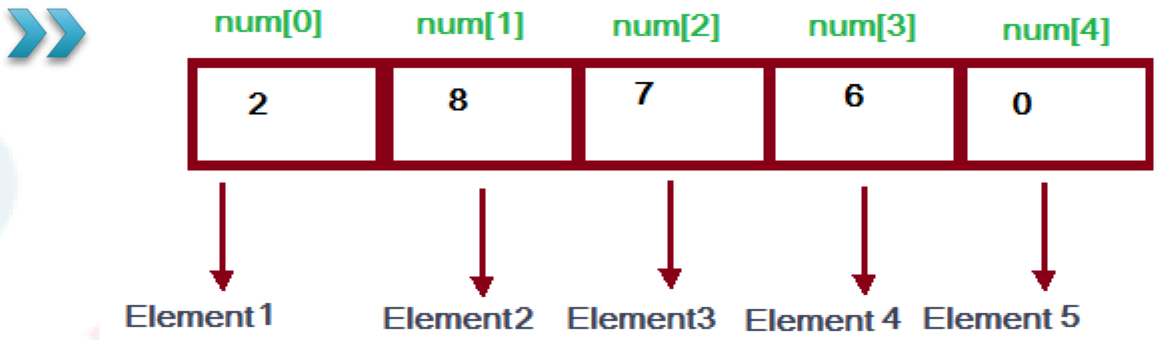
FACULTY OF INFORMATION TECHNOLOGY

# DATA STRUCTURES (CTDL)

Semester 1, 2021/2022

# One-Dimensional Arrays



## What is array?

- Array: *a data structure* that stores a fixed-size sequential collection of elements of the same types.

- The **size** of an array must be specified by an int value and not long or short.

- The direct superclass of an array type is Object.

- Array can be also be used as a static field, a local variable or a method parameter.

## Declaring Array Variables

- The syntax for declaring an array variable:

```
dataType[] arrayRefVar;
```

Or

```
dataType arrayRefVar[];
```

- Example:

```
double [] myList;
```

## Creating Array Variables

- Declaration of an array variable doesn’t allocate any space in memory for the array.

- Only a storage location for the reference to an array is created.

- If a variable doesn’t reference to an array, the value of the variable is null.

- Create an array by using the new operator with the following syntax:

  arrayRefVar = new dataType[arraySize];

## Creating Array Variables (cont.)

- This element does two things:
  1) It creates an array using new dataType[arraySize];
  2) It assigns the reference of the newly created array to the variable arrayRefVar

- Combining approach:

```
dataType[] arrayRefVar = new dataType[arraySize];
```

## Array Size and Default values

- When space for an array is allocated, the array size must be given.
- The size of an array cannot be changed after the array is created.
- Size can be obtained using

  arrayRefVar.length

- When an array is created, its elements are assigned the default value of 0 for the **numeric** primitive data types, '\u0000' (the null character) for **char** types, and false for **boolean** types.

How about **Object** types???

## Array Initializers

- Shorthand notation: combines declaring an array, creating an array and initializing it at the same time

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

- This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

```
int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
```

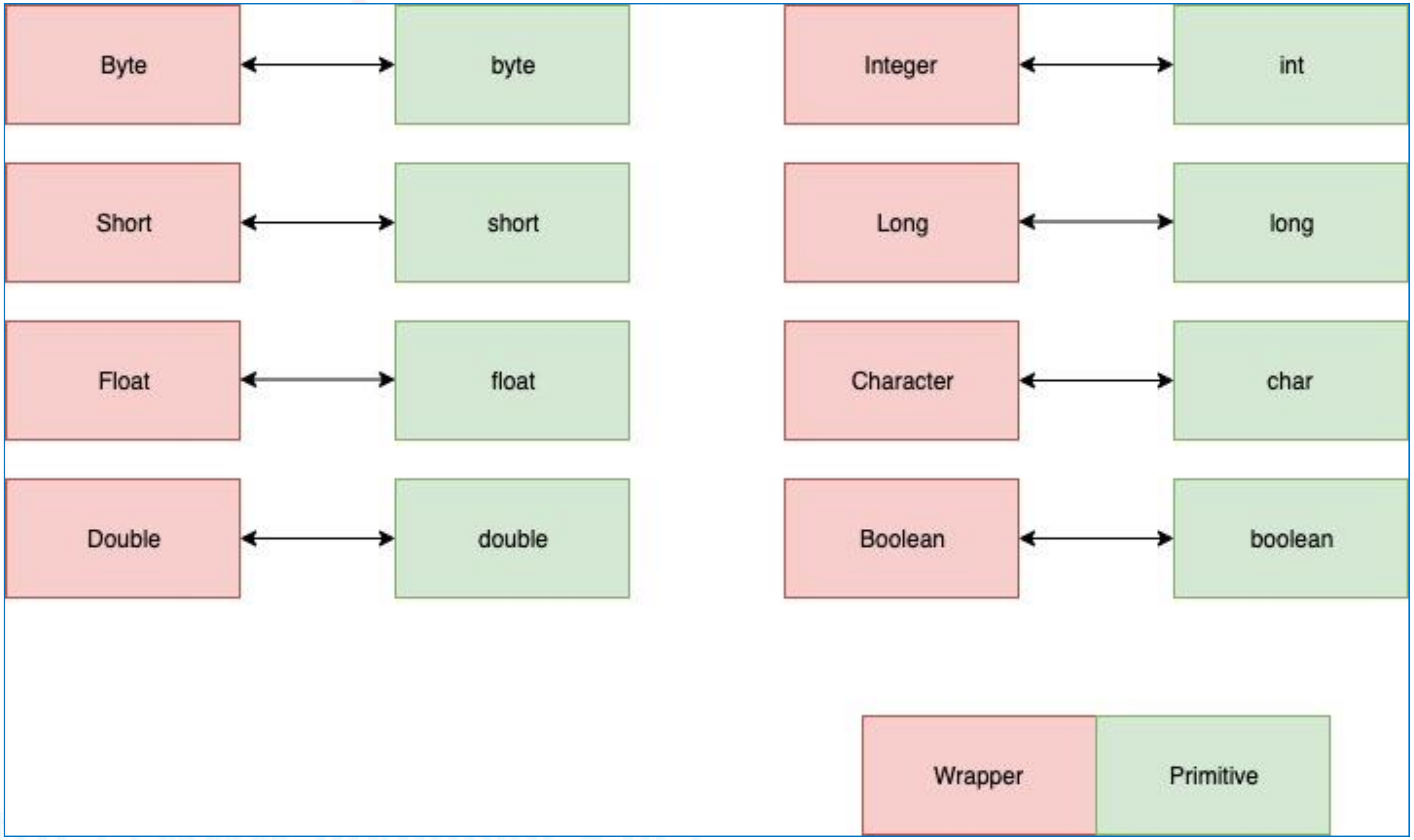
## Array Initializers (cont.)

