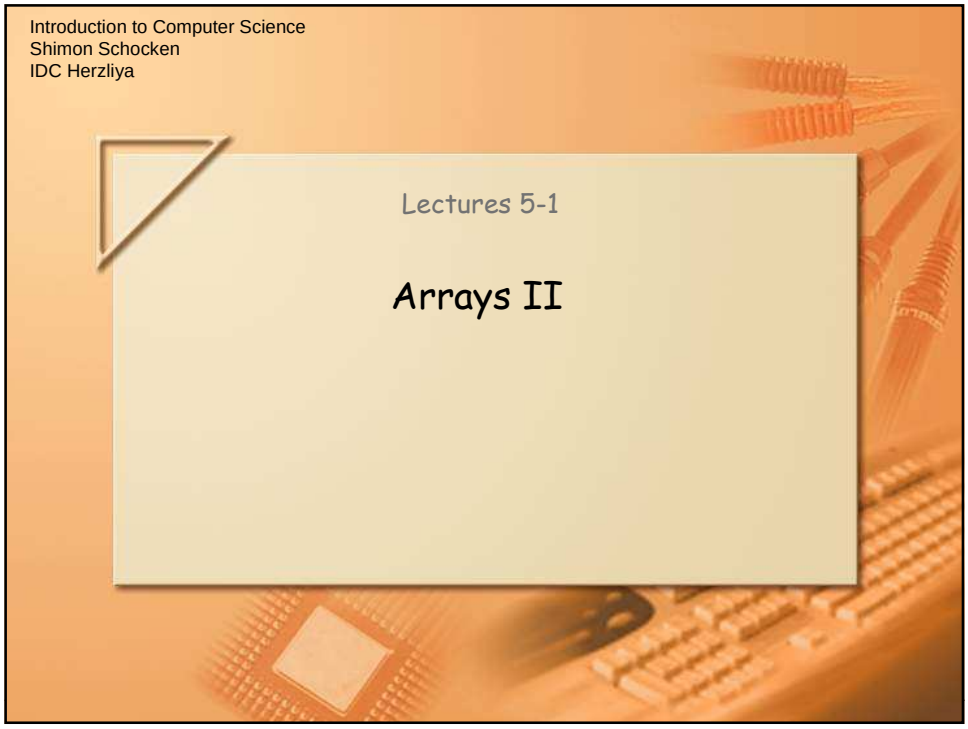


Array processing: best practice advice

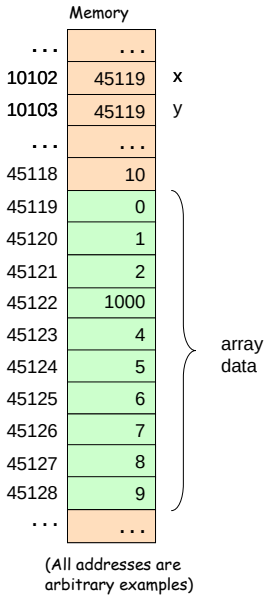
```
char[] x;
// Let's assume that x has been populated with data.

// Traditional array processing logic
for (int i = 0; i < x.length; i++)
    System.out.println(x[i]);


// Use this when the computation of the termination condition is costly
for (int i = 0, n = costlyComputation(); i < n; i++)
    System.out.println(x[i]);


// For...each style: Use this whenever possible
for (char c : x)
    System.out.println(c);


// Avoid while loops -- the variable i remains in scope and can cause
// trouble later
int i = 0;
while (i < x.length)
    System.out.println(x[i++]);
}
```

Outline

- How arrays are implemented in Java
- Arrays as parameters
- Arrays of objects
- Array processing examples
  - Polynomial
  - Sorting
  - Merging

Behind the scene view of arrays

```
int[] x = new int[10];
for (int j = 0; j < x.length; j++) {
    x[j] = j;
}

int[] y = x;
y[3] = 1000;

System.out.print(x[3]);
// Will print 1000.
```



(All addresses are arbitrary examples)

A peek into the compiler's working:

- `int[] x = new int[10]`: This command generates low-level code that asks the OS to allocate a memory block that can accommodate 10 integers. Next, the code assigns the base address of this block to the variable `x`.
- `int[] y = x;` This command generates low-level code that assigns the contents of `x` (a memory address) to `y`.

Arrays as parameters

```
// A static class providing various services
// that help process arrays of integers
public class IntArrays {

   public static boolean containsZero (int[] a) {
      for (int x : a)
          if (x == 0)
            return true;
      return false;
    }


    public static int sum (int[] a) {
       int sum = 0;
       for (int x : a)
           sum = sum + x;
       return sum;
    }

    // More static methods
    // for processing
    // arrays of integers.
}
```

Example of a "helper" static class, designed to help process arrays

Side comment:
- It would be nice if we could generalize this class to handle arrays of *any* numeric type
- Can be done using "generics" (later in the course)

Output:

```
45
true
false
```

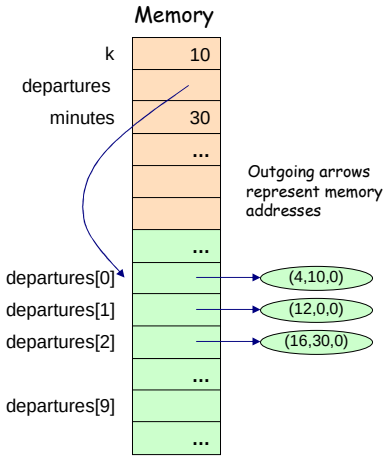
client

```
public class ArrayDemo {
  public static void main(String[] args) {
    int[] x = new int[10];
    int[] y = new int[10];
    for (int j = 0; j < x.length; j++) x[j] = j;
    for (int j = 0; j < y.length; j++) y[j] = j + 1;
    System.out.println(IntArrays.sum(x));
    System.out.println(IntArrays.containsZero(x));
    System.out.println(IntArrays.containsZero(y));
  }
```


Array of objects

Specification: Create a sequence of *departures*. Each departure is represented by a `Time` object whose data is `(hours, minutes, seconds)`

```
int k = 10;
Time[] departure = new Time[k];
departure[0] = new Time(4,10,0);
departure[1] = new Time(12,0,0);
Departure[2] = new Time(16,30,0);
// ...
int minutes = departure[2].getMinutes();
```

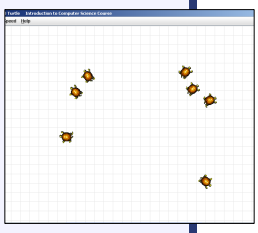


stack — Memory area that holds the variables of running methods

heap — Memory area that holds object and array data

Array of objects: example

```
// Creates some turtles and sends them to move in random directions.
public class TurtleJungle {
    static final int STEP = 20;
    static final int N_TURTLES = 7;

    public static void main(String[] args) {
        Turtle[] turtles = new Turtle[N_TURTLES];
        for (int i = 0; i < turtles.length; i++) {
            turtles[i] = new Turtle();
            turtles[i].turnLeft((int) (Math.random() * 360));
            turtles[i].tailUp();
        }

        while (true)
            for (Turtle t : turtles)
                t.moveForward(STEP);
        }
 }
```



Outline

- How arrays are implemented in Java
- Arrays as parameters
- Arrays of objects
- Array processing examples
  - Polynomial
  - Sorting
  - Merging

Polynomials

A Polynomial function, aka "Polynomial", is a real-valued function of the following form:

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots c_1 x + c_0$$

Where n is a nonnegative integer, called the polynomial *degree*, $c_n, \ldots, c_0$ are called *coefficients*, and x is the Polynomial *argument*, i.e. the number at which the Polynomial is evaluated.

Examples of some polynomials:

$5x^2 - 17x + 2$ $\quad$ $x^3 - 27$ $\quad$ $3x$ $\quad$ 2

Note that a polynomial is completely characterized by its coefficients.
For example, the first polynomial is characterized by `(2, -17, 5)`

We wish to write a Java class that represents and manipulates polynomials with integer coefficients. In particular, we want to be able to create polynomials, evaluate them, add them, print them, and so on.

