JAVA Programming

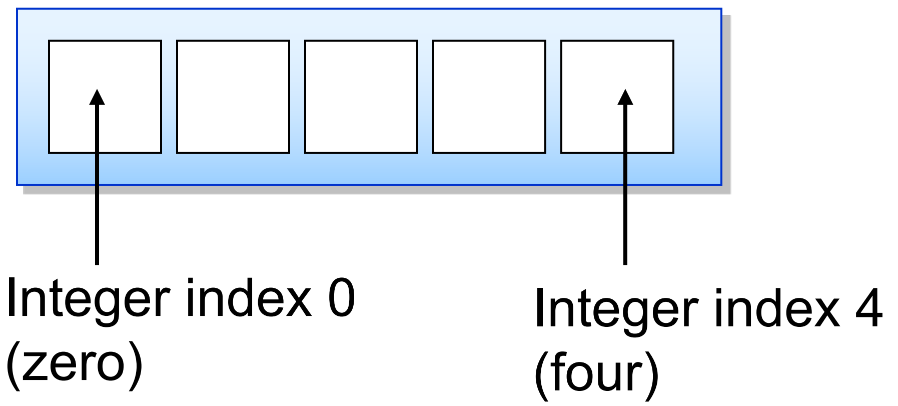
Arrays

Overview

- Array basics
- Array bounds
- Comparing Arrays to Collections
- Using Arrays

Array basics

- An array is a sequence of elements
  - All elements in an array have the same type
  - Individual elements are accessed using integer indexes



Array basics

- An array is a reference-type
- element-type can be any type
- *array of arrays*
- elements are initialized to default initial value for its type
- Length is fixed

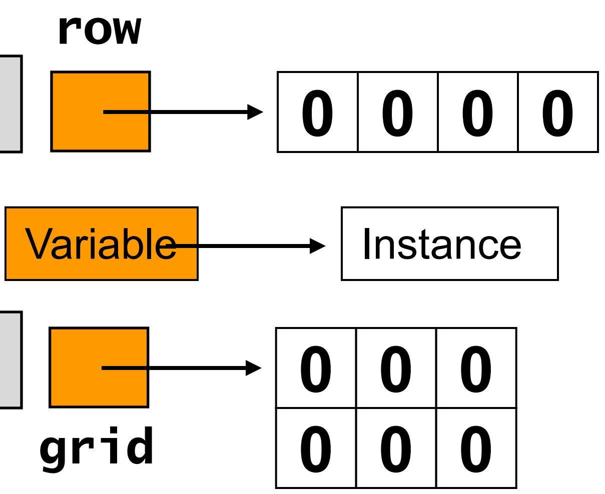
```
int [] myTable;
int [][] myTableOfTables;

myTable = new int[3]
myTableOfTables = new int[5][];
```

Array basics

- Declaring an array variable does **not** create an array!
  – You must use **new** to explicitly create the array instance

```
long[] row = new long[4];
```

```
int[][] grid = new int[2][3];
```



Array basics

- Declare an array:

```
0  1  2  3  4  5  6
```

7 elements

```java
int[] numbers = {8, 5, 3, 9, 0, 2, 7};
```

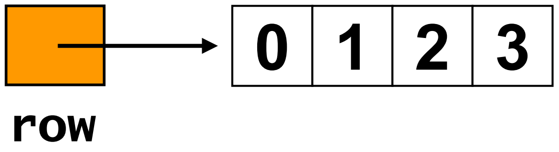
```java
System.out.println(numbers[2]);
```

- Arrays are ZERO based!

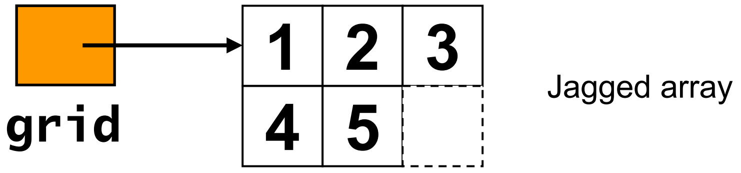
Array basics

- array initializer:

```
int[ ] row = {0, 1, 2, 3};
//or
int[ ] row = new int[] {0, 1, 2, 3};
```



```
int[][] grid= { { 1, 2, 3 }, { 4, 5 } };
```



Array bounds

- What's the output of:

```
int[] numbers = {8, 5, 3, 9, 0, 2, 7};
```

```
System.out.println(numbers[7]);
```

- Java error: java.lang.ArrayIndexOutOfBoundsException
- Bound checking at runtime

Array Bounds

- All array access attempts are bound checked
  - A bad index throws a java.lang.ArrayIndexOutOfBoundsException
  - Use the **length** field

```
int[] row=new int[]{1,2,3,4};
for (int i = 0; i < row.length; i++) {
        System.out.println(row[i]);
}
```

```
int[ ][ ] grid = new int[ ][ ] { { 1, 2, 3, 4 }, { 5, 6 }, { 7, 8, 9 } };
for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[i].length; j++) {
                System.out.printf("%d ", grid[i][j]);
        }
        System.out.println();
}
```

Comparing Arrays to Collections

- An array cannot resize itself
  - A collection class can resize
- In general, arrays are faster but less flexible than collections
  - Collections are slightly slower but more flexible

Creating a Computed Size Array

- The array size does not need to be a compile-time constant
  - Any valid integer expression will work
  - Accessing elements is equally fast in all cases
    - Array size specified by compile-time integer constant:

```
long[ ] arrayOfLongs = new long[4]
```

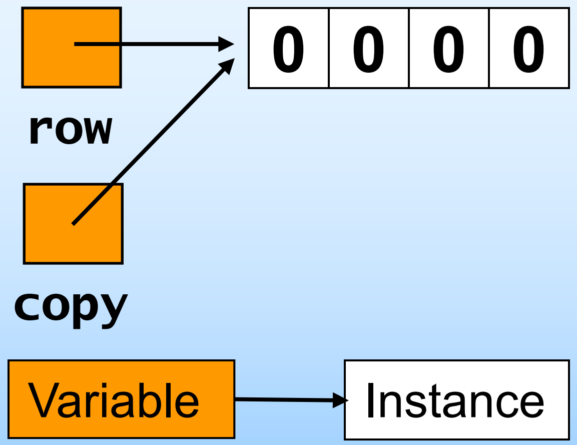
    - Array size specified by run-time integer value:

```
int rows=5
int columns=3
long[ ] flatten= new long [rows * columns]
```

Copying Array Variables

- Copying an array variable copies the array variable only
  - It does not copy the array instance
  - Two array variables can refer to the same array instance

```
long[] row=new long[4];
long[] copy=row;

row[0]++;

System.out.println(copy[0]);
```



Using Arrays

- Returning Arrays from Methods
- Passing Arrays as Parameters
- Command-Line Arguments
- Using Arrays with foreach

Returning Arrays from Methods

- You can declare methods to return arrays

```
int[] intArr=createArray(10, 50);
```

```
public static int[] createArray(int length,int upperBound) {
        Random random=new Random();
        int[] tempArr=new int[length];
        for (int i = 0; i < tempArr.length; i++) {
                tempArr[i]=random.nextInt(upperBound)+1;
        }
        return tempArr;
}
```

Passing Arrays as Parameters

- An array parameter is a copy of the array variable
  - Not a copy of the array instance

```
double[] accountBalance = { 10000, 20000, 5000 };
addInterest(accountBalance);
```

```
public static void addInterest(double[] accounts) {
       for (int i = 0; i < accounts.length; i++) {
              accounts[i] *= 1.04;
       }
}
```

This method will modify the original array instance

Command-Line Arguments

- The runtime passes command line arguments to main
  - **main** can take an array of strings as a parameter
  - The name of the program is not a member of the array

```
public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
                System.out.println(args[i]);
        }
```

Using Arrays with foreach

- The foreach statement abstracts away many details of array handling

```
public static void main(String[] args) {
        for (String arg : args) {
                System.out.println(arg);
        }
}
```

Dynamic Arrays

- java.util. contains several dynamic collections like
  - ArrayList
  - Dictionary
  - HashMap
  - Stack etc.

Dynamic Arrays

```
public static void main(String[] args) {
        ArrayList al=new ArrayList();
        al.add(2);
}
```

Lab 6: Creating and Using Arrays