- How to initialize an **Integer** array:

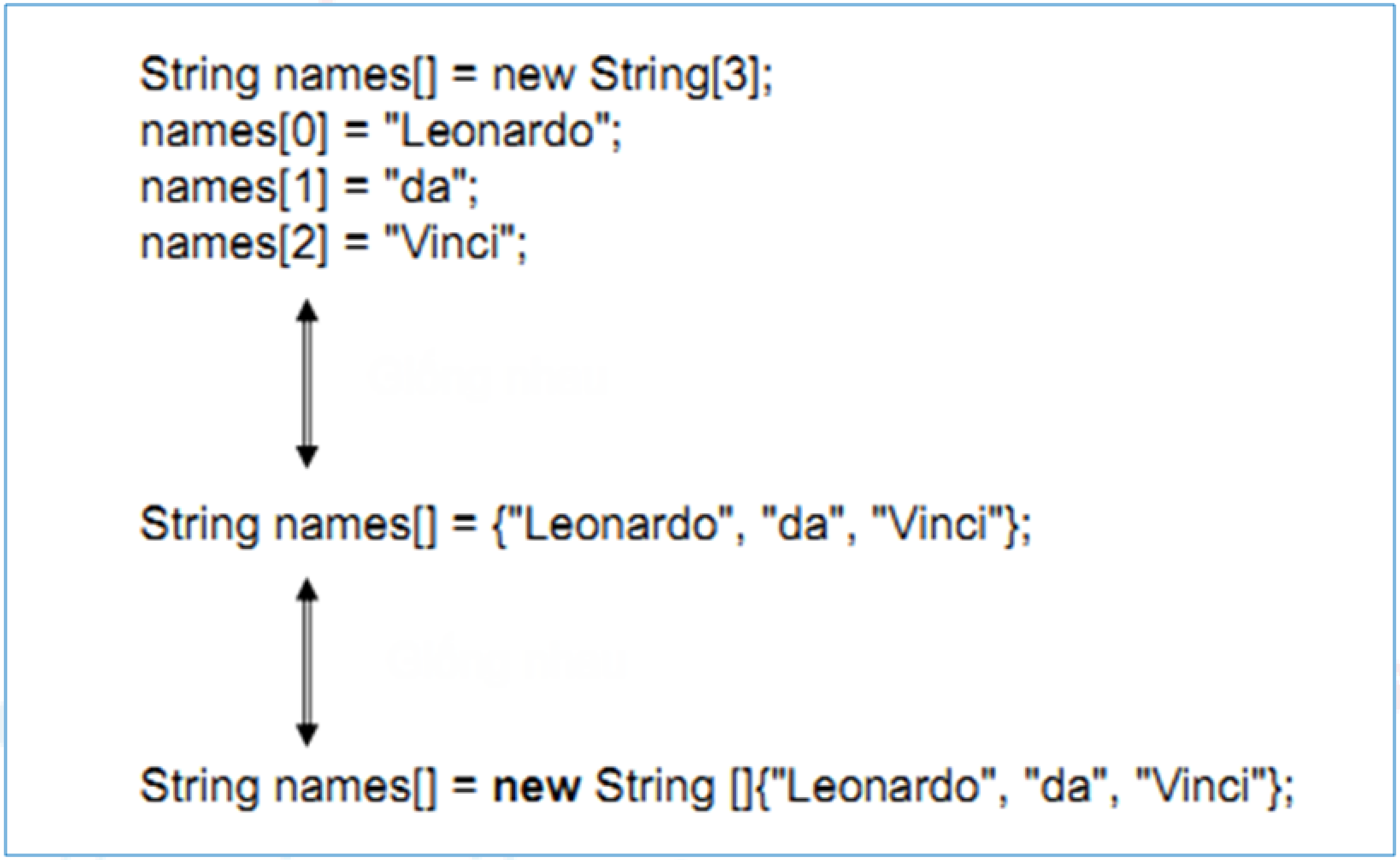
```
Integer[] arr = new Integer[2];
arr[0] = ?;
arr[1] = ?
```

```
Integer[] arr = {?, ?, …, ?};
```

# Wrapper class



## Example



```
String names[] = new String[3];
names[0] = "Leonardo";
names[1] = "da";
names[2] = "Vinci";
```

```
String names[] = {"Leonardo", "da", "Vinci"};
```

```
String names[] = new String []{"Leonardo", "da", "Vinci"};
```

# Accessing Array Elements

- The array elements are accessed through an index.
- The array indices are 0-based, they start from 0 to the length of the array -1
- Loop with for statement

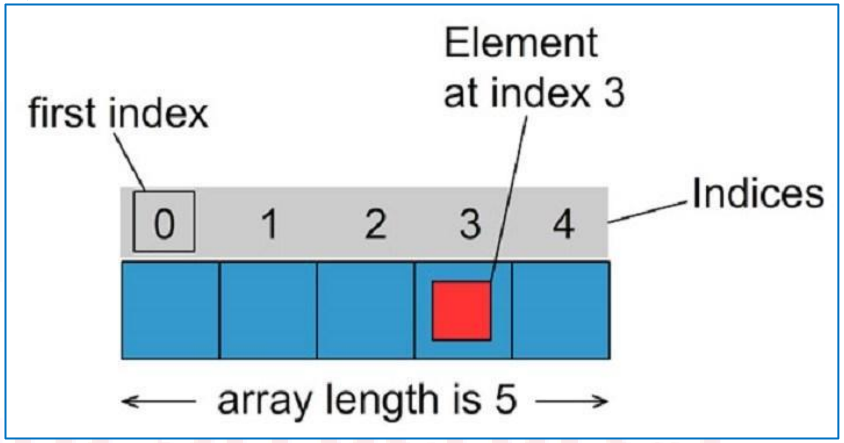
```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

Normal for:

```
    for (int i=0; i< myList.length; i++){
        //myList[i] ...;
    }
```

For each:

```
    for (double element : myList) {
        //element ...;
    }
```



How to use while statement???

## Passing Arrays to Methods

- Consider the following code fragment. What is the output?

```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }

  public static void m(int number, int[] numbers) {
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

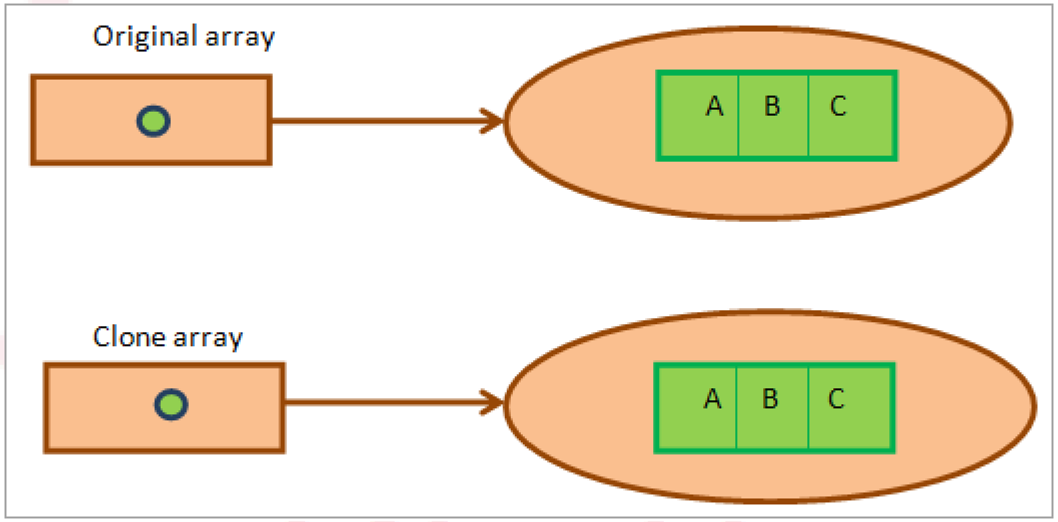
## Question

- **Pass by value**: makes a copy in memory of the parameter's value, or a copy of the contents of the parameter.

- **Pass by reference**: a copy of the address (or reference) to the parameter is stored rather than the value itself.

Java is pass by value and pass by reference?