We start by making some design decisions:

- Each polynomial will be represented by an immutable object
- The object's data will be the polynomial's coefficients, and its degree

Polynomial specification

```
// An immutable polynomial with integer coefficients c0, c1, c2, ...
// representing the polynomial c0 + c1*x + c2*x^2 ...
public class Polynomial {

    // Constructs a new polynomial using the supplied coefficients.
    public Polynomial (int[] coefficients)

    // Returns the coefficient of this polynomial's k'th term
    public int getCoefficient (int k)

    // Returns the value of this polynomial at point x
    public double valueAt (double x)

    // Returns the degree of this polynomial
    public int getDegree ()

    // Returns the polynomial resulting from the addition of this polynomial and p
    public Polynomial plus (Polynomial p)

    // Displays this polynomial as a string of the form an*x^n + ... + a1*x + a0
    public String toString ()
}
```

Using Polynomial

```
// An immutable polynomial with integer
// coefficients c0, c1, c2, ...
// representing the polynomial
// c0 + c1*x + c2*x^2 ...
public class Polynomial {

   // Constructs a new polynomial using
   // the supplied coefficients.
   public Polynomial (int[] coefficients)

   // Returns the coefficient of this
   // polynomial's k'th term
   public int getCoefficient (int k)

   // Returns the value of this
   // polynomial at point x
   public double valueAt (double x)

   // Returns the degree of this polynomial
   public int getDegree ()

   // Returns the polynomial resulting from the
   // addition of this polynomial and p
   public Polynomial plus (Polynomial p)

   // Displays this polynomial as a string of the
   // form an*x^n + ... + a1*x + a0
   public String toString ()
}
```

client

```
//  p(x) = 4x^3 + 2x^2 - 3x + 1
int[] pCoefficients = {1, -3, 2, 4};
Polynomial p = new Polynomial(pCoefficients);

System.out.println(p);
System.out.println("p(2) = " + p.valueAt(2) + "\n");

//  q(x) = x^4 + 2x - 7
int[] qCoefficients = {-7, 2, 0, 0, 1};
Polynomial q = new Polynomial(qCoefficients);

System.out.println(q);
System.out.println("q(3) = " + q.valueAt(3) + "\n");

Polynomial pPlusq = p.plus(q);
System.out.println(pPlusq);
System.out.println("pPlusq(2) = " + pPlusq.valueAt(2));
```

```
D:\demo>java PolynomialDemo
4x^3 + 2x^2 - 3x + 1
p(2) = 35.0

1x^4 + 2x - 7
q(3) = 80.0

1x^4 + 4x^3 + 2x^2 - 1x - 6
pPlusq(2) = 48.0
```


Polynomial implementation

Polynomial class

```
public class Polynomial {

    // The polynomial's coefficients are kept in an array:
    // The coefficient of x^i is stored in coefficients[i].
    // The coefficient's degree is kept in n.
    private int[] coefficients;
    private int n;

    // Constructs this polynomial from given coefficients
    public Polynomial(int[] coefficients) {
        this.n = coefficients.length;
        this.coefficients = new int[n];
        System.arraycopy(coefficients, 0, this.coefficients, 0, n);
    }

    // Returns the coefficient of this polynomial's k'th term
    public int getCoefficient(int k) {
        return k < n ? coefficients[k] : 0;
    }

   // More methods in the next slides
}
```

System.arraycopy: Copies n entries from a source array to a destination array, beginning and ending at specified positions.

More readable than a for loop.

client

```
//  p(x) = 4x^3 + 2x^2 - 3x + 1
int[] pCoefficients = {1, -3, 2, 4};
Polynomial p = new Polynomial(pCoefficients);
```

Polynomial implementation (cont.)

