Arrays (Chap 5)

- Array – an ordered collection of similar items
  - Student test scores
  - Mail boxes at your college
  - A collection of books on a shelf
- An array can be thought of as a type of container.

Array operations

- Create an array
  `new int[50]`
- Use a variable to refer to the newly created array
  `int[] ourFirstArray = new int[50];`
- Place items into certain positions in the array
  `ourFirstArray[i] = Console.in.readInt();`
- Get the value of an item stored at a certain position in the array
  `System.out.println(ourFirstArray[i]);`

Array operations

- Create an array (build a book case to hold 20 books).
- Give a name to the array (the white book case in the hallway).
- Place items into certain positions in the array (put the books on the shelf in some particular order).
- Get the value of an item stored at a certain position in the array (get me the 5th book).

```
// ReverseArray.java: One-dimensional array
import tio.*;
class ReverseArray {
  public static void main(String[] args) {
    int[] simpleArray = new int[5];
    System.out.println("Enter 5 integers.");
    for(int i = 0; i < 5; i++)
      simpleArray[i] = Console.in.readInt();
    System.out.println("In reverse they are:");
    for(int i = 0; i < 5; i++)
      System.out.println(simpleArray[4-i]);
  }
}
```

```
// ArraySum.java - sum the elements in an array and
//    compute their average
class ArraySum {
  public static void main(String[] args) {
    int[] data = {11, 12, 13, 14, 15, 16, 17};
    int sum = 0;
    double average;
    for (int i = 0; i < 7; i++) {
      sum = sum + data[i];
      System.out.print(data[i] + ", ");
    }
    average = sum / 7.0;
    System.out.println("\n\n sum = " + sum
        + " average = " + average);
  }
}
```

Common Error

IndexOutOfBoundsException

```
for(int i = 1; i <= data.length; i++)
  System.out.println(data[i]);
```

Another Common Error

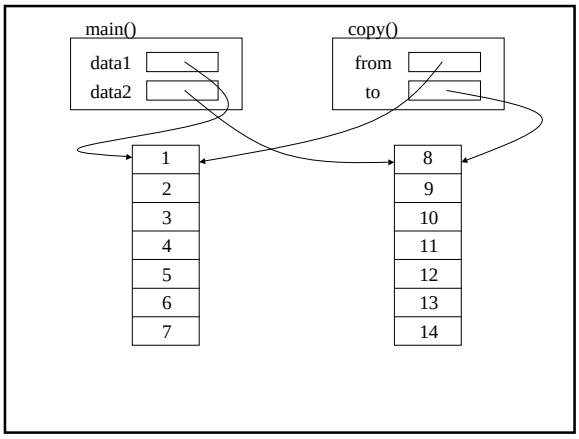
- Forgot to create the array:

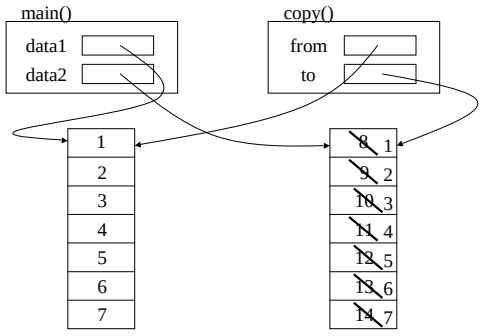
```
int[] x;
...
for(int i = 0; i < x.length; i++)
  x[i] = ...;
```

- For local variables, the Java compiler will catch this (x is used before being assigned a value).
- If x is an instance variable, then it defaults to null and you get a `NullPointerException`.

