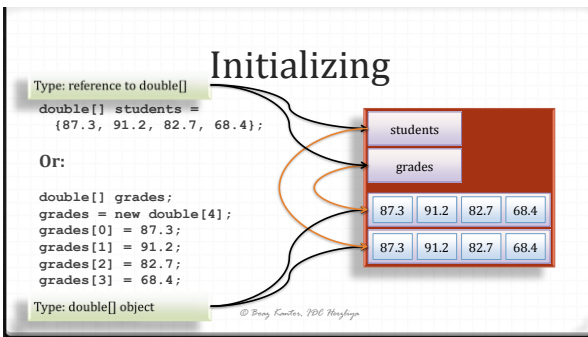


Syntax

- Declaring:
`<type>[] <name>;`
`double[] vector;`
- Initializing:
`<name> = new <Type>[<number_of_cells>;`
`vector = new double[63];`
 ◦ Constant initialization:
`<Type>[] <name> = {<cell_value_1>, <cell_value_2>, ...};`
`double[] vector = {12.0, 13.52, -1.444};`
- Accessing:
 ◦ Array object:
`<name>.<member>;`
`vector.length;`
 ◦ Array cell:
`<name>[<cell_number>]`
`vector[2] = 14.8;`
`double element = vector[1];`

© Barry Kantner, PDC Maryland



```

class Average {
    public static void main(String[] args) {
        double[] xy = new double[2];
        xy[0] = 4.2;
        xy[1] = 4.8;
        System.out.println("The average of " + xy[0] + " and " + xy[1] + " is: " + average(xy));
        double[] xyz = {xy[0], xy[1], 2.1};
        System.out.println("The average of " + xyz[0] + ", "
            + xyz[1] + " and " + xyz[2] + " is: "
            + average(xyz));
    }

    public static double average(double[] numbers) {
        if (numbers == null) {
            throw new IllegalArgumentException();
        }
        if (numbers.length == 0) {
            return 0;
        }
        double sum = 0;
        for (int currCell = 0; currCell < numbers.length; ++currCell) {
            sum += numbers[currCell];
        }
        return sum / numbers.length;
    }
}
    
```

© Bruce Kamstra, PDC, Haverly

Class Arrays

- o Arrays of classes
- o Initializing the array does not initialize its cells!

```

Turtle[] bale = new Turtle[3];
bale[0] = new Turtle();
bale[0].tailDown();
    
```

NullPointerException !!

© Bruce Kamstra, PDC, Haverly

Multi Dimensional Arrays

- o Declaring:


```

<Type>[] <name>;
double[] vector;
double[][] matrix;
double[][][] cube;
double[][][][][] omniDimensionalArray;
            
```

```

int[][] board = new int[10][10];
for (int i = 1; i <= 10; ++i) {
    for (int j = 1; j <= 10; ++j) {
        board[i][j] = i*j;
    }
}
    
```

Where is the bug?
What 2 approaches can fix it?

© Bruce Kamstra, PDC, Haverly

Multi Dimensional Arrays – Exercise

Define a class `TravelLengths` which provides information services about distances between cities.

API:

```
public TravelLengths(String[] cityNames, int[][] distances);
public int getDistance(String from, String to);
```

Sample use:

```
String[] cities = {"Tel Aviv", "Netanya", "Haifa"};
int[][] distances = {{0, 30, 85}, {30, 0, 55}, {85, 55, 0}};
TravelLengths t = new TravelLengths(cities, distances);
System.out.println("The distance from Haifa to Tel Aviv is " +
    t.getDistance("Haifa", "Tel Aviv") + " km");
```

	Tel Aviv	Netanya	Haifa
Tel Aviv	0	30	85
Netanya	30	0	55
Haifa	85	55	0

© Brany Kunitz, PDC Herzliya

Multi Dimensional Arrays – Solution

```
public class TravelLengths {
    private String[] cityNames;
    private int[][] distances;
    public TravelLengths(String[] cityNames, int[][] distances) {
        this.cityNames = cityNames; // should we copy the array?
        this.distances = distances; // should we copy the array?
        // array copying is described next
    }
    public int getDistance(String from, String to) {
        // search for the desired cities
        int fromIndex = 0;
        int toIndex = 0;
        while (!cityNames[fromIndex++].equals(from));
        while (!cityNames[toIndex++].equals(to));
        return distances[fromIndex - 1][toIndex - 1];
    }
}
```