Polynomial class

```
// Displays this polynomial as a string
public String toString() {
    int degree = getDegree();
    if (degree ==  0) return "" + coefficients[0];
    if (degree ==  1) return coefficients[1] + "x + " + coefficients[0];

   String s = coefficients[degree] + "x^" + degree;
    for (int i = degree-1; i >= 0; i--) {
        if      (coefficients[i] == 0) continue;
        else if (coefficients[i]  > 0) s = s + " + " + ( coefficients[i]);
        else if (coefficients[i]  < 0) s = s + " - " + (-coefficients[i]);
        if      (i == 1) s = s + "x";
        else if (i >  1) s = s + "x^" + i;
    }
    return s;
  }
}
```

client

```
int[] pCoefficients = {1, -3, 2, 4};

Polynomial p = new Polynomial(pCoefficients);

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

```
D:\demo>java PolynomialDemo
4x^3 + 2x^2 - 3x + 1
```


Polynomial implementation (cont.)

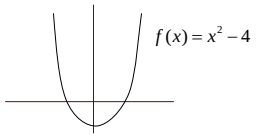
Polynomial class

```
// Returns the value of this polynomial at a given point.
public double valueAt(double x) {
    int value = 0;
    int power = 1;
    for (int c : coefficients) {
        value += c * power;
        power *= x;
    }
    return value;
  }
```

client

```
//  f(x) = x^2 - 4
int[] fCoefficients = {-4, 0, 1};
Polynomial f = new Polynomial(fCoefficients);
System.out.println("f(x) = "+ f + "\n");

for (int x = -9; x <= 9; x++) {
    double y = f.valueAt(x);
    System.out.print("f("+ x +")= " + y);
    if (y == 0) System.out.print("  root");
    System.out.println();
}
```

```
D:\demo>java PolynomialDemo
f(x) = 1x^2 - 4

f(-9)= 77.0
f(-8)= 60.0
f(-7)= 45.0
f(-6)= 32.0
f(-5)= 21.0
f(-4)= 12.0
f(-3)= 5.0
f(-2)= 0.0  root
f(-1)= -3.0
f(0)= -4.0
f(1)= -3.0
f(2)= 0.0  root
f(3)= 5.0
f(4)= 12.0
f(5)= 21.0
f(6)= 32.0
f(7)= 45.0
f(8)= 60.0
f(9)= 77.0
```



$f(x) = x^2 - 4$

Polynomial implementation (cont.)

Polynomial class

```
// Returns the degree of this polynomial.
public int getDegree() {
    int degree = n - 1;
    while (coefficients[degree] == 0) degree--;
    return degree;
  }

// Returns the polynomial resulting from
// the addition of this polynomial and another one.
public Polynomial plus(Polynomial p) {
    int n = 1 + Math.max(getDegree(), p.getDegree());
    int[] sum = new int[n];
    for (int i = 0; i < n; i++)
       sum[i] = getCoefficient(i)
                + p.getCoefficient(i);
    return new Polynomial(sum);
  }
```

$$p(x) = 4x^3 + 2x^2 - 3x + 1$$

$$q(x) = x^4 + 2x - 7$$

$$(p+q)(x) = ?$$

Polynomial addition is done by adding like terms

client

```
//  p(x) = 4x^3 + 2x^2 - 3x + 1
int[] pCoefficients = {1, -3, 2, 4};

Polynomial p = new Polynomial(pCoefficients);
System.out.println(p);

//  q(x) = x^4 + 2x - 7
int[] qCoefficients = {-7, 2, 0, 0, 1};

Polynomial q = new Polynomial(qCoefficients);
System.out.println(q);

System.out.println(p.plus(q));
```

```
D:\demo>java PolynomialDemo
4x^3 + 2x^2 - 3x + 1
1x^4 + 2x - 7
1x^4 + 4x^3 + 2x^2 - 1x - 6
```


Outline

- How arrays are implemented in Java
- Arrays as parameters
- Arrays of objects
- Array processing examples
  - Polynomial
  - Sorting
  - Merging

Sorting

The sorting task:

- Input: an array of values, e.g. `(5,2,10,8,5)`
- Output: a sorted array, e.g. `(2,5,5,8,10)`
- The values can be any object whose data type has an order
- Sorting is a common computational task:
  - Sort your contacts list
  - Sort pages by their pageRank
  - Sorting before searching

We will show one sorting algorithm, applied to integers

Later in the course:

- More sorting algorithms
- Java implementations designed to sort any type data.

Selection sort

Pseudo code:

```
n = array.length - 1

step 0:
  k = index of the maximum in the range 0 ... n
  switch a[k] with a[n]

step 1:
  k = index of the maximum in the range 0 ... n-1
  switch a[k] with a[n-1]

step j:
  k = index of the maximum in the range 0 ... n-j
  switch a[k] with a[n-j]
```

Basic idea:

In the `j`th step, find the `j`th largest number and put it in the `n-j` location of the array.

Unsorted:	3	10	4	8	15	1	7	4
Step 0:	3	10	4	8	4	1	7	15
Step 1:	3	7	4	8	4	1	10	15
Step 2:	3	7	4	1	4	8	10	15

Etc.

Sorting animation: http://www.sorting-algorithms.com/insertion-sort

Selection sort

```
public class SelectionSort {
   public static void main(String[] args) {
      int[] data = {3, 10, 4, 6, 15, 1, 7, 4};
      sort(data);
      for (int d : data)
         System.out.println(d);
  }

  // Returns a sorted array (in-place),
  // using selection sort.
  private static void sort (int[] a) {
    for (int j = 0, n = a.length; j < n; j++) {
       // find index of maximal element in a[0]...[n-j]
       int k = 0;
       for (int i = 1; i < n-j; i++)
          if (a[i] > a[k])
             k = i;
       // switch
       {int temp=a[n-j-1]; a[n-j-1]=a[k]; a[k]=temp;}
    }
  }
}
```

```
D:\demo>java SelectionSort
1
3
4
4
6
7
10
15
```

In-place sorting:

The variable `data` holds a reference to the array's values.

It's an object variable; when we pass it to the sorting method, the passing mode is "by reference"

Therefore, the method has read/write access to the array values

Best practice advice:

Such methods are said to have "side effects", and must be used with great caution.

Selection sort

It would make sense to put the sort method in the "IntArrays" class.

This way, any class will be able to use its sorting services.

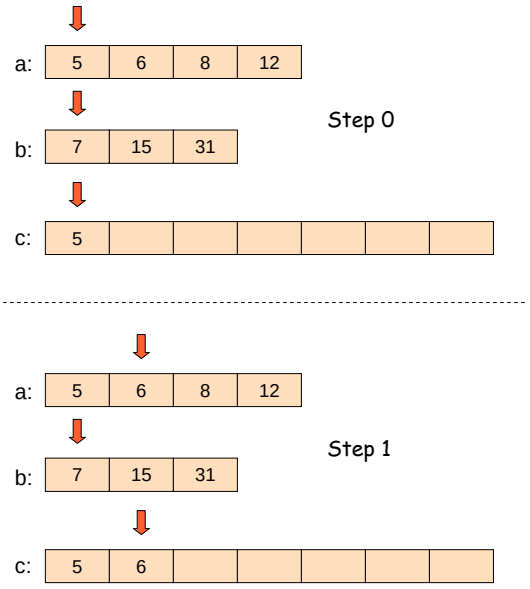
```
// Methods for processing arrays of integers.
public class IntArrays {

   // Returns true if a contains zero and false otherwise
   public static boolean containsZero (int[] a)

   // Returns the sum of the values in the array
   public static int sum (int[] a)

   // Returns the array, sorted.
   // SIDE EFFECT: this method changes the array!
   public static void sort (int[] a)

   // More array processing methods
}
```

```
class SomeClass {
    ...
    int[] data = {3, 10, 4, 6, 15, 1, 7, 4};
    IntArrays.sort(data);
    for (int d : data)
      System.out.println(d);
}
```

Outline

- How arrays are implemented in Java
- Arrays as parameters
- Arrays of objects
- Array processing examples
  - Polynomial
  - Sorting
  - Merging

Merging



Input: sorted arrays a and b

Output: merged and sorted array c

Algorithm (informal):

```
int[] c = new int[a.length + b.length]
int i = j = k = 0
```

In each step:

```
if (a[i] < b[j])
   c[k++] = a[i++]
else
   c[k++] = b[j++]
```