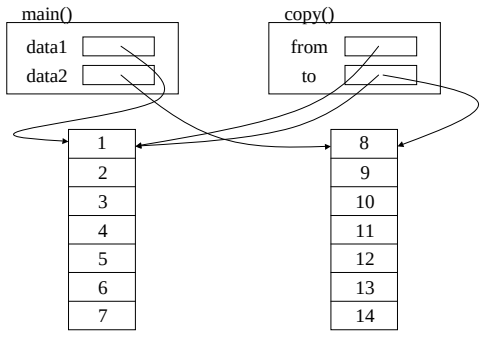
```
// sum the elements in an array using a method
class ArraySum2 {
  public static void main(String[] args) {
    int[] data1 = {1, 2, 3, 4, 5, 6, 7};
    int[] data2 = {16, 18, 77};

    System.out.println("data1:" + sum(data1));
    System.out.println("data2:" + sum(data2));
  }
  // sum the elements in an array
  static int sum (int[] a) {
    int sum = 0;

    for (int i = 0; i < a.length; i++)
      sum = sum + a[i];
    return sum;
  }
}
```

```
// demonstrate that array parameters can be modified
class TestCopy {
  public static void main(String[] args) {
    int[] data1 = {1, 2, 3, 4, 5, 6, 7};
    int[] data2 = {8, 9, 10, 11, 12, 13, 14};
    copy(data1, data2);
    System.out.println("data1:");
    for (int i = 0; i < data1.length; i++)
      System.out.println(data1[i]);
    System.out.println("data2:");
    for (int i = 0; i < data2.length; i++)
      System.out.println(data2[i]);
  }
  static void copy(int[] from, int[] to) {
    for (int i = 0; i < from.length; i++)
      to[i] = from[i];
  }
}
```

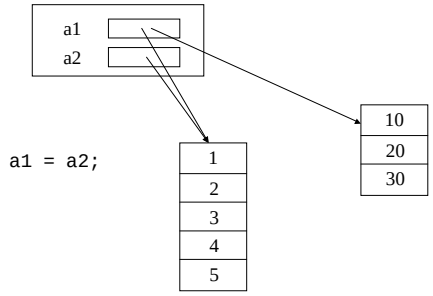





```
static void copy(int[] from, int[] to){
  to = from;
}
```



```
int[] a1 = {10, 20, 30};
int[] a2 = {1, 2, 3, 4, 5};
```



Duplicating/Coping an array

```
static int[] duplicate(int[] a) {
  int[] theCopy = new int[a.length];
  for (int i = 0; i < a.length; i++)
    theCopy[i] = a[i];
  return theCopy;
}

a1 = (int[])a2.clone(); // built in copy
```

```
// ArrayTest.java - minimum and maximum of an array
import tio.*;

class ArrayTest {
  public static void main(String[] args) {
    int[] data;
    int n;
    System.out.println("Enter size of data[]:");
    n = Console.in.readInt();
    data = new int[n];
    System.out.println("Enter " + n + " integers:");
    readArray(data);
    printArray(data, "My Data");
    System.out.println("minimum is " + minimum(data)
        + "   maximum is " + maximum(data));
  }
```

```
// fill an array by reading values from the console
static void readArray(int[] a) {
  for ( int i = 0; i < a.length; i++) {
    a[i] = Console.in.readInt();
  }
}
// find the maximum value in an array
static int maximum(int[] a) {
  int  max = a[0]; //initial max value
  for (int i = 1; i < a.length; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
```

```
  // find the minimum value in an array
  static int minimum(int[] a) {
    int  min = a[0]; //initial max value

    for (int i = 1; i < a.length; i++)
      if (a[i] < min)
        min = a[i];
    return min;
  }
  // print the elements of an array to the console
  static void printArray(int[] a, String arrayName) {
    System.out.println(arrayName );
    for (int i = 0; i < a.length; i++)
      System.out.print(a[i] + "  ");
    System.out.println();
  }
}
```

Sorting

- Sorting is a common task for computers.
- There are many different algorithms for sorting. Just a few are:
  - insertion sort - add elements one at a time
  - bubble sort - move item up until in right place
  - selection sort - pick 1st, then 2nd, etc.
  - quick sort - divide and conquer
- They differ in the number of comparisons required to sort N items.

```
// SelectionSort.java - sort an array of integers
import tio.*;

class SelectionSort {
  public static void main(String[] args) {
    int[] a = {7, 3, 66, 3, -5, 22, -77, 2};

    sort(a);
    for (int i = 0; i < a.length; i++){
      System.out.println(a[i]);
    }
  }
```

```
  // sort using the selection sort algorithm
  static void sort(int[] data) {
    int   next, indexOfNext;
    for (next = 0; next < data.length - 1; next++) {
      indexOfNext = min(data,next,data.length - 1);
      swap(data, indexOfNext, next);
    }
  }
```

```
  // find the index of the smallest element in
  // a specified range of indicies in an array
  static int min(int[] data, int start, int end) {
    int indexOfMin = start; // initial guess

    for (int i = start+1; i <= end; i++)
      if (data[i] < data[indexOfMin])
        indexOfMin = i; // found a smaller value
    return indexOfMin;
  }
  // swap to entries in an array
  static void swap(int[] data, int first, int second){
    int temp;

    temp = data[first];
    data[first] = data[second];
    data[second] = temp;
  }
```

