Announcements

• PS 4 is ready, due next Thursday, 9:00pm

• Midterm Exam 1: 10/14 (Fri), 9:00am-10:53am
  • Room TBD
  • Scope: Lecture 1 to Lecture 9 (Chapters 1 to 6 of text)
  • You may bring a sheet of paper (A4, both sides)

• Classroom change to A115 (both lecture and lab)

• Reading assignment for this lecture: Chapter 7
Arrays
Problem

• Read/Generate one hundred numbers,
• Compute their average, and
• Find out how many numbers are above the average.
With any new data structure, like an array

Learn four things

1. How to **create** a new instance
2. How to **write** a value into it
3. How to **read** a value out of it
4. How to destroy it when done using it (with Java this last one is not necessary)
Introducing arrays

Array is a data structure that represents a collection of the same types of data.

```java
double[] myList = new double[10];
```
Declaring array variables

- datatype[] arrayRefVar;

Examples:

double[] myList;

int[] scores;

String[] names;
Creating arrays

arrayRefVar = new datatype[arraySize];

Example:
myList = new double[10];
scores = new int[100];
names = new String[32];

myList[0] references the first element in the array.
myList[9] references the last element in the array.
Declaring and creating in one Step

- datatype[] arrayRefVar = new datatype[arraySize];

Examples

double[] myList = new double[10];
int[] scores = new int[100];
String[] names = new String[32];
The **length** of an array

- Once an array is created, its size is fixed
- It **cannot** be changed

- You can find its size using
  `arrayRefVar.length` *(note: not `length()`)*

- For example,
  `myList.length` returns 10
Default values

When an array is created, its elements are assigned the default value of

0 for the numeric primitive data types,
'\u0000' for char types, and
false for boolean types.
Indexed variables

• The array elements are accessed through the index
• Indices are 0-based, from 0 to arrayRefVar.length-1
• Each element in the array is represented using the following syntax, known as an indexed variable:
  – arrayRefVar[index]

• Example:
  – scores[3]
  – names[0]
Using indexed variables

• After an array is created, an indexed variable can be used in the same way as a regular variable.

• For example,
  • Write access: writing a value into it
    myList[0] = 3.4;
  • Read access: reading a value our of it
    System.out.println(myList[0]);
  • Read and write in one: add the value in myList[0] and myList[1] and put it into myList[2]
    myList[2] = myList[0] + myList[1];
Array initializers

• Declaring, creating, initializing in one step:

```java
int[] scores = {9, 9, 4, 5, 2};
```

• This shorthand syntax must be in one statement.
Declaring, creating, initializing using the shorthand notation

```java
int[] scores = {9, 9, 4, 5, 2};
```

This shorthand notation is equivalent to the following statements:

```java
int[] scores = new double[5];
scores[0] = 9;
scores[1] = 9;
scores[2] = 4;
scores[3] = 5;
Scores[4] = 2;
```
Caution

• Using the shorthand notation, you have to declare, create, and initialize the array all in one statement.

• Splitting it would cause a syntax error.

• For example, the following is wrong:

```java
int[] scores;
scores = {9, 9, 4, 5, 2};
```
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
Trace program with arrays

```java
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
```

After the array is created:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}

After i++, i becomes 2

After the first iteration

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
Trace program with arrays

```java
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
```

i ( =5) < 5 is false. Exit the loop

After the fourth iteration:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}
Processing arrays

See these examples in the text.

1. (Initializing arrays with input values)
2. (Initializing arrays with random values)
3. (Printing arrays)
4. (Summing all elements)
5. (Finding the largest element)
6. (Finding the smallest index of the largest element)
7. (Random shuffling)
8. (Shifting elements)

See Average.java
See ArrayExamples.java
Initializing arrays with input values

```java
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++) {
    myList[i] = input.nextDouble();
}
```
Initializing arrays with random values

```java
for (int i = 0; i < myList.length; i++) {
    myList[i] = Math.random() * 100;
}
```
Printing arrays

```java
for (int i = 0; i < myList.length; i++) {
    System.out.print(myList[i] + " ");
}
```
Summing all elements

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

Finding the largest element

double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
    }
}
}
Random shuffling

```java
for (int i = myList.length - 1; i > 0; i--) {
    // Generate an index j randomly with 0 <= j <= i
    int j = (int)(Math.random() * (i + 1));

    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}
```
**Shifting elements**

```java
double temp = myList[0]; // Retain the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}

// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```
Enhanced **for** loop ("**for-each**" loop)

A new **for** loop that enables you to traverse the array sequentially without using an index variable.

For example, the following code displays all elements in the array `myList`:

```java
for (double value: myList)
    System.out.println(value);
```

In general, the syntax is

```java
for (elementType value: arrayRefVar) {
    // Process the value
}
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.
Opening problem

Read one hundred numbers, compute their average, and find out how many numbers are above the average.

Write it!
Deck of cards

The problem is to write a program that picks four cards randomly from a deck of 52 cards. All the cards can be represented using an array named deck, filled with initial values 0 to 51, as follows:

```java
int[] deck = new int[52];
// Initialize cards
for (int i = 0; i < deck.length; i++) {
    deck[i] = i;
}
```
Deck of cards (cont.)

<table>
<thead>
<tr>
<th>0</th>
<th>13 Spades (♠)</th>
<th>[0] 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>13 Hearts (♥)</td>
<td>[12] 12</td>
</tr>
<tr>
<td>13</td>
<td>13 Diamonds (♦)</td>
<td>[13] 13</td>
</tr>
<tr>
<td>25</td>
<td>13 Clubs (♣)</td>
<td>[25] 25</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>[26] 26</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>[38] 38</td>
</tr>
<tr>
<td>39</td>
<td></td>
<td>[39] 39</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td>[51] 51</td>
</tr>
</tbody>
</table>

Card number 6 is the 7 (6 % 13 = 6) of Spades (7 / 13 is 0)
Card number 48 is the 10 (48 % 13 = 9) of Clubs (48 / 13 is 3)
Card number 11 is the Queen (11 % 13 = 11) of Spades (11 / 13 is 0)
Card number 24 is the Queen (24 % 13 = 11) of Hearts (24 / 13 is 1)
Deck of cards (cont.)

\[
\begin{align*}
cardNumber / 13 &= \begin{cases} 
0 & \rightarrow \text{Spades} \\
1 & \rightarrow \text{Hearts} \\
2 & \rightarrow \text{Diamonds} \\
3 & \rightarrow \text{Clubs}
\end{cases} \\
cardNumber \% 13 &= \begin{cases} 
0 & \rightarrow \text{Ace} \\
1 & \rightarrow \text{2} \\
\cdot & \rightarrow \cdot \\
10 & \rightarrow \text{Jack} \\
11 & \rightarrow \text{Queen} \\
12 & \rightarrow \text{King}
\end{cases}
\end{align*}
\]
Deck of cards

This problem builds a foundation for future more interesting and realistic applications:

See Exercise 22.15.
Lotto numbers (optional)

Suppose you play the Pick-10 lotto. Each ticket has 10 unique numbers ranging from 1 to 99. You buy a lot of tickets. You like to have your tickets to cover all numbers from 1 to 99. Write a program that reads the ticket numbers from a file and checks whether all numbers are covered. Assume the last number in the file is 0. (We don’t know how to read things from a file yet. So, this is a future exercise.)
Lotto numbers (optional) (cont.)

(a)  

(b)  

(c)  

(d)  

(e)  

isCovered

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>97</th>
<th>98</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>3</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>97</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>98</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Copying arrays

Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;

Before the assignment

\[
\text{list2} = \text{list1};
\]

list1 \rightarrow \text{Contents of list1}

list2 \rightarrow \text{Contents of list2}

After the assignment

\[
\text{list2} = \text{list1};
\]

list1 \rightarrow \text{Contents of list1}

list2 \rightarrow \text{Contents of list2}
Copying arrays

Using a loop:

```java
int[] sourceArr = {2, 3, 1, 5, 10};
int[] targetArr = new int[sourceArr.length];

for (int i = 0; i < sourceArr.length; i++){
    targetArr[i] = sourceArr[i];
}
```
The `arraycopy` utility

```java
arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
```

Example:

```java
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```
Passing arrays to methods

```java
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}

// Invoke the method
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);

// Invoke the method
printArray(new int[]{3, 1, 2, 6, 4, 2});

Anonymous array
```
Anonymous array  (optional)

The statement

```java
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the following syntax:

```java
new dataType[]{literal0, literal1, ..., literalk};
```

There is no explicit reference variable for the array. Such array is called an **anonymous array**.
Pass by value

Java uses *pass by value* to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

- For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the called method does not affect the value of the variable outside the method.

- For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the called method body will affect the original array that was passed as the argument.
Simple example

```java
public class Pass1 {
    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]
    }
}

See Pass1.java
```
Call stack

When invoking \( m(x, y) \), the values of \( x \) and \( y \) are passed to \texttt{number} and \texttt{numbers}. Since \( y \) contains the reference value to the array, \texttt{numbers} now contains the same reference value to the same array.
When invoking $m(x, y)$, the values of $x$ and $y$ are passed to number and numbers. Since $y$ contains the reference value to the array, numbers now contains the same reference value to the same array.

See Pass2.java
Heap

The JVM stores the array in an area of memory, called *heap*, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.
Passing arrays as arguments

- Objective: Demonstrate differences of passing primitive data type variables and array variables.
Example (cont.)

Invoke swap(int n1, int n2). The primitive type values in a[0] and a[1] are passed to the swap method.

The arrays are stored in a heap.

Invoke swapFirstTwoInArray(int[] array). The reference value in a is passed to the swapFirstTwoInArray method.

See Pass2.java
Returning an array from a method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

See `ReverseTest.java`
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

list  
1 2 3 4 5 6

result  
0 0 0 0 0 0 0
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

**list**

| 1 | 2 | 3 | 4 | 5 | 6 |

**result**

| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

i (= 0) is less than 6
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

---

**Trace:**

- **Initial state:**
  - List: `1, 2, 3, 4, 5, 6`
  - Result: `0, 0, 0, 0, 0, 0`

- **Iteration 1:**
  - `i = 0, j = 5`
  - Assign `list[0]` to `result[5]`
  - Result: `0, 0, 0, 0, 0, 1`

- **Iteration 2:**
  - `i = 1, j = 4`
  - Assign `list[1]` to `result[4]`
  - Result: `0, 0, 0, 0, 1, 2`

- **Iteration 3:**
  - `i = 2, j = 3`
  - Assign `list[2]` to `result[3]`
  - Result: `0, 0, 0, 1, 2, 3`

- **Iteration 4:**
  - `i = 3, j = 2`
  - Assign `list[3]` to `result[2]`
  - Result: `0, 0, 1, 2, 3, 4`

- **Iteration 5:**
  - `i = 4, j = 1`
  - Assign `list[4]` to `result[1]`
  - Result: `0, 1, 2, 3, 4, 5`

- **Iteration 6:**
  - `i = 5, j = 0`
  - Assign `list[5]` to `result[0]`
  - Result: `1, 2, 3, 4, 5, 6`

**Final state:**

- List: `1, 2, 3, 4, 5, 6`
- Result: `1, 2, 3, 4, 5, 6`
Trace the reverse method

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

list     1 2 3 4 5 6
result    0 0 0 0 0 1

After this, i becomes 1 and j becomes 4
Trace the reverse method

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}

list 1 2 3 4 5 6
result 0 0 0 0 0 0 1
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
```

After this, i becomes 2 and j becomes 3.
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>list</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1 2 3 4 5 6</td>
<td>0 0 0 3 2 1</td>
</tr>
</tbody>
</table>
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

After this, i becomes 3 and j becomes 2
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

```
<table>
<thead>
<tr>
<th>list</th>
<th>1 2 3 4 5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0 0 0 3 2 1</td>
</tr>
</tbody>
</table>
```
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i = 3 and j = 2
Assign list[i] to result[j]
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

After this, i becomes 4 and j becomes 1

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 4 and j becomes 1

```
<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```
<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
```

i (=4) is still less than 6
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

```
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i = 4 and j = 1
Assign list[i] to result[j]
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

After this, i becomes 5 and j becomes 0

<table>
<thead>
<tr>
<th>list</th>
<th>1 2 3 4 5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0 5 4 3 2 1</td>
</tr>
</tbody>
</table>
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```plaintext
<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
```

i (=5) is still less than 6
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {  
        result[j] = list[i];
    }

    return result;
}
```

text

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

**Animation:**
- `i = 5` and `j = 0`
- Assign `list[i]` to `result[j]`

<table>
<thead>
<tr>
<th>list</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 6 and j becomes -1.
Trace the reverse method

```java
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```

```
list
  1 2 3 4 5 6
result
  6 5 4 3 2 1
```

i (=6) < 6 is false. So exit the loop.
Trace the reverse method

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Return result
Counting occurrence of each letter

- Generate 100 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.
Command line arguments:
What was that (String[] args) thing?

• Who calls the main method in a Java program, if ever?
  – The operating system (e.g., Unix, DOS, etc.) does

• See ArgsTest.java

• See ArgsTest2.java

• See SimulateOS.java
Measuring execution time

• Use `System.currentTimeMillis()` to get the current time
  – Call it twice and compute the difference to get the elapsed time between two points in time

• See `Benchmark.java`
Sorting arrays

Sorting is a common task in computer programming.

Many different algorithms have been developed for sorting.

This section introduces a simple, intuitive sorting algorithms: *selection sort*.
Selection sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.

- Select 1 (the smallest) and swap it with 2 (the first) in the list.
- The number 1 is now in the correct position and thus no longer needs to be considered.
- Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.
- The number 2 is now in the correct position and thus no longer needs to be considered.
- Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.
- The number 4 is now in the correct position and thus no longer needs to be considered.
- Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.
- The number 5 is now in the correct position and thus no longer needs to be considered.
- Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.
- The number 6 is now in the correct position and thus no longer needs to be considered.
- Since there is only one element remaining in the list, the sort is completed.

The number 8 is now in the correct position and thus no longer needs to be considered.
From idea to solution

```java
for (int i = 0; i < a.length; i++) {
    select the smallest element in a[i...aSize-1];
    swap the smallest with a[i], if necessary;
    // a[i] is in its correct position.
    // The next iteration apply on a[i...aSize-1]
}
```

See `Sort.java`
The Arrays.sort method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code sorts an array of numbers and an array of characters.

```java
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

Java 8 now provides Arrays.parallelSort(list) that utilizes the multicore for fast sorting.
Searching arrays

Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, \textit{linear search} and \textit{binary search}.

```java
public class LinearSearch {
    // The method for finding item in the list a
    public static int linearSearch(int item, int[] a) {
        for (int i = 0; i < a.length; i++)
            if (item == a[i])
                return i;
        return -1;
    }
}
```
Linear search

The linear search approach compares the key element, $key$, \textit{sequentially} with each element in the array list. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns $-1$. 
### Linear search animation

<table>
<thead>
<tr>
<th>Key</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
</tbody>
</table>

**Key**

- 3

**List**

- 6 4 1 9 7 3 2 8
From idea to solution

// The method for finding item in the list a
public static int linearSearch(int item int[] list) {
    for (int i = 0; i < a.length; i++)
        if (key == a[i])
            return i;
    return -1;
}

int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5

See Search.java
Binary search

For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order.

   e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79

The binary search first compares the key with the element in the middle of the array.
Binary search (cont.)

Consider the following three cases:

• If the key is less than the middle element, you only need to search the key in the first half of the array.
• If the key is equal to the middle element, the search ends with a match.
• If the key is greater than the middle element, you only need to search the key in the second half of the array.
## Binary search

<table>
<thead>
<tr>
<th>Key</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1 2 3 4 6 7 8 9</td>
</tr>
<tr>
<td>8</td>
<td>1 2 3 4 6 7 8 9</td>
</tr>
<tr>
<td>8</td>
<td>1 2 3 4 6 7 8 9</td>
</tr>
</tbody>
</table>

*animation*
Binary search (cont.)

Key is 11

Key < 50

Key > 7

Key == 11
Binary search (cont.)

key is 54

key > 50

key < 66

key < 59
Binary search (cont.)

The `binarySearch` method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns -1.
From idea to solution

// Use binary search to find x in array a
public static int binarySearch(int[] a, int x) {
    int lo = 0;
    int hi = a.length - 1;

    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (a[mid] == x)
            return mid;
        else if (a[mid] < x)
            lo = mid + 1;
        else
            hi = mid - 1;
    }
    return -1;
}

See Search.java
The **Arrays.binarySearch** method

Since binary search is frequently used in programming, Java provides several overloaded binarySearch methods for searching a key in an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code searches the keys in an array of numbers and an array of characters.

```java
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is "+
    java.util.Arrays.binarySearch(list, 11));  // Return is 4

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is "+
    java.util.Arrays.binarySearch(chars, 't'));  // Return is -4 (insertion point is 3, so return is -3-1)
```

For the binarySearch method to work, the array must be pre-sorted in increasing order.
The `Arrays.toString(list)` method can be used to return a string representation for the list.
Calculator

• Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

  > