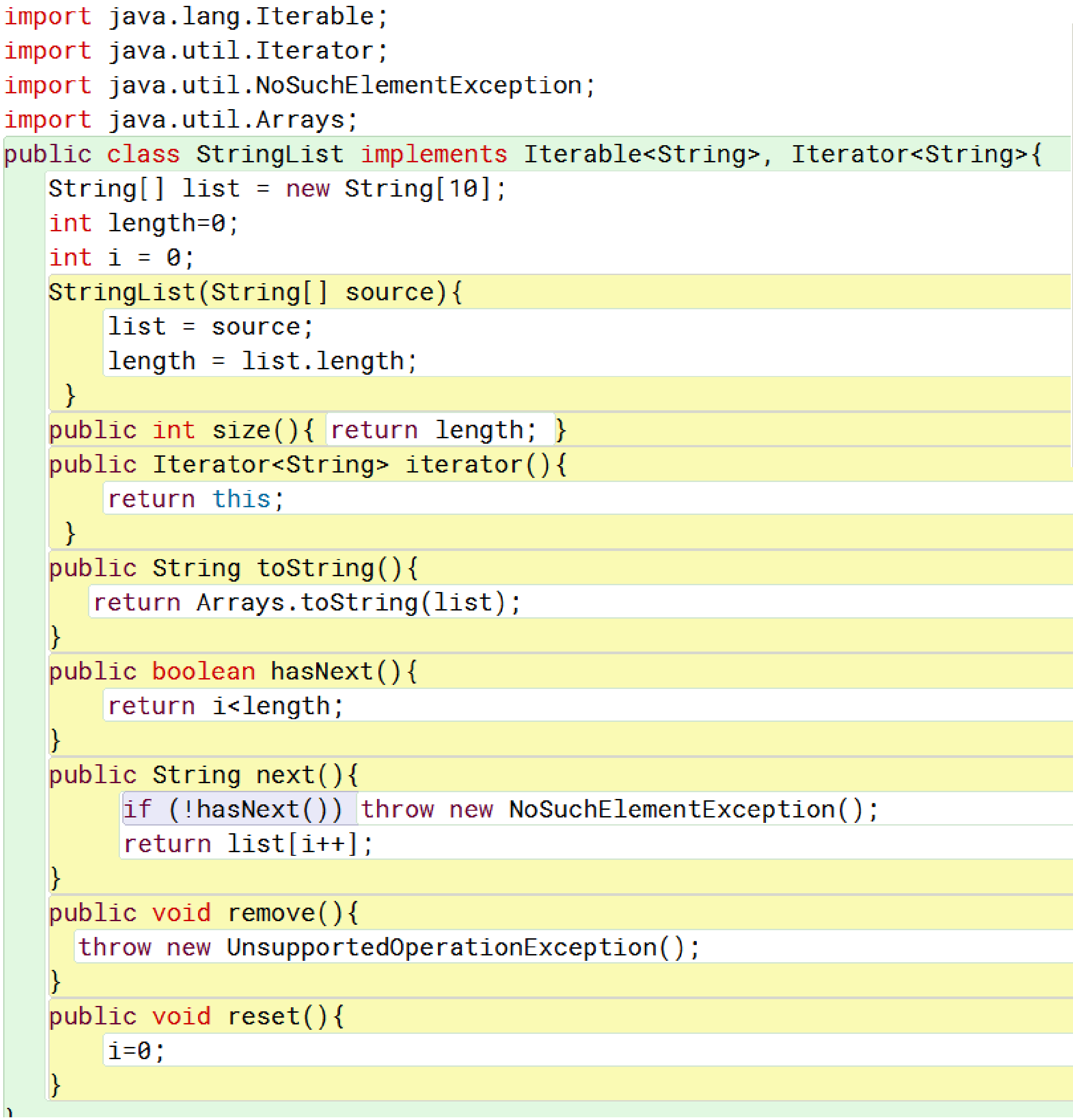
iterator() method

Iterator Interface:

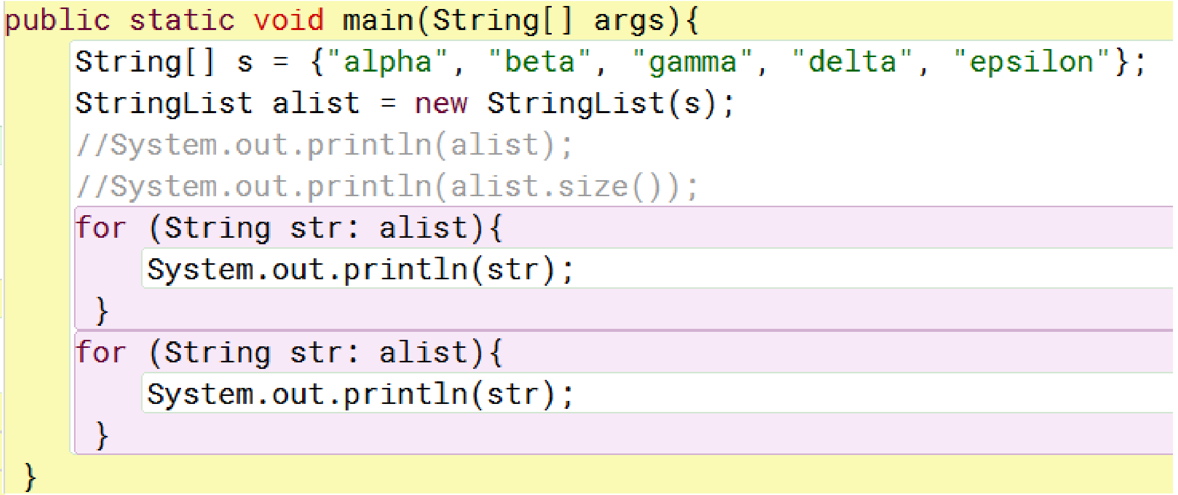
hasNext() method

next() method

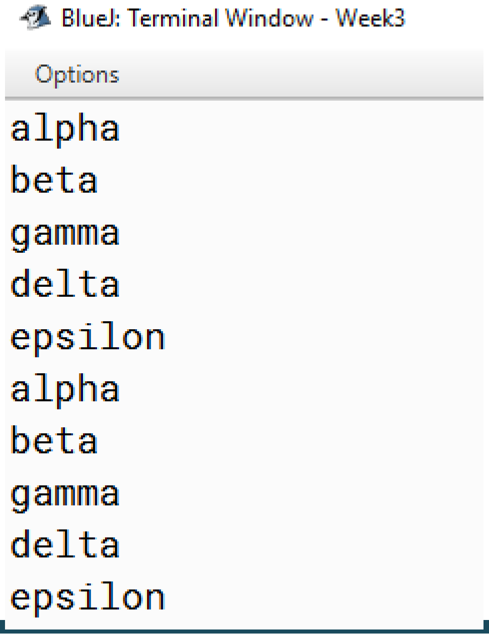
remove() method



```java
import java.lang.Iterable;
import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.Arrays;
public class StringList implements Iterable<String>, Iterator<String>{
    String[] list = new String[10];
    int length=0;
    int i = 0;
    StringList(String[] source){
        list = source;
        length = list.length;
    }
    public int size(){ return length; }
    public Iterator<String> iterator(){
        return this;
    }
    public String toString(){
       return Arrays.toString(list);
    }
    public boolean hasNext(){
        return i<length;
    }
    public String next(){
         if (!hasNext()) throw new NoSuchElementException();
         return list[i++];
    }
    public void remove(){
      throw new UnsupportedOperationException();
    }
    public void reset(){
        i=0;
    }
}
```



```java
public static void main(String[] args){
    String[] s = {"alpha", "beta", "gamma", "delta", "epsilon"};
    StringList alist = new StringList(s);
    //System.out.println(alist);
    //System.out.println(alist.size());
    for (String str: alist){
        System.out.println(str);
    }
    for (String str: alist){
        System.out.println(str);
    }
}
```



```
alpha
beta
gamma
delta
epsilon
alpha
beta
gamma
delta
epsilon
```

SECTION 3

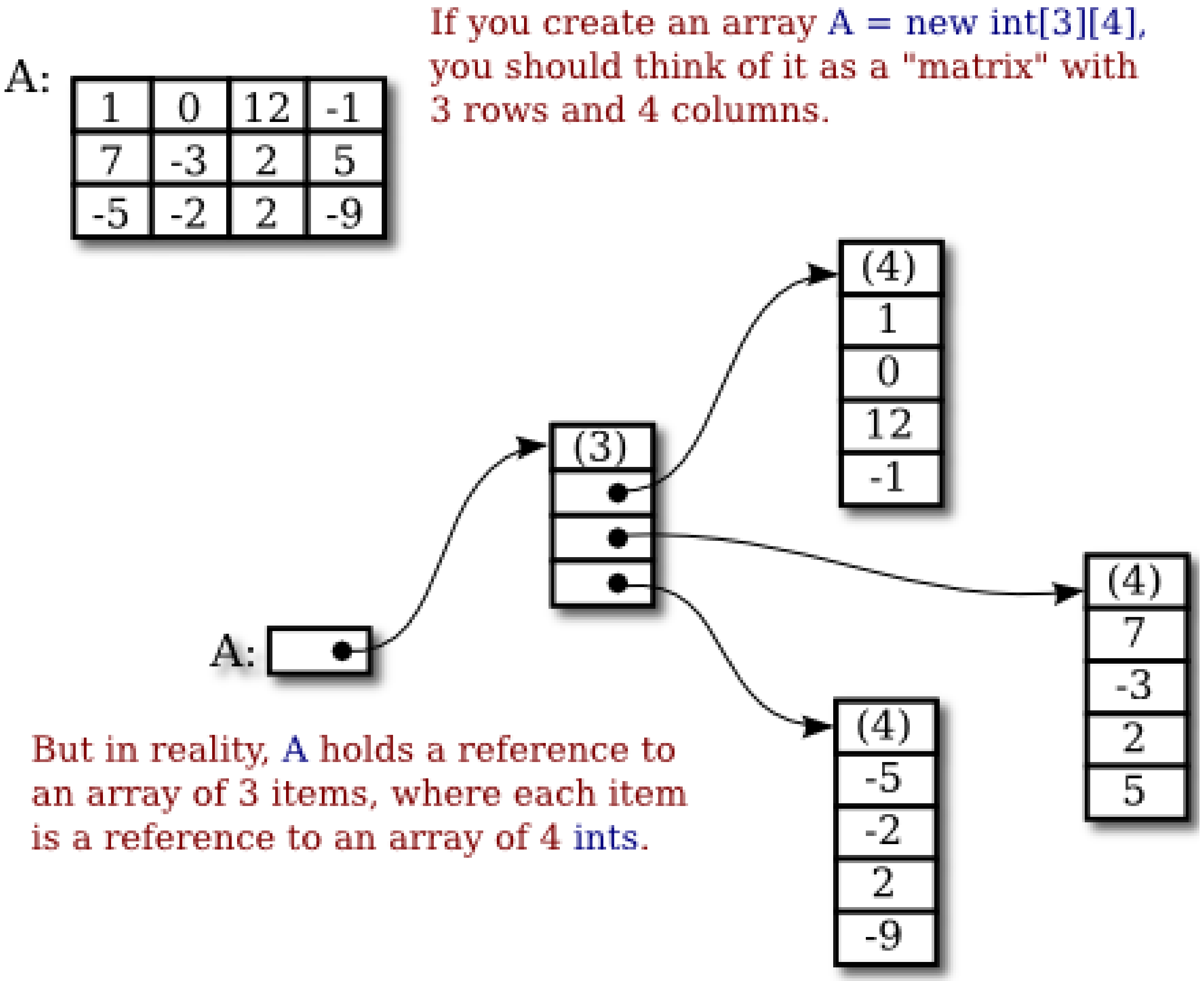
2D Array

General Form for Creating 2-D Array Using a Pre-defined List of Data

```
datatype[][] nameOf2DArray = {
   {value1, value2, value3},
   {value4, value5, value6},
   { …, …, …}
};
```

```
String[][] candyBoard = {
 {“Jelly Bean”, “Lozenge”, “Lemon Drop”},
 {“Gum Square”, “Lollipop Head”, “Jujube Cluster”},
 {“Lozenge”, “Lollipop Head”, “Lemon Drop”},
 {“Jelly Bean”, “Lollipop Head”, “Lozenge”}
};
```

2D Array is Array of Arrays

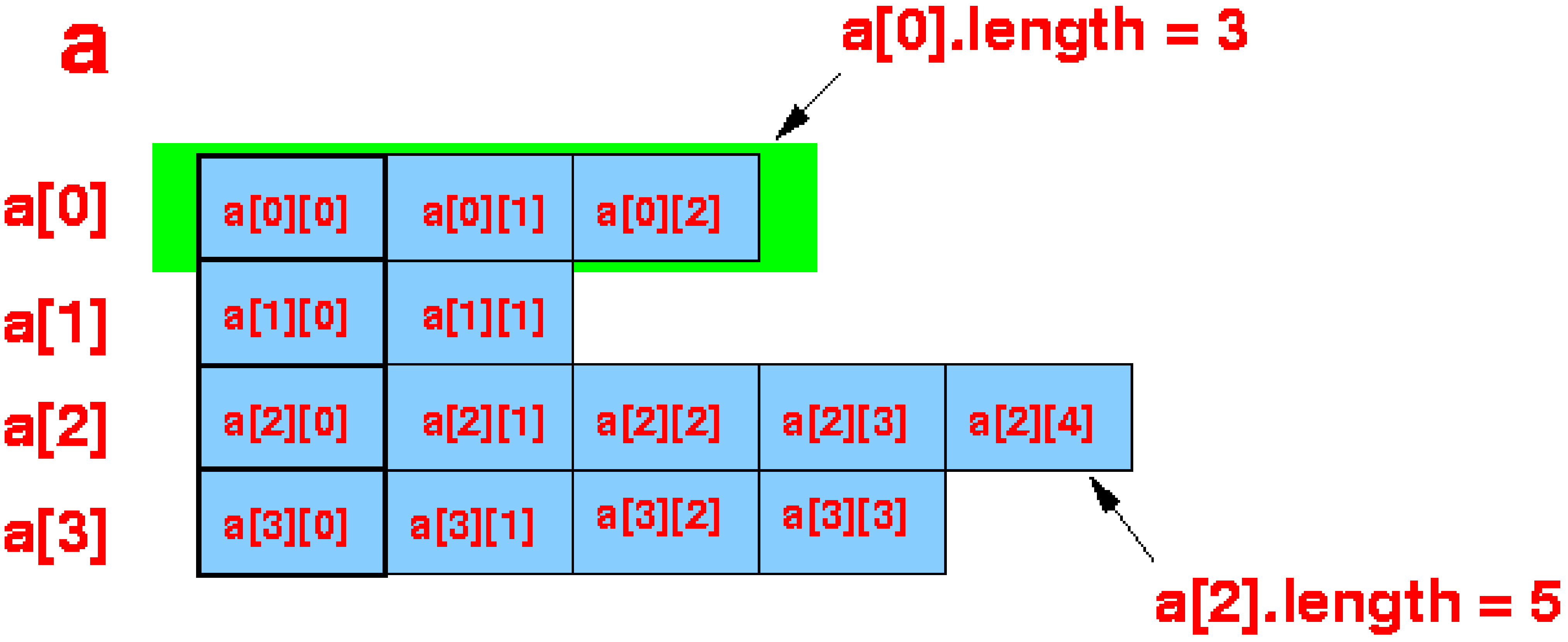


If you create an array A = new int[3][4], you should think of it as a "matrix" with 3 rows and 4 columns.

A:

1	0	12	-1
7	-3	2	5
-5	-2	2	-9

But in reality, A holds a reference to an array of 3 items, where each item is a reference to an array of 4 ints.



a

a[0] a[0][0] a[0][1] a[0][2]

a[1] a[1][0] a[1][1]

a[2] a[2][0] a[2][1] a[2][2] a[2][3] a[2][4]

a[3] a[3][0] a[3][1] a[3][2] a[3][3]

a[0].length = 3

a[2].length = 5

Using the length Field to Find the Number of Rows and Columns

Retrieve the number of rows from a 2-D array:

```
double[][] myBoard = new double[8][3];
int result = myBoard.length;
```

Retrieve the number of columns from a 2-D array:

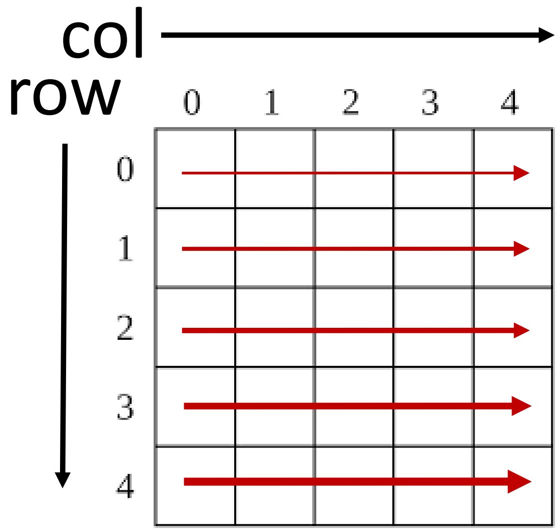
```
double[][] myBoard = new double[8][3];
int result1 = myBoard[0].length;
int result2 = myBoard[5].length;
```

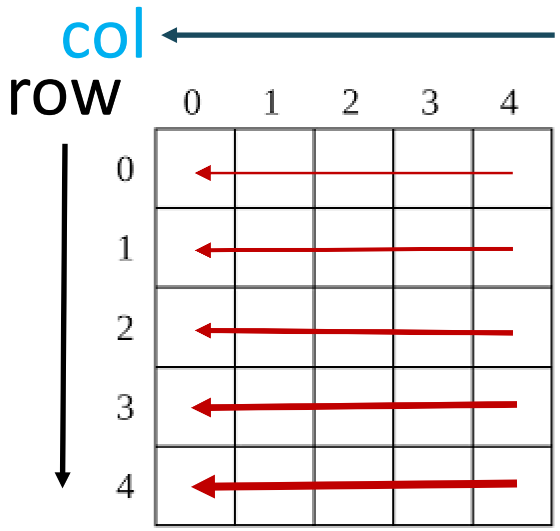
2D Traversal

ROW/COLUMN MANAGEMENT

2-D Array Indexing for Traversal

int row, col; int[][] m=new int[5][5];

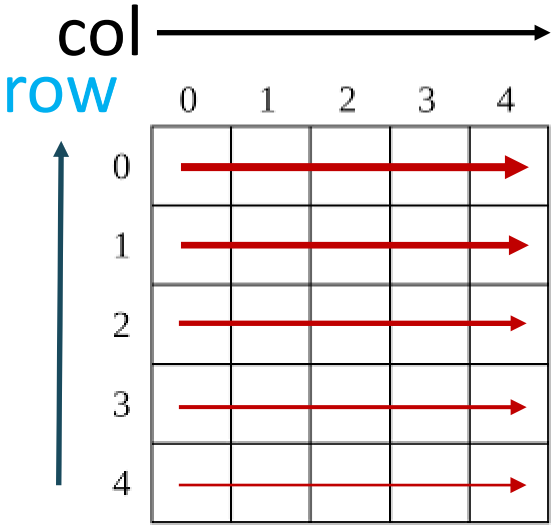


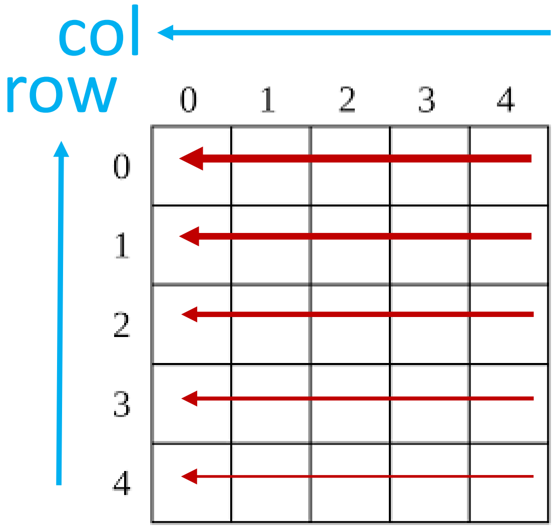


```
for(row=0; row<m.length; row++)
  for(col=0; col<m[0].length; col++)
      System.out.println(m[row][col]);
```

```
for(row=0; row<m.length; row++)
  for(col=m[0].length-1; col>=0; col--)
      System.out.println(m[row][col]);
```

2-D Array Indexing for Traversal
int row, col; int[][] m=new int[5][5];



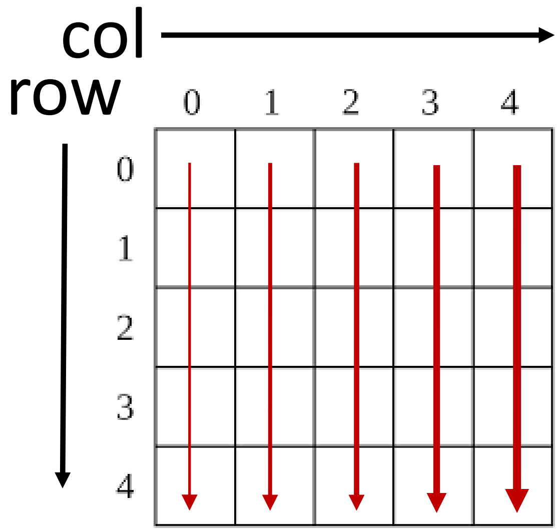


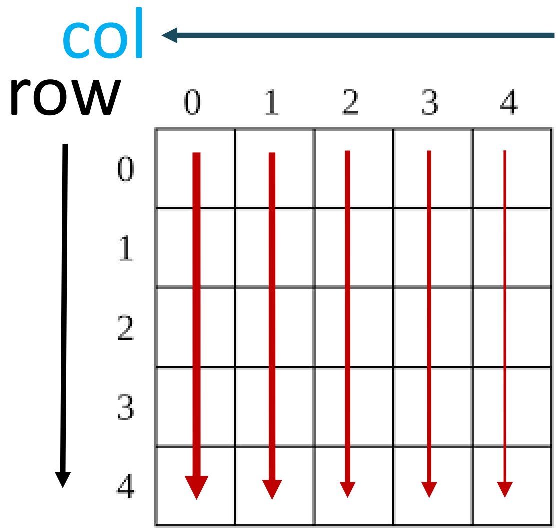
```
for(row=m.length-1; row>=0; row--)
  for(col=0; col<m[0].length; col++)
     System.out.println(m[row][col]);
```

```
for(row=m.length-1; row>=0; row--)
  for(col=m[0].length-1; col>=0; col--)
     System.out.println(m[row][col]);
```

2-D Array Indexing for Traversal

int row, col; int[][] m=new int[5][5];

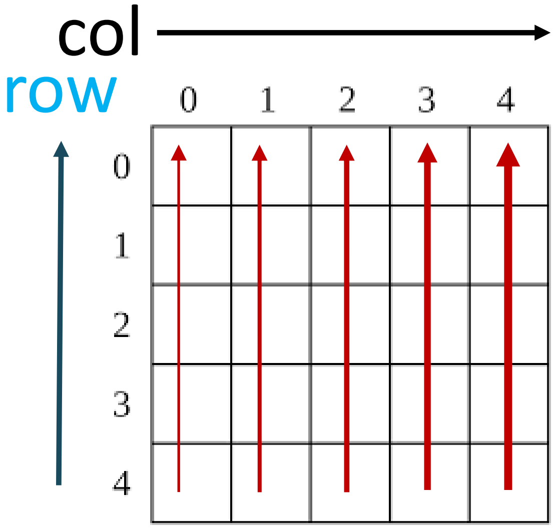


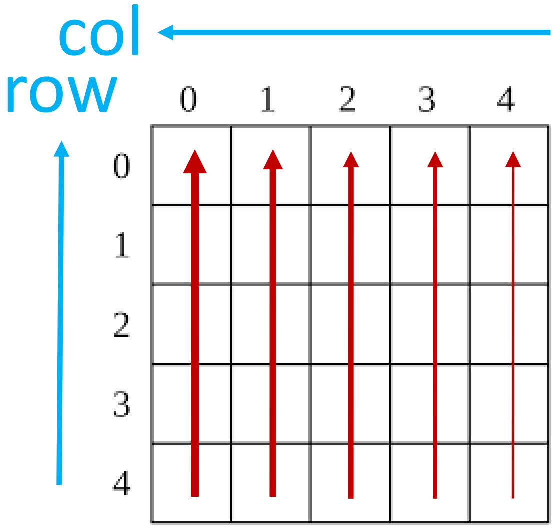


```
for(col=0; col<m[0].length; col++)
  for(row=0; row<m.length; row++)
    System.out.println(m[row][col]);
```

```
for(col=m[0].length-1; col>=0; col--)
  for(row=0; row<m.length; row++)
    System.out.println(m[row][col]);
```

2-D Array Indexing for Traversal
int row, col; int[][] m=new int[5][5];

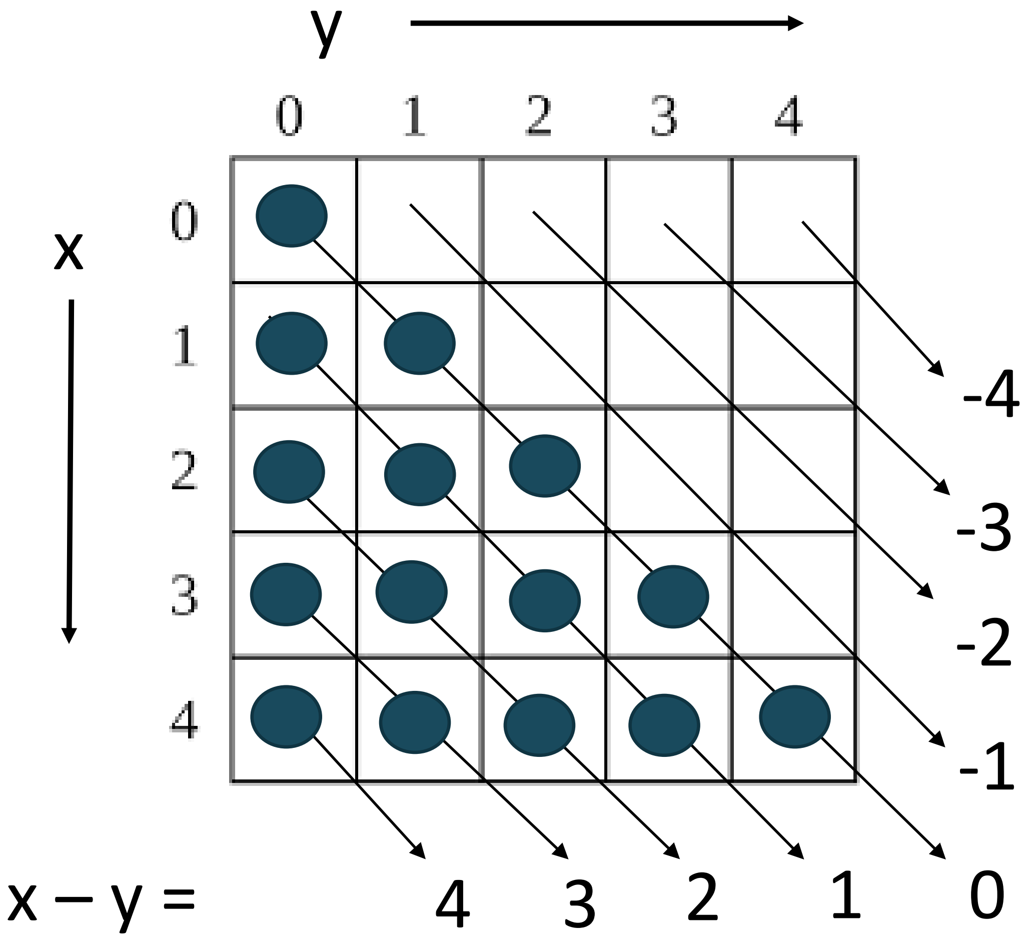




```
for(col=0; col<m[0].length; col++)
  for(row=m.length-1; row>=0; row--)
    System.out.println(m[row][col]);
```

```
for(col=m[0].length-1; col>=0; col--)
  for(row=m.length-1; row>=0; row--)
    System.out.println(m[row][col]);
```

Partial Array Traversal



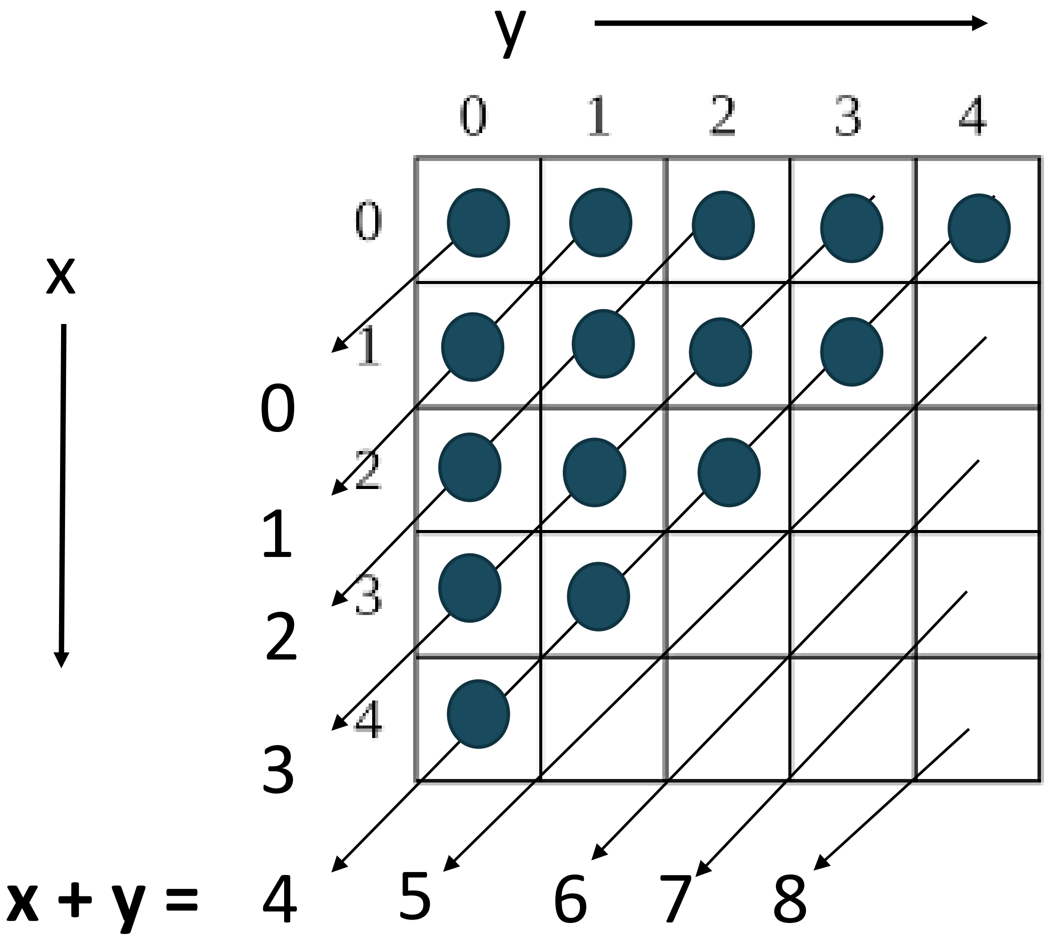
```
for (int I = 0; i<m.length; i++)
   for (int j =0; j< i +1;  j ++)
      { /* do something */}
```

Index:

Stop Condition: j stop at j = i.

i – j = 0;

Partial Array Traversal



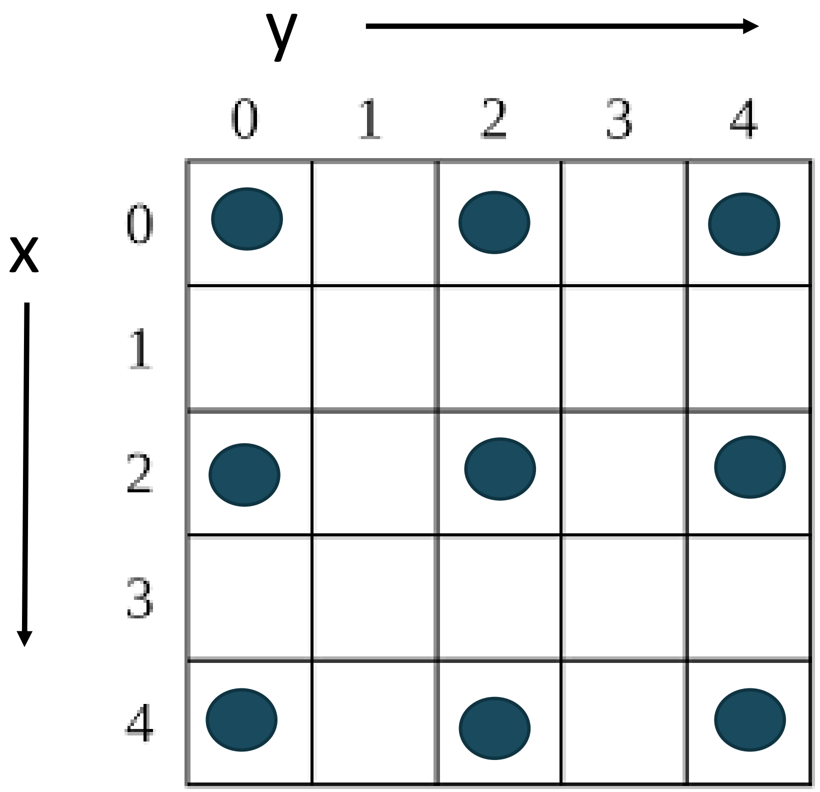
```
for (int i = 0; i<m.length; i++)
  for (int j =m.length-1-i; j>=0;  j --)
    { /* do something */}
```

Index:

Start Condition: i + j = m.length-1;

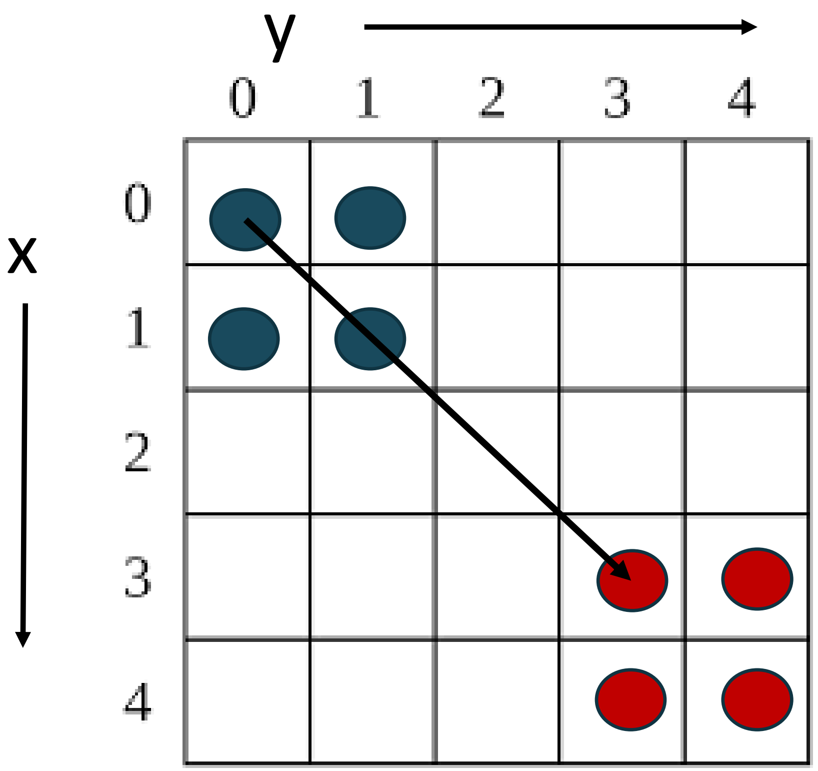
i + j = 4;

Partial Array Traversal

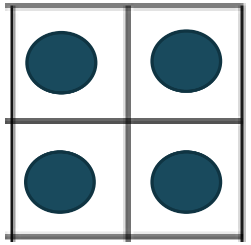


```
for (int i = 0; i<m.length; i+=2)
   for (int j =0; j<m[0].length;  j+=2)
      { /* do something */}
```

Vector Operation (Area Copy)



```
for (int i = 0; i<2; i++)
  for (int j=0; j<2; j++)
    m[3+i][3+j] = m[0+i][0+j];
      /* 0 is not needed */
```

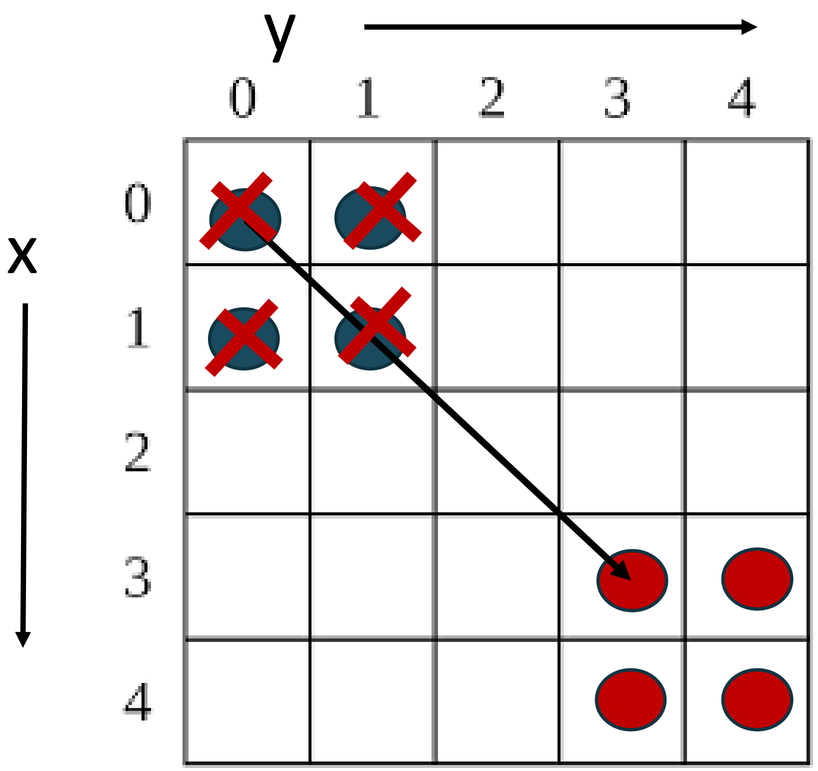


Area to be Copied: **2 x 2** block

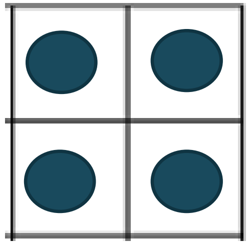
From (0, 0) to (3, 3)

If there is overlap, we need to put the area into a buffer first.

Vector Operation (Area Move)



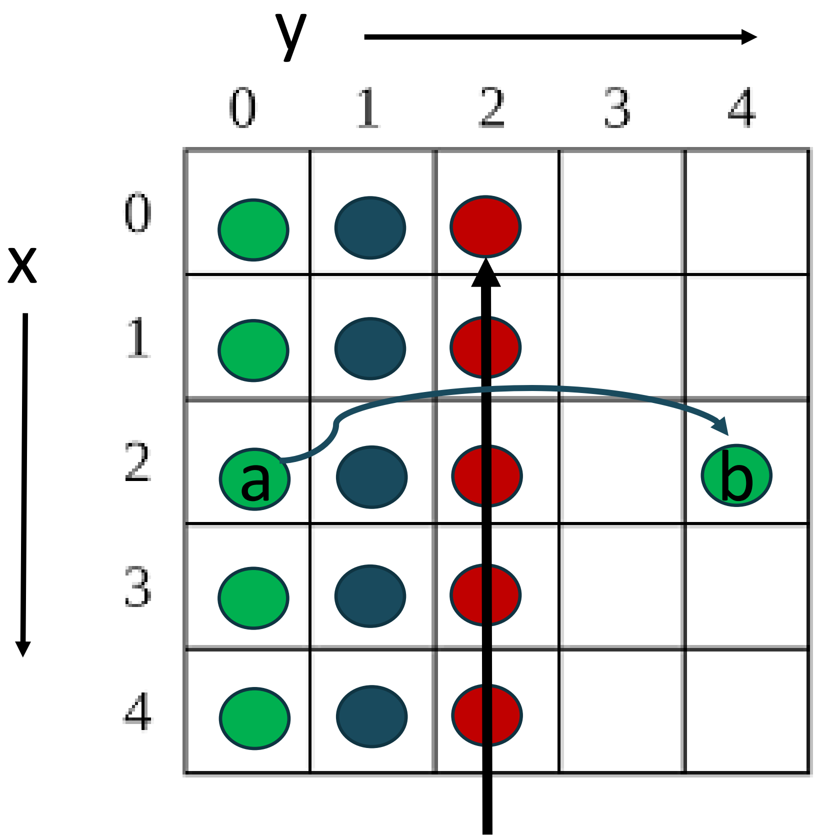
```
for (int i = 0; i<2; i++)
   for (int j=0; j<2; j++)
     m[3+i][3+j] = m[0+i][0+j];
for (int i = 0; i<2; i++)
   for (int j=0; j<2; j++)
     m[0+i][0+j]=0;
```



Area to be Copied: **2 x 2** block

From (0, 0) to (3, 3)

Flip



Symmetric Line for Flipping:

j = 2;   // j = m.length/2

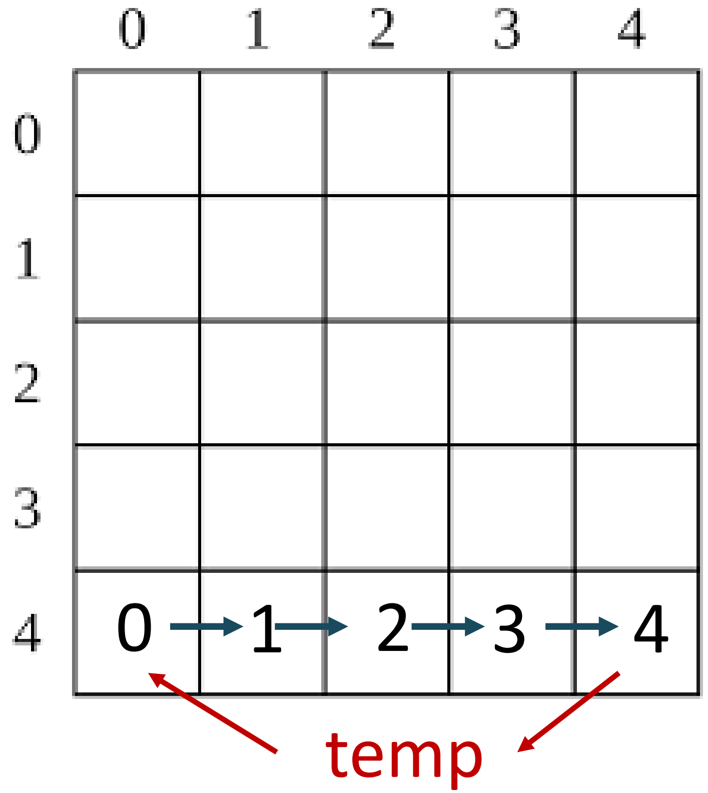
(a + b)/2 = 2;

b == 4-a;   // b = m.length –a-1;

```
for (int i=0; i<m.length; i++){
  for (int j=0; j<m.length/2; j++){
    m[i][m.length-j-1] = m[i][j];
  }
}
```

Area Shift

```
for (int i=0; i<m.length; i++){
    int temp = m[i][m.length-1];
    for (int j=m[0].length-2; j<=0; j--){
        m[i][j+1] = m[i][j];
    }
    m[i][0] =temp;
}
```

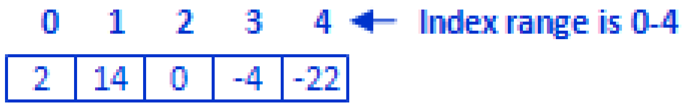


One Row after Shift: 4 0 1 2 3

SECTION 4

ArrayList

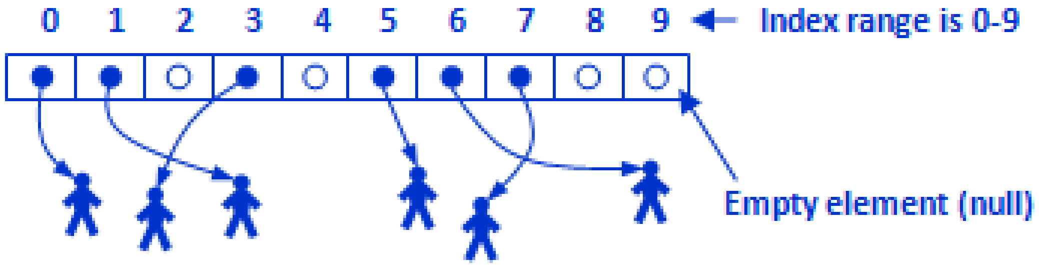
Array of 5 integers



0	1	2	3	4
2	14	0	-4	-22

Index range is 0-4

Array of 10 agents

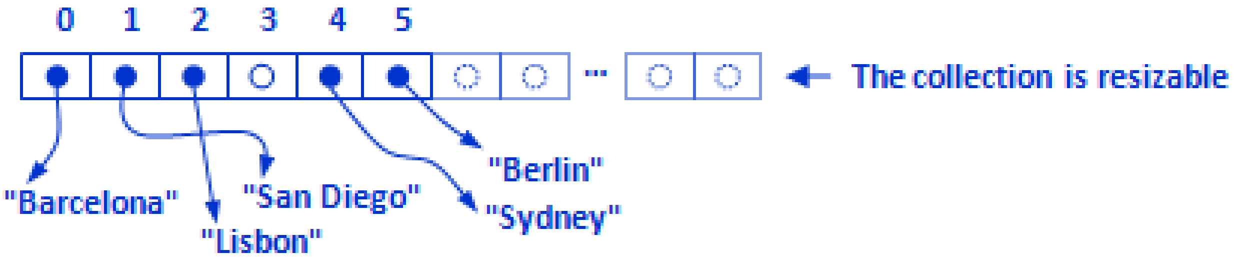


0 1 2 3 4 5 6 7 8 9

Index range is 0-9

Empty element (null)

ArrayList (collection) of strings, currently contains 6 elements



0 1 2 3 4 5

The collection is resizable

"Barcelona"

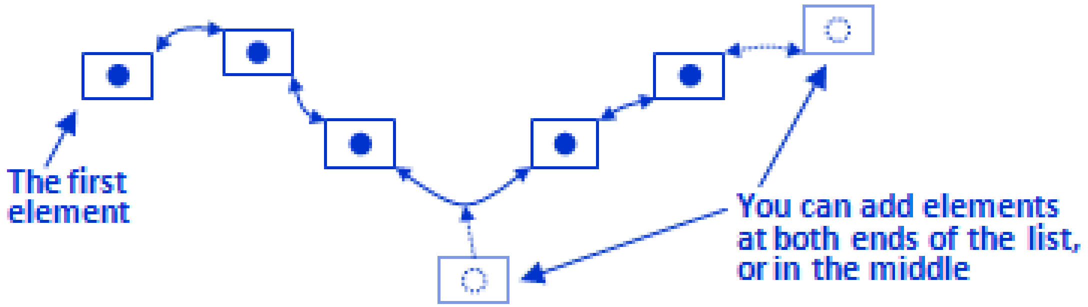
"Lisbon"

"San Diego"

"Berlin"

"Sydney"

LinkedList (collection)



The first element

You can add elements at both ends of the list, or in the middle

ArrayList Class, List/Iterable Interface

Declaration of ArrayList

like a train. The datatype it carries is like the cargo.

In declaring a variable of type **ArrayList** we use a statement like this:

```
    ArrayList<String> aList;
```

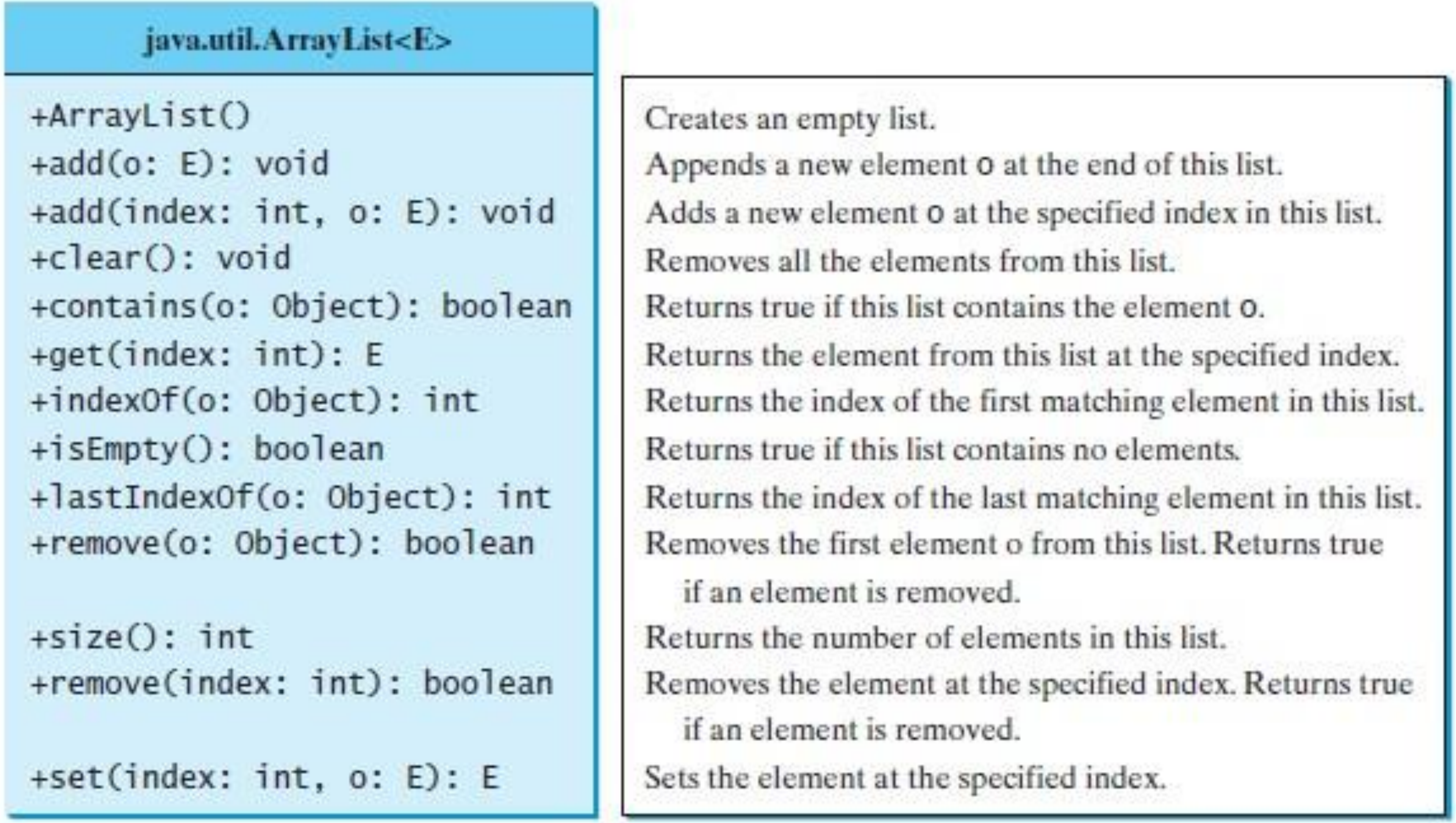
The word between the *angle brackets*, **<…>**, indicates the data type of the elements that the **ArrayList** will store. In this case, **aList** is declared to be an **ArrayList** each of whose elements will be a **String**.

The following statement creates an **ArrayList** of **String**s and then assigns it to **aList**:

```
    aList = new ArrayList<String>();
```

We can also declare and assign to the variable in a single statement:

```
    ArrayList<String> aList = new ArrayList<String>();
```



java.util.ArrayList<E>	
`+ArrayList()`	Creates an empty list.
`+add(o: E): void`	Appends a new element `o` at the end of this list.
`+add(index: int, o: E): void`	Adds a new element `o` at the specified index in this list.
`+clear(): void`	Removes all the elements from this list.
`+contains(o: Object): boolean`	Returns true if this list contains the element `o`.
`+get(index: int): E`	Returns the element from this list at the specified index.
`+indexOf(o: Object): int`	Returns the index of the first matching element in this list.
`+isEmpty(): boolean`	Returns true if this list contains no elements.
`+lastIndexOf(o: Object): int`	Returns the index of the last matching element in this list.
`+remove(o: Object): boolean`	Removes the first element o from this list. Returns true if an element is removed.
`+size(): int`	Returns the number of elements in this list.
`+remove(index: int): boolean`	Removes the element at the specified index. Returns true if an element is removed.
`+set(index: int, o: E): E`	Sets the element at the specified index.

ArrayList

- An arraylist alwasys starts out empty.
- An arraylist is resizable.
- An arraylist requires an import statement.
- An arraylist can only store objects.

ArrayList Methods

ADD, REMOVE, SET, INDEXOF

Differences and Similarities between Arrays and ArrayList

Operation	*Array*	*ArrayList*
Creating an array/ArrayList	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`

Important Methods

ArrayList<E>

- void add(E object)
- void add(int index, E Object)
- int size()
- E remove(int index)
- E get(int index)
- E set(int index, E object)

Creation of ArrayList:

Constructor, Loop Instantiation with add()

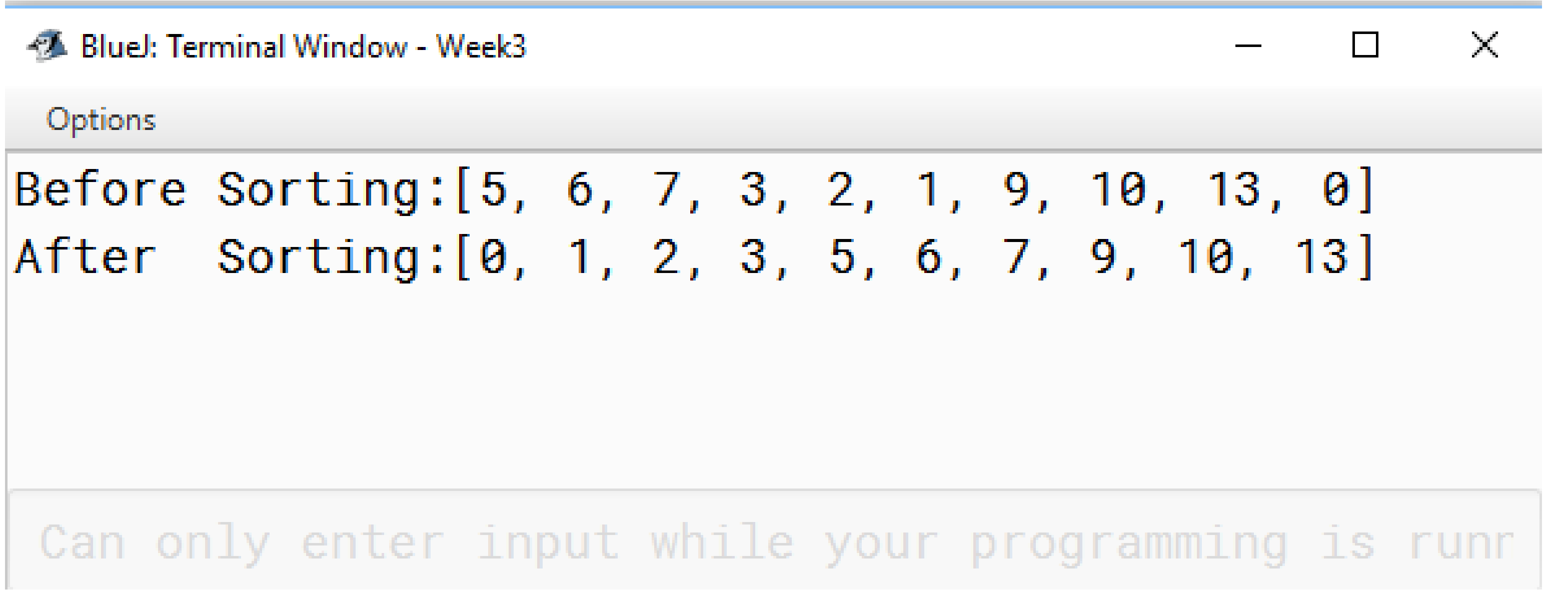
```
public static void main(String[] args){
  int count =5;
  ArrayList<Integer> aList= new ArrayList<Integer>();
  for (int i=0; i<5; i++){
    aList.add((int)(Math.random()*8));
  }
  System.out.println(“Loop Creation of an ArrayList: ”+aList);
}
```

Guideline

- Getter and Setter Methods using index are fine.
- Remove, add with index need extra cautions.

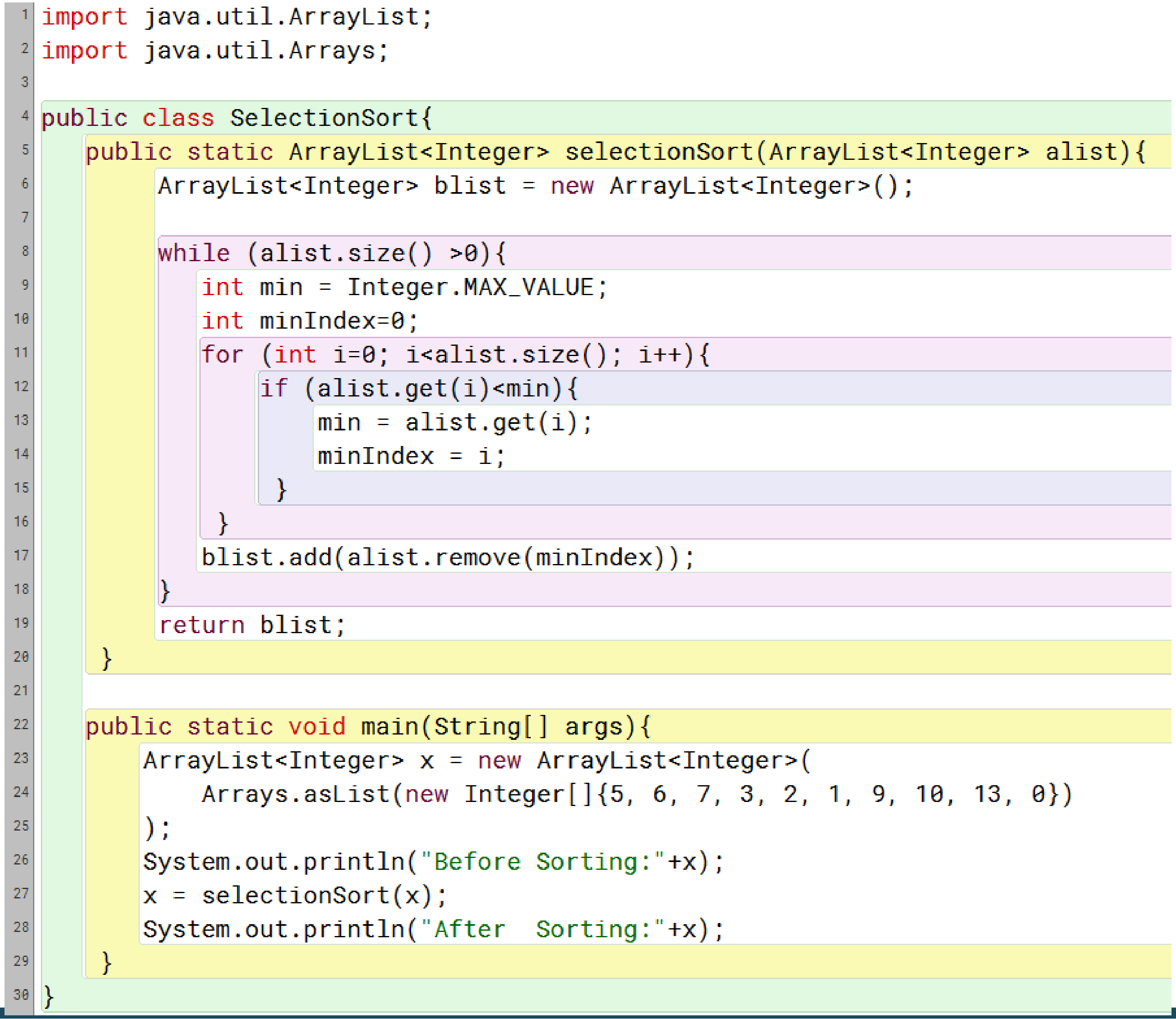
Selection Sort by ArrayList

- If insertion sort or selection sort is required, arraylist is a better option than array.
- It is just a finding maximum with a outer loop.



```
Before Sorting:[5, 6, 7, 3, 2, 1, 9, 10, 13, 0]
After  Sorting:[0, 1, 2, 3, 5, 6, 7, 9, 10, 13]
```

Can only enter input while your programming is runr



```java
import java.util.ArrayList;
import java.util.Arrays;

public class SelectionSort{
    public static ArrayList<Integer> selectionSort(ArrayList<Integer> alist){
        ArrayList<Integer> blist = new ArrayList<Integer>();

        while (alist.size() >0){
            int min = Integer.MAX_VALUE;
            int minIndex=0;
            for (int i=0; i<alist.size(); i++){
                if (alist.get(i)<min){
                    min = alist.get(i);
                    minIndex = i;
                }
            }
            blist.add(alist.remove(minIndex));
        }
        return blist;
    }

    public static void main(String[] args){
        ArrayList<Integer> x = new ArrayList<Integer>(
            Arrays.asList(new Integer[]{5, 6, 7, 3, 2, 1, 9, 10, 13, 0})
        );
        System.out.println("Before Sorting:"+x);
        x = selectionSort(x);
        System.out.println("After  Sorting:"+x);
    }
}
```