© Brany Kunitz, PDC Herzliya

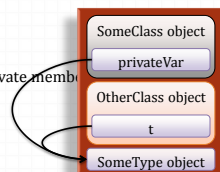
Copying Arrays – Encapsulation

When we have a reference as a private member:

```
class SomeClass {
    private SomeType privateVar;
    public SomeClass(SomeType other) {
        privateVar = other;
    }
}
```

The caller keeps a reference to our private member:

```
class OtherClass {
    SomeType t;
    private static void foo() {
        t = new SomeType("I have $10 in the bank");
        SomeClass c = new SomeClass(t);
        t.setBalance(1000000);
    }
}
```

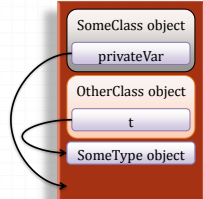


© Brany Kunitz, PDC Herzliya

Copying Arrays – Encapsulation

To solve this, we create our own copy:

```
class SomeClass {
    private SomeType privateVar;
    public SomeClass(SomeType other) {
        // using copy-constructor:
        privateVar = new SomeType(other);
        // using clone() method:
        privateVar = other.clone();
        // using assignments:
        privateVar = new SomeType();
        privateVar.setName(other.getName());
        privateVar.setBalance(other.getBalance());
    }
}
```



How do we do that with arrays?

© Bruce Kenton, PDC, Hayslope

Copying Arrays

Either:

```
class SomeClass {
    private int[] privateVar;
    public SomeClass(int[] other) {
        // using System.arraycopy:
        privateVar = new int[other.length];
        System.arraycopy(other, 0, this.privateVar, 0, other.length);
        // using clone() method:
        privateVar = other.clone();
        // using assignments:
        privateVar = new int[other.length];
        for (int curIndex = 0; curIndex < privateVar.length; ++curIndex) {
            privateVar[curIndex] = other[curIndex];
        }
    }
}
```

Every array has a clone() method!

How do we do that with multi-dimensional arrays?

© Bruce Kenton, PDC, Hayslope

Copying Multi-Dimensional Arrays

```
public class TravelLengths {
    private String[] cityNames;
    private int[][] distances;

    // either of the following:
    public TravelLengths(String[] cityNames, int[][] distances) {
        this.cityNames = new String[cityNames.length];
        System.arraycopy(cityNames, 0, this.cityNames, 0, cityNames.length);
        this.distances = new int[distances.length][distances[0].length];
        for (int curDistance = 0; curDistance < distances.length; ++curDistance) {
            System.arraycopy(distances[curDistance], 0,
                this.distances[curDistance], 0, distances.length);
        }
    }

    Or:
    public TravelLengths(String[] cityNames, int[][] distances) {
        this.cityNames = cityNames.clone();
        this.distances = distances.clone();
    }
}
```

© Bruce Kenton, PDC, Hayslope

Multi Dimensional Arrays – Exercising

- ◊ How to solve:
 1. Write the skeleton (called "a stub"):
 1. All the needed method definitions
 2. Documentation
 3. Trivial implementation:

```
◊ return 0;  
◊ return false;  
◊ throw new NotImplementedException("NotImplemented"); //< Recommended!
```
 2. Write a test Main method (called "a driver")
- 3. Decide on data structures:
 1. Add as instance variables
 2. Add getters/setters where needed
- 4. Write the solution as pseudo-code comments
- 5. Divide to methods where needed
- 6. Implement, test and fix. And test and fix and test (and fix, and test). Then fix.
- 7. Test.

© Boaz Kantor, PDC Herzliya

Boaz Kantor
Introduction to Computer Science,
Fall semester 2009-2010
IDC Herzliya

Arrays

© Boaz Kantor, PDC
Herzliya