## Cloning Arrays



## Cloning Arrays

- Java supports 3 approaches:
  - By using variable assignment (=)
  - By using clone() method
  - By using System.arraycopy()

## Cloning Arrays (cont.)

- Using variable assignment (=):

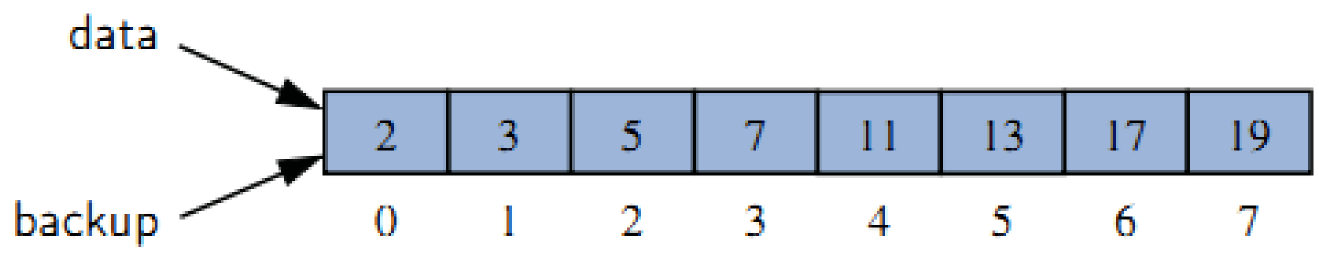
```
public static void main(String[] args) {
    int[] arr1 = {1, 2, 3};
    int[] arr2 = arr1;

    arr1[0]++;

    System.out.println(arr1[0]+" "+arr2[0]);
}
```

This method has side effects as changes to the element of an array reflects on both the places.

## Cloning Arrays (cont.)

- Consider the following code:

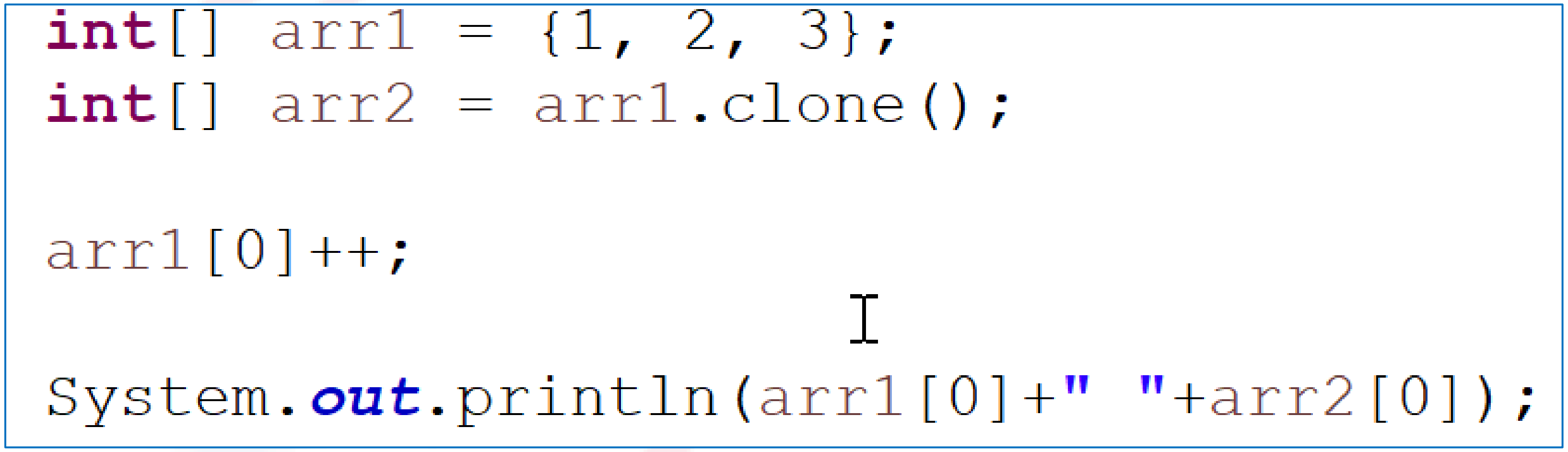
```
int[ ] data = {2, 3, 5, 7, 11, 13, 17, 19};
int[ ] backup;
backup = data;
```



The result of the command backup = data for **int** arrays.

## Cloning Arrays (cont.)

- Using clone() method



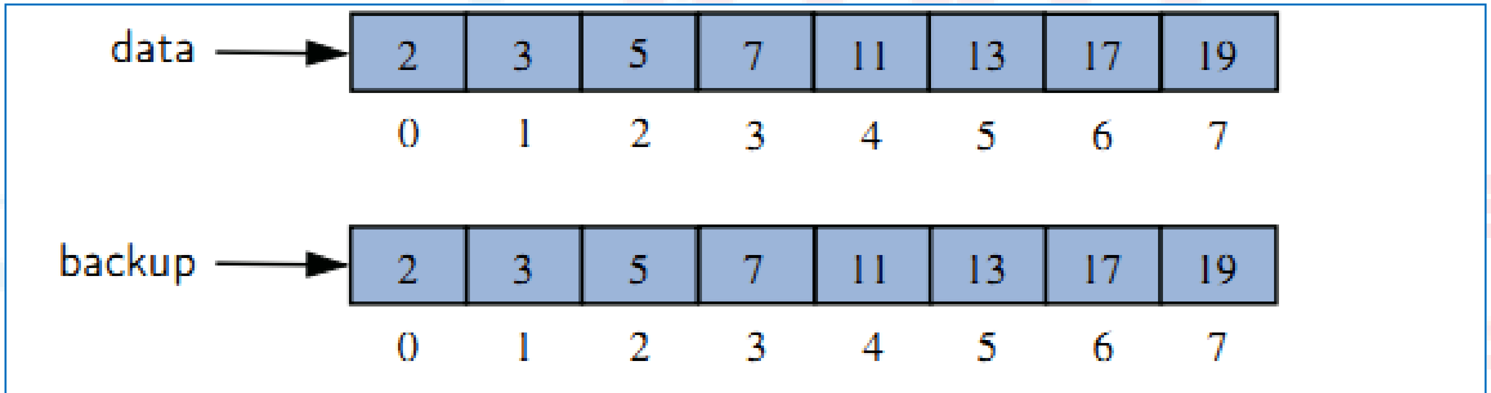
```
int[] arr1 = {1, 2, 3};
int[] arr2 = arr1.clone();

arr1[0]++;

System.out.println(arr1[0]+" "+arr2[0]);
```

Clone methods create a new array of the same size

## Cloning Arrays (cont.)

- Consider the following code:

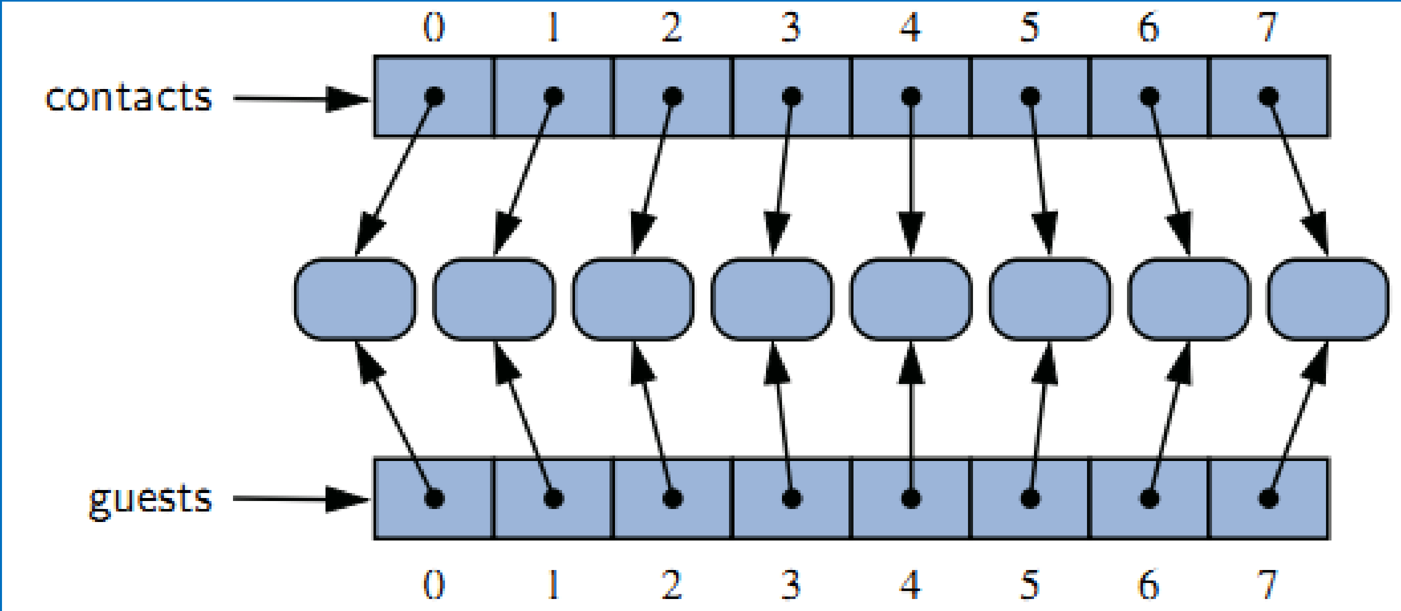
```
int[ ] data = {2, 3, 5, 7, 11, 13, 17, 19};
int[ ] backup;

backup = data.clone( );
```



The result of the command backup = data.clone( ) for **int** arrays.

## Cloning Data Structures

- The result of the command guests = contacts.clone() produces a shallow copy

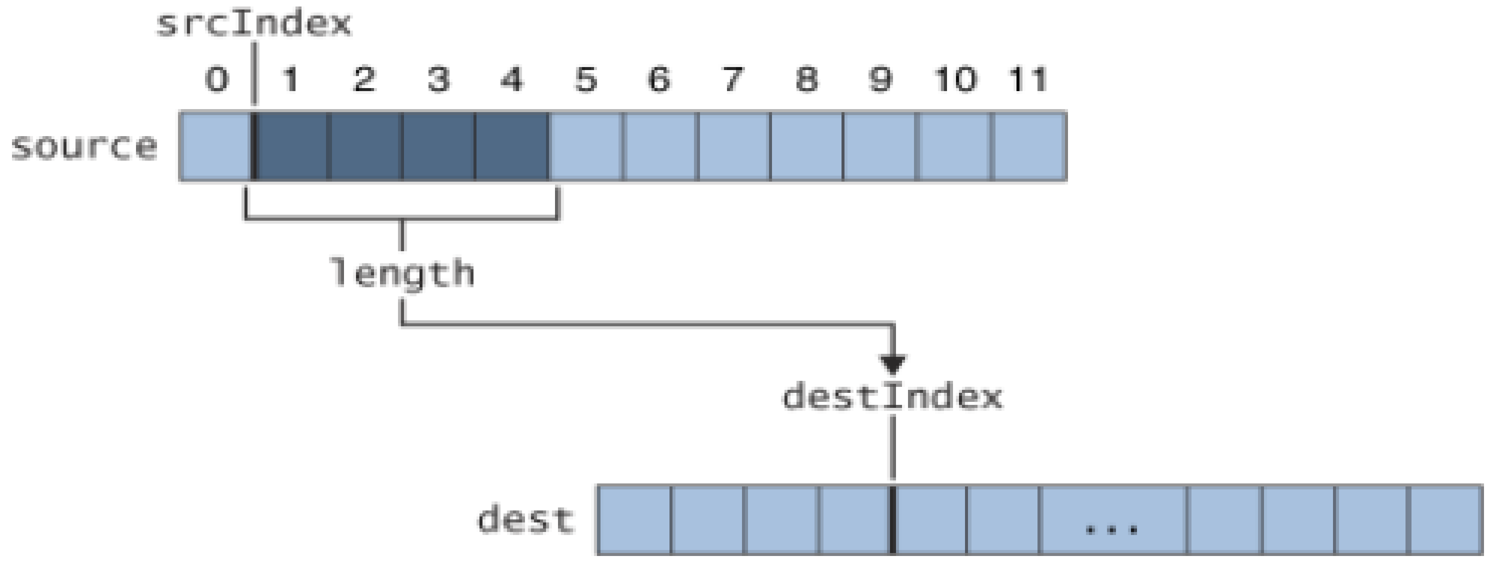


A shallow copy of an array of objects, resulting from the command guests = contacts.clone().

## Cloning Arrays (cont.)

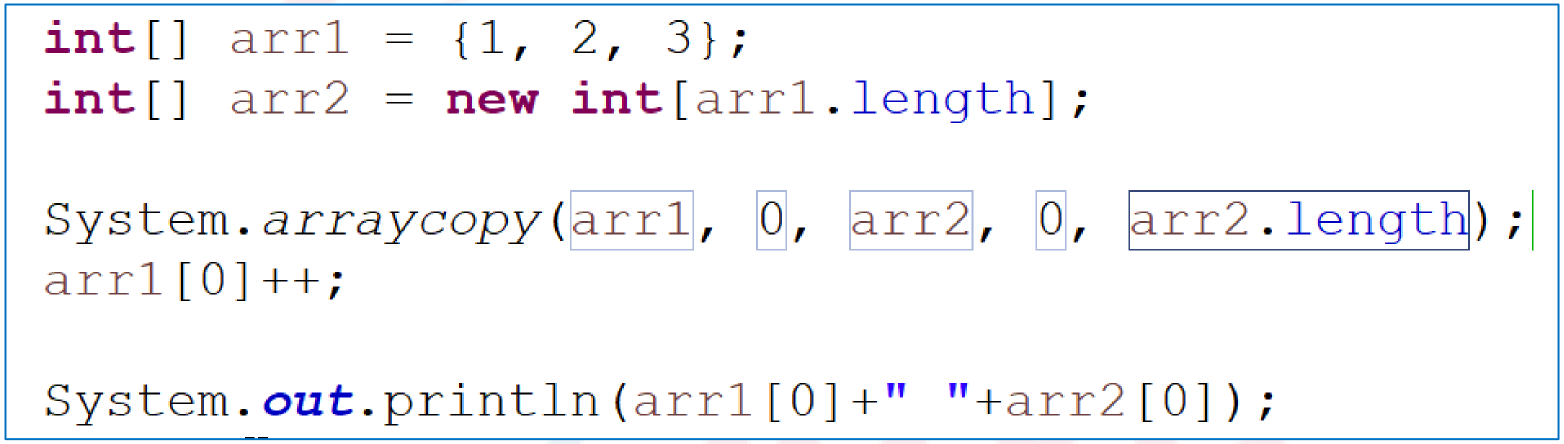
- Using System.arraycopy(…) arraycopy can be used to copy a subset of an array.

public static void arraycopy(Object *source*,int *srcIndex*, Object *dest*,int *destIndex*, int *length*)



## Cloning Arrays (cont.)

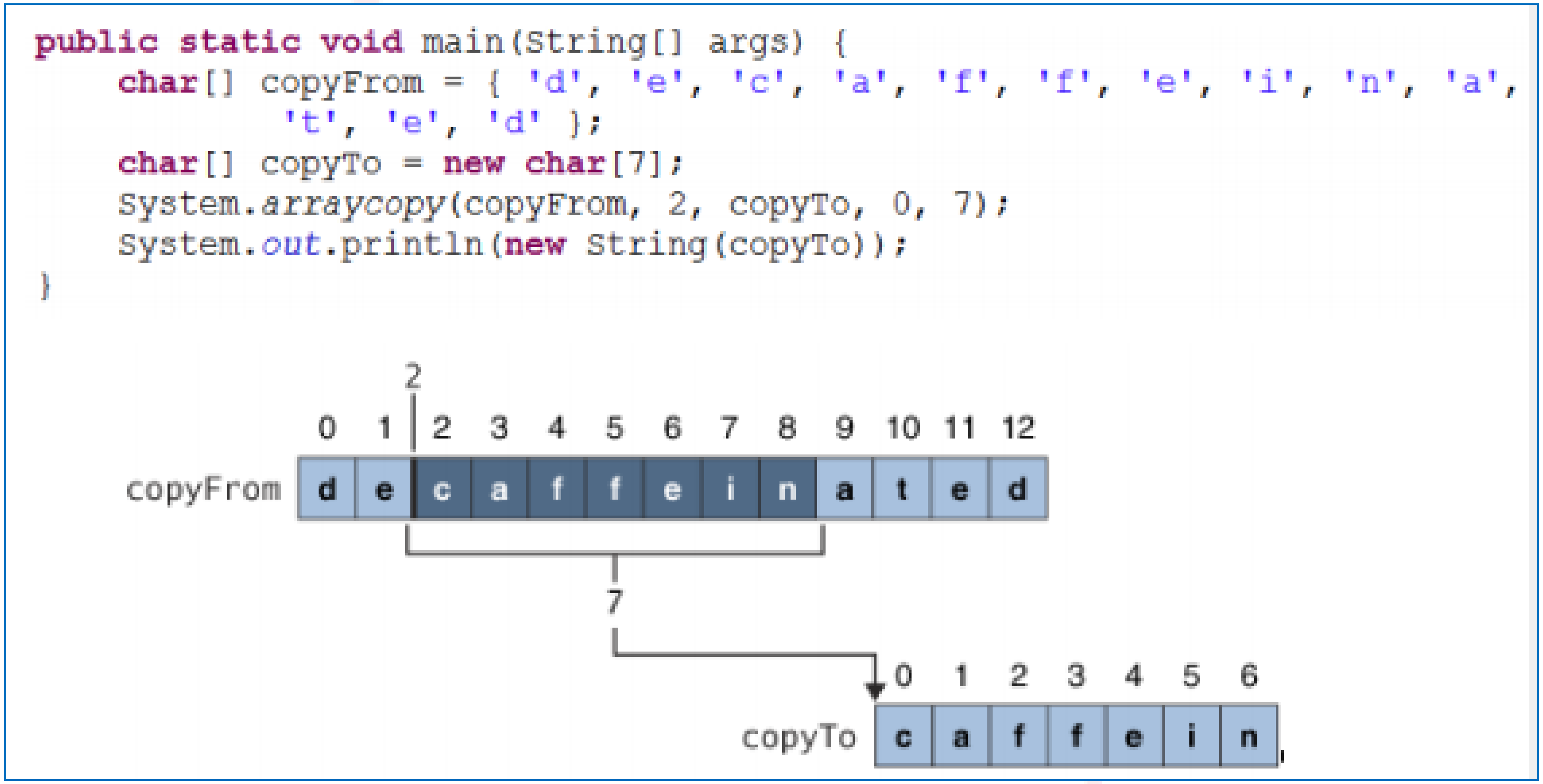
- Using System.arraycopy():



```
int[] arr1 = {1, 2, 3};
int[] arr2 = new int[arr1.length];

System.arraycopy(arr1, 0, arr2, 0, arr2.length);
arr1[0]++;

System.out.println(arr1[0]+" "+arr2[0]);
```

➢ Changes to the element of an array does not effect on the other

# Cloning Arrays (cont.)



```
public static void main(String[] args) {
    char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a',
            't', 'e', 'd' };
    char[] copyTo = new char[7];
    System.arraycopy(copyFrom, 2, copyTo, 0, 7);
    System.out.println(new String(copyTo));
}
```

## Array Exceptions in Java

# Array Exceptions in Java

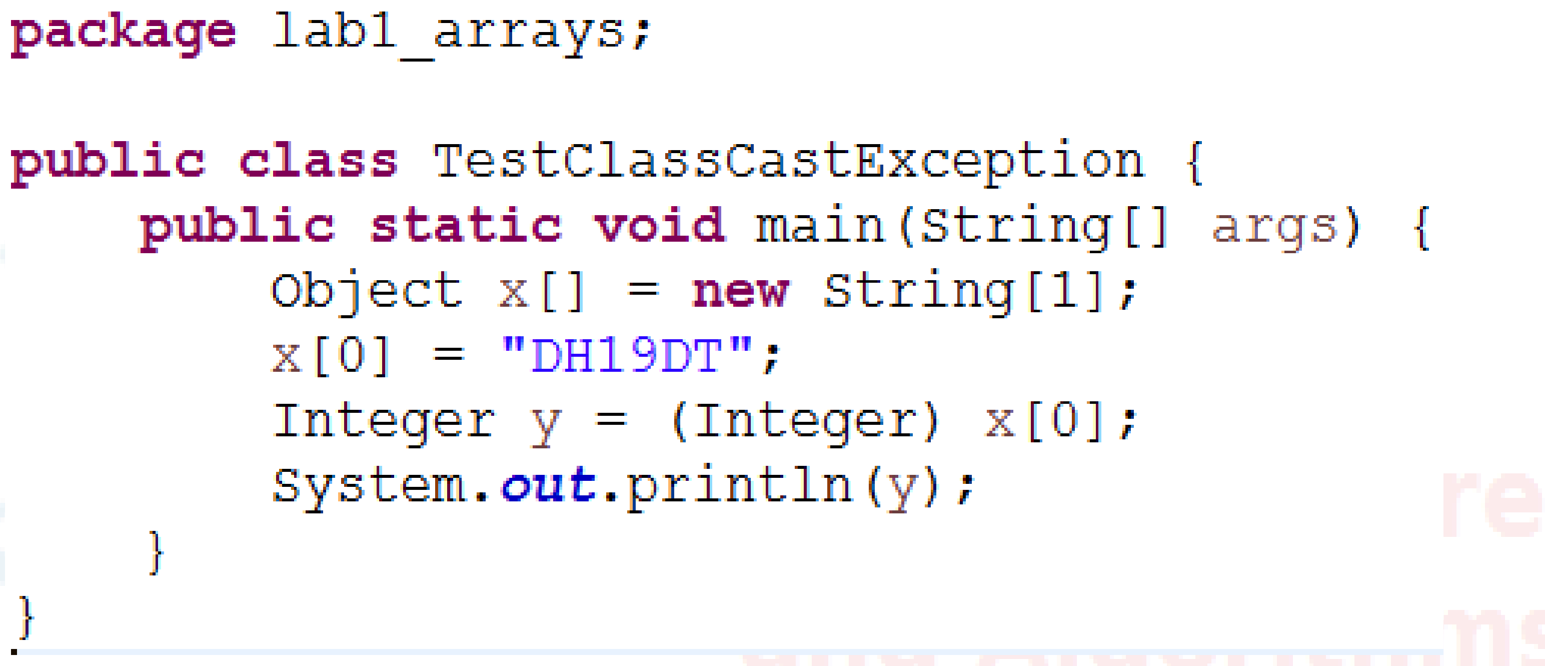
- Possible Error Type in Array:
  - NullPointerException
  - ClassCastException
  - NegativeArraySizeException
  - IndexOutOfBoundsException
  - ArrayIndexOutOfBoundsException
  - ArrayStoreException

## NullPointerException

- Calling the instance method of a null object.
- Accessing or modifying the field of a null object.
- Taking the length of null as if it were an array.
- Accessing or modifying the slots of null as if it were an array.

## ClassCastException

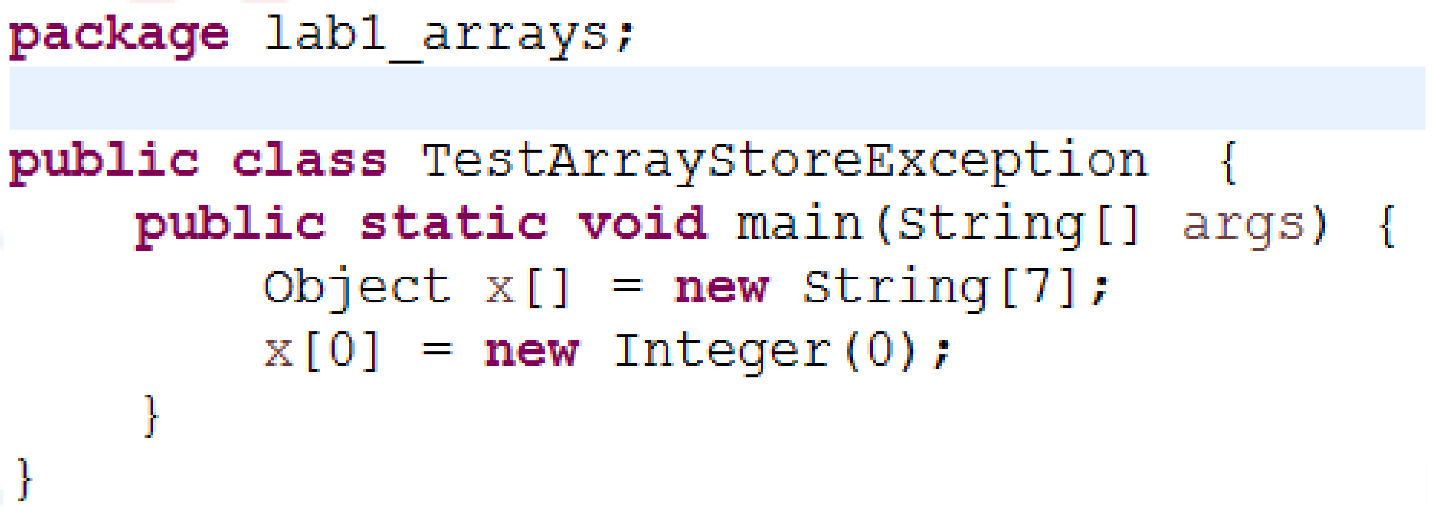
- An attempt has been made to cast an object to a subclass of which it is not an instance



```
package lab1_arrays;

public class TestClassCastException {
    public static void main(String[] args) {
        Object x[] = new String[1];
        x[0] = "DH19DT";
        Integer y = (Integer) x[0];
        System.out.println(y);
    }
}
```

## ArrayStoreException

- An attempt has been made to store the wrong type of object into an array of objects



```
package lab1_arrays;

public class TestArrayStoreException  {
    public static void main(String[] args) {
        Object x[] = new String[7];
        x[0] = new Integer(0);
    }
}
```

## Other exceptions

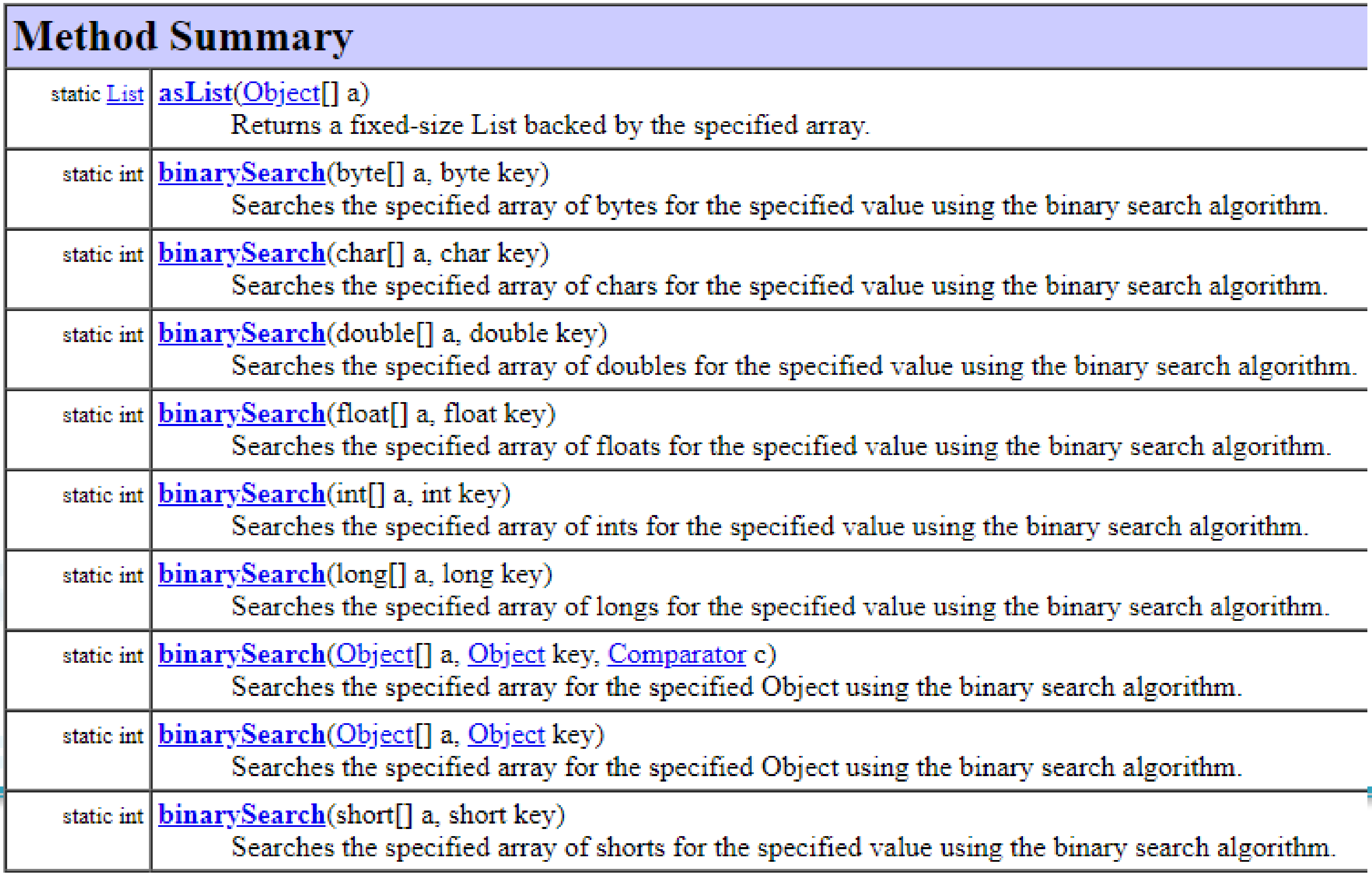
- NegativeArraySizeException:
  - This error is thrown when anyone wants create an array with a negative size
- ArrayIndexOutOfBoundsException:
  - an array has been accessed with an illegal index.
  - Ex. an array is accessed by a negative index or more than the size of the array
  - extends IndexOutOfBoundException
- IndexOutOfBoundsException:
  - This type of exception is thrown by all indexing pattern data types such as an array string and a vector etc. when it is accessed out of the index (range).

## Class java.util.Arrays

- Class Arrays helps you avoid reinventing the wheel by providing static methods for common array manipulations
- Methods:
  - sort(array): Arranges array elements into increasing order.
  - binarySearch(array , element): Determines whether an array contains a specific value and, if so, returns where the value is located.
  - equal(array1, array2): Compares arrays.
  - fill(array , element): Places Values into an array.
  - toString(): Converts array to String.

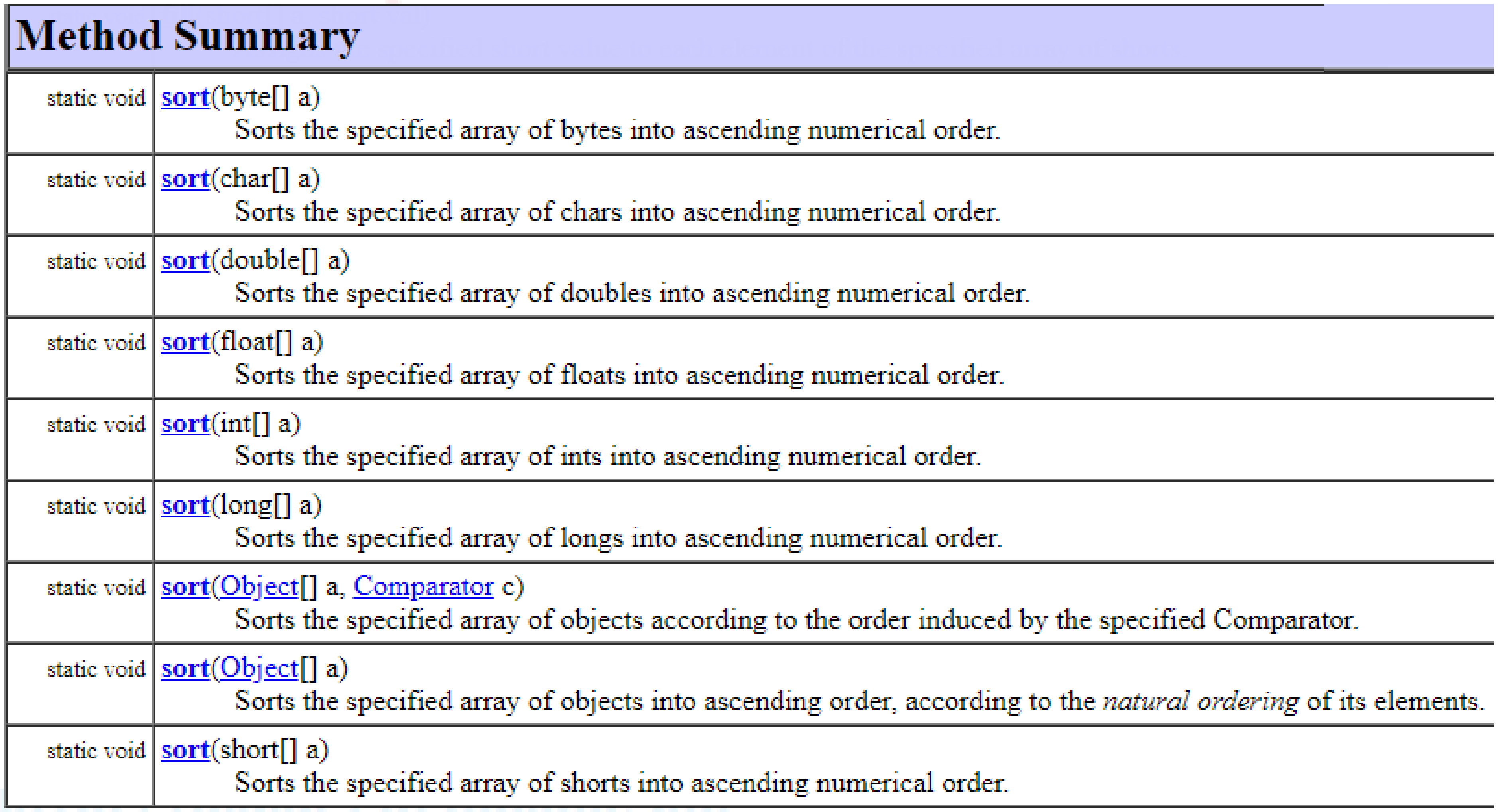
## Class java.util.Arrays

- This class contains various methods for manipulating arrays (such as sorting and searching)



| Method Summary | |
|---|---|
| static List | asList(Object[] a) Returns a fixed-size List backed by the specified array. |
| static int | binarySearch(byte[] a, byte key) Searches the specified array of bytes for the specified value using the binary search algorithm. |
| static int | binarySearch(char[] a, char key) Searches the specified array of chars for the specified value using the binary search algorithm. |
| static int | binarySearch(double[] a, double key) Searches the specified array of doubles for the specified value using the binary search algorithm. |
| static int | binarySearch(float[] a, float key) Searches the specified array of floats for the specified value using the binary search algorithm. |
| static int | binarySearch(int[] a, int key) Searches the specified array of ints for the specified value using the binary search algorithm. |
| static int | binarySearch(long[] a, long key) Searches the specified array of longs for the specified value using the binary search algorithm. |
| static int | binarySearch(Object[] a, Object key, Comparator c) Searches the specified array for the specified Object using the binary search algorithm. |
| static int | binarySearch(Object[] a, Object key) Searches the specified array for the specified Object using the binary search algorithm. |
| static int | binarySearch(short[] a, short key) Searches the specified array of shorts for the specified value using the binary search algorithm. |

# Class java.util.Arrays



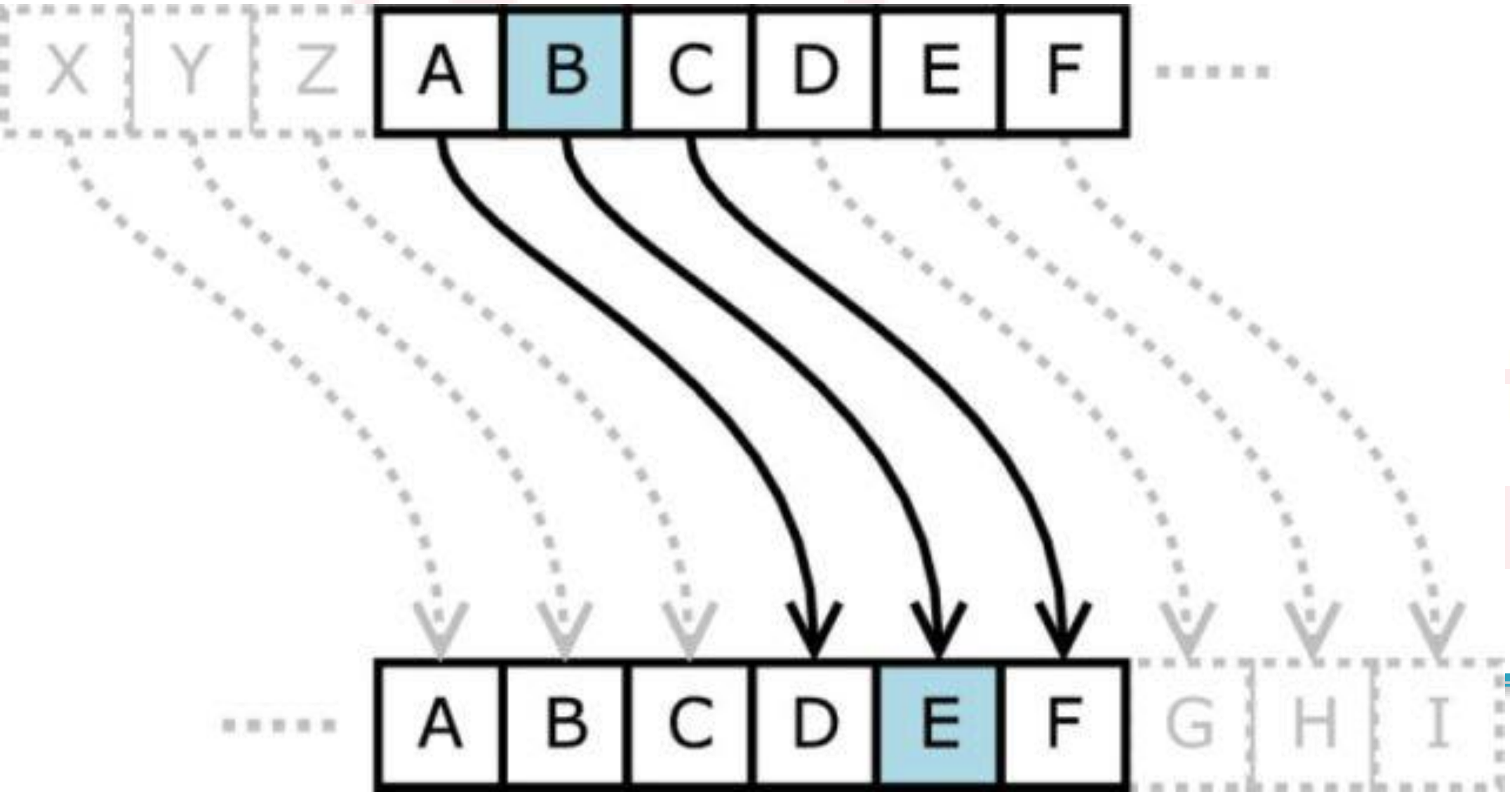
| Method Summary | |
|---|---|
| static void | sort(byte[] a)<br>Sorts the specified array of bytes into ascending numerical order. |
| static void | sort(char[] a)<br>Sorts the specified array of chars into ascending numerical order. |
| static void | sort(double[] a)<br>Sorts the specified array of doubles into ascending numerical order. |
| static void | sort(float[] a)<br>Sorts the specified array of floats into ascending numerical order. |
| static void | sort(int[] a)<br>Sorts the specified array of ints into ascending numerical order. |
| static void | sort(long[] a)<br>Sorts the specified array of longs into ascending numerical order. |
| static void | sort(Object[] a, Comparator c)<br>Sorts the specified array of objects according to the order induced by the specified Comparator. |
| static void | sort(Object[] a)<br>Sorts the specified array of objects into ascending order, according to the *natural ordering* of its elements. |
| static void | sort(short[] a)<br>Sorts the specified array of shorts into ascending numerical order. |

# Exercise 1

- For a given array of integers, implements the following methods to get:
  - The number of even integers in the array
  - The second largest integer in the array
  - The index of the second even integer in the array
  - All odd integers in the array (return an array of integers)

## Exercise 2. Caesar Cipher

- The Caesar Cipher technique is one of the earliest and simplest method of encryption technique.
- A type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet.



## Exercise 2 (cont.)

- For a given **PLAIN TEXT**, how to encrypt it?
- First, transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25.
- Encryption of a letter by a ***shift n*** can be described mathematically as:

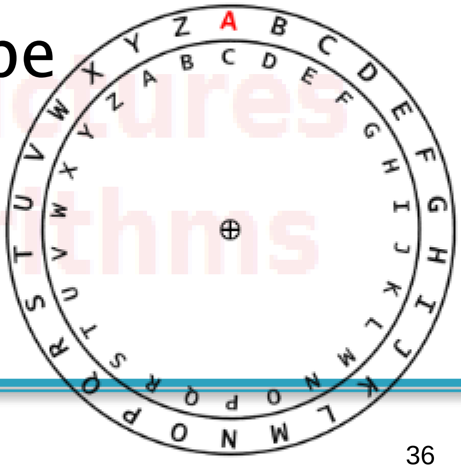
$$E_n(x) = (x + n) mod\ 26$$

(Encryption Phase with shift n)

- Decryption of a letter by a shift n can be described mathematically as:

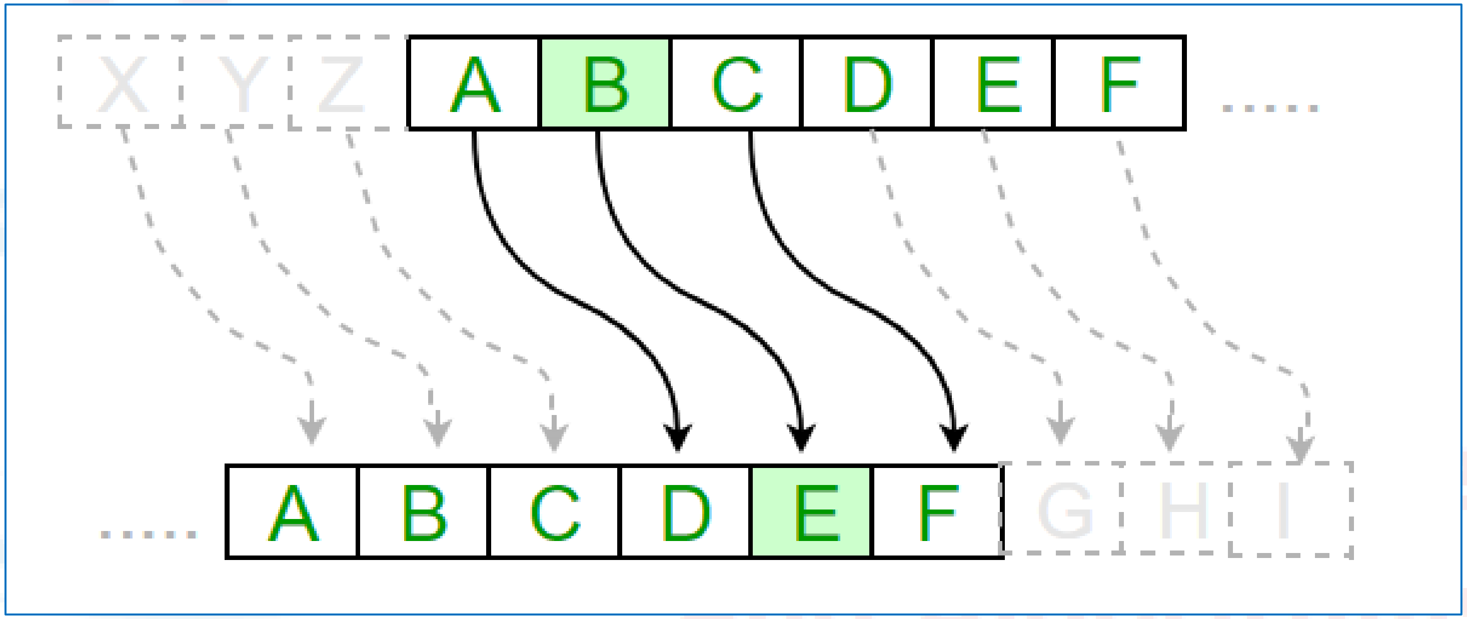
$$D_n(x) = (x - n) mod\ 26$$

(Decryption Phase with shift n)



## Exercise 2 (cont.)

- **Text** : ATTACKATONCE
- **Shift**: 3
- **Cipher**: DWWDFNDWRQFH



https://www.boxentriq.com/code-breaking/keyed-caesar-cipher

FACULTY OF INFORMATION TECHNOLOGY