Pass	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
original	7	3	66	3	-5	22	-77	2
First	-77	3	66	3	-5	22	7	2
Second	-77	-5	66	3	3	22	7	2
Third	-77	-5	2	3	3	22	7	66
Fourth	-77	-5	2	3	3	22	7	66
Fifth	-77	-5	2	3	3	22	7	66
Sixth	-77	-5	2	3	3	7	22	66
Seventh	-77	-5	2	3	3	7	22	66

Searching an ordered array

- Like sorting, searching for an item in an ordered list is another common operation for computers.
- We can just search the list from the start.
- Starting in the middle is better - binary search.

```
// BinarySearch.java - use bisection search to find
//   a selected value in an ordered array
class BinarySearch {
  public static void main(String[] args) {
    int[] data = {100, 110, 120, 130, 140, 150};
    int index = binarySearch(data, 120);

    System.out.println(index);
  }
```

```
  // find the index of element v in array keys
  // return -1 if it is not found
  static int binarySearch(int[] keys, int v) {
    int position;
    int begin = 0, end = keys.length - 1;
    while (begin <= end) {
      position = (begin + end)/2;
      if (keys[position] == v)
        return position;            // just right
      else if (keys[position] < v)
        begin = position + 1;       // too small
      else
        end = position - 1;         // too big
    }
    return -1;
  }
}
```

An array of something other than integers

```
//Sieve of Eratosthenes for Primes up to 100.
class Primes {
  public static void main(String[] args) {
    boolean[] sieve = new boolean[100];
    int i;
    System.out.println(" Table of primes to 100.");
    for (i = 0; i < 100; i++)
      sieve[i] = true;
    for (int j = 2; j < Math.sqrt(100); j++)
      if (sieve[j])
         crossOut(sieve, j, j + j);
    for (i = 0; i < 100; i++) //print primes
      if (sieve[i])
         System.out.print(" " + i);
  }
```

```
  public static void crossOut(boolean[] s,
                int interval, int start)
  {
   for (int i = start; i < s.length; i += interval)
      s[i] = false;
  }
}
```

Working with arrays of characters

```
//CountWord.java
import tio.*;
public class CountWord {
  public static void main(String[] args) {
    String input;
    char[] buffer;
    System.out.println("type in line" );
    input = Console.in.readLine();
    System.out.println(input);
    buffer = input.toCharArray();
    System.out.println("word count is " +
                        wordCount(buffer));
  }
```

```
  //word are separated by nonalphabetic characters
  public static int wordCount(char[]buf) {
    int position = 0, wc = 0;
    while (position < buf.length) {
      while (position < buf.length &&
             !isAlpha(buf[position]))
        position++;
      if (position < buf.length)
        wc++;
      while (position<buf.length &&
             isAlpha(buf[position]))
        position++;
    }
    return wc;
  }
}
```

```
  public static boolean isAlpha(char c) {
     return (c >= 'a' && c <= 'z') ||
            (c >= 'A' && c <= 'Z');
  }
```

There is actually a predefined function to do this:

```
public static boolean Character.isLetter(char c)
```

Arrays of non-primitive types

- We can have arrays of any type.

Arrays of Strings

```
// StringArray.java - uses a string array initializer
class StringArray {
  public static void main(String[] args) {
    String[] myStringArray = { "zero", "one", "two",
        "three", "four", "five", "six", "seven",
        "eight", "nine"};
    for (int i = 0; i < myStringArray.length; i++)
      System.out.println(myStringArray[i]);
  }
}
```

Command Line Arguments

```
// CommandLine.java - print command line arguments
class CommandLine {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

```
os-prompt>java CommandLine this "is another" test
this
is another
test
```

```
// PointArray.java-example array nonprimitive values
import java.awt.Point; // Point is a standard Java class

class PointArray {
  public static void main(String[] args) {
    Point[] triangle;

    triangle = new Point[3];
    triangle[0] = new Point(10,20);
    triangle[1] = new Point(35,90);
    triangle[2] = new Point(20, 85);
    for (int i = 0; i < triangle.length; i++)
      System.out.println(triangle[i]);
    translate(triangle,100,200);
    for (int i = 0; i < triangle.length; i++)
      System.out.println(triangle[i]);
  }
```

```
  public static void translate(Point[] points,
                       int deltaX, int deltaY)
  {
    for (int i = 0; i < points.length; i++)
      points[i].translate(deltaX, deltaY);
  }
}
```

```
java.awt.Point[x=10,y=20]
java.awt.Point[x=35,y=90]
java.awt.Point[x=20,y=85]
java.awt.Point[x=110,y=220]
java.awt.Point[x=135,y=290]
java.awt.Point[x=120,y=285]
```

Multi-Dimensional Arrays

- 2-D arrays can be used to represent tables of data with columns and rows.
- Three dimensional arrays can represent volume data with three coordinates, x-y-z.

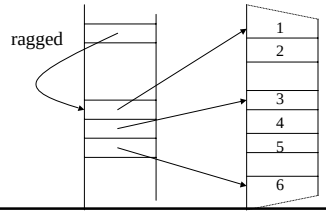
```
String[] one;
double[][] two;
char[][][] three;
```

data[0][0]	data[0][1]	data[0][2]	data[0][3]	data[0][4]
data[1][0]	data[1][1]	data[1][2]	data[1][3]	data[1][4]
data[2][0]	data[2][1]	data[2][2]	data[2][3]	data[2][4]

```
  // TwoD.java - simple example of two-dimensional array
class TwoD {
  public static void main(String[] args) {
    int[][] data = new int[3][5];
    for (int i = 0; i < data.length; i++) {
      System.out.print("Row " + i + ":   ");
      for (int j = 0; j < data[i].length; j++) {
        data[i][j] = i * j;
        System.out.print(data[i][j] + ", ");
      }
      System.out.println();
    }
  }
}
```

2D Initializers

```
int[][] a = {{1, 2}, {3, 4}, {5, 6}};
int[][] b = {{1, 2, 3}, {4, 5, 6}};
int[][] c = {{1, 2, 3, 4, 5, 6}}; //degenerate
int[][] ragged = {{1, 2}, {3, 4, 5}, {6}};
```



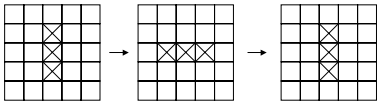
Conway's Game of Life

- A cell is either empty (blank), or alive (X).
- Each cell is the center of a 3 x 3 square grid of cells, which contain the cell's 8 neighbors.
- Simple rules control which cells become alive or die in each step of the simulation.
- The simulation is conducted in principle on an infinite two-dimensional grid.

The "Rules" of Life

- An empty cell at time t becomes alive at time t+1 if and only if exactly 3 neighboring cells are alive at time t.
- An alive cell at time t remains alive at time t+1 if and only if either 2 or 3 neighboring cells are alive at time t. Otherwise it dies for lack of company (< 2) or overcrowding (> 3).

The "Blinker"



Game of Life: Requirements

- Used finite grid instead of infinite grid.
- Treat borders as lifeless zones.
- Set initial state by reading X's and dots from the console.
- The user will specify the size of the grid, which will always be square.
- The user specifies the number of steps to simulate.
- The program should echo the initial configuration and then print each new generation.

Top-Level Pseudocode

```
read in the size of the grid
read in the initial generation
read in the number of generations to simulate
print the current generation
for the specified number of generations
    advance one generation
    print the generation
end for loop
```

Advance one generation

```
for each cell
    compute the number of neighbors
    apply the game rules to decide if the cell
        should be alive or dead in the next gen
end for loop
```

Compute the number of neighbors

```
visit each cell in the 3x3 subarray around the cell
keep a count of how many of those are alive
if the center cell is alive
    then return one less than the count found
    else return the count found
```

Top-Level Refined

```
read in the size of the grid
create two arrays, currentGen and nextGen
read in the initial generation into currentGen
read in the number of generations to simulate
print the current generation
for the specified number of generations
    update nextGen using currentGen
    swap currentGen and nextGen
    print currentGen
end for loop
```

Advance one generation: Refined

```
input: currentGen and nextGen
output: updated nextGen

for each cell in currentGen
    compute the number of neighbors
    apply the game rules to decide if the cell
        should be alive or dead in the next gen
    store the result, alive or dead, in the
        corresponding cell in nextGen
end for loop
```

```
//GameOfLife.java - Conway's Game of Life
import tio.*;
class GameOfLife {
  public static void main(String[] args) {
    // read in the size of the grid and create arrays
    int size = 10; // fix at 10 for testing
    boolean[][]  currentGeneration, nextGeneration;
    currentGeneration = new boolean[size][size];
    nextGeneration = new boolean[size][size];
    readInitialGeneration(currentGeneration);
```

```
    // read in the number of generations to simulate
    int cycles = 4; // fix at 4 for testing
    printState(currentGeneration);
    for (int i = 0; i < cycles; i++) {
      System.out.println("Cycle = " + i + "\n\n");
      advanceOneGen(currentGeneration,
                    nextGeneration);
      printState(nextGeneration);
      // swap current and next generations
      boolean[][] temp = nextGeneration;
      nextGeneration = currentGeneration;
      currentGeneration = temp;
    }
  }
```

```
  // read the initial generation from the input
  // a dot means empty and a * means alive
  // any other characters are ignored
  // the border cells are all set to empty
  // the method assumes the array is square
  static void readInitialGeneration(boolean[][] w) {
    for (int i = 0; i < w.length; i++)
      for (int j = 0; j < w[i].length; j++) {
        char c = (char)Console.in.readChar();
        //skip illegal characters
        while (c != '.' && c != '*')
          c = (char)Console.in.readChar();
        if (c == '.')
          w[i][j] = EMPTY;
        else
          w[i][j] = ALIVE;
      }
```

```
//set border cells to be empty
  int border = w.length - 1;

  for (int i = 0; i < w.length; i++){
    w[i][0] = w[0][i] = EMPTY;
    w[i][border] = w[border][i] = EMPTY;
  }
}
```

```
  // print a generation to the console
  static void printState(boolean[][] w) {
    for (int i = 0; i < w.length; i++) {
      System.out.println();
      for (int j = 0; j < w[i].length; j++)
        if (w[i][j] == ALIVE)
          System.out.print('X');
        else
          System.out.print('.');
    }
    System.out.println();
  }
```

```
// compute the number of alive neighbors of a cell
static int neighbors(int row, int column,
                     boolean[][] w)
{
  int neighborCount = 0;
  for (int i = -1; i <= 1; i++)
    for (int j = -1; j <= 1; j++)
      if (w[row + i][column + j] == ALIVE)
        neighborCount = neighborCount + 1;
  if (w[row][column] == ALIVE)
    neighborCount--;
  return neighborCount;
}
```

```
  static void advanceOneGen(boolean[][] wOld,
                            boolean[][] wNew)
  {
    int neighborCount;
    for (int i = 1; i < wOld.length - 1; i++)
      for (int j = 1; j < wOld[i].length - 1; j++) {
        neighborCount = neighbors(i, j, wOld);
        if (neighborCount == 3)
          wNew[i][j] = ALIVE;
        else if (wOld[i][j] == ALIVE &&
                 neighborCount == 2)
          wNew[i][j] = ALIVE;
        else
          wNew[i][j] = EMPTY;
      }
  }
  static final boolean ALIVE = true;
  static final boolean EMPTY = false;
}
```

A sample input file

```
..........
....*.....
.....*....
.*...*....
..****....
..........
..........
..........
..........
..........
```

Sample Output

```
..........
....X.....
.....X....
.X...X....
..XXXX....
..........
..........
..........
..........
..........
Cycle = 0
```

Skipping some intermediate output

```
Cycle = 3
..........
...X..X...
.......X..
...X...X..
....XXXX..
..........
..........
..........
..........
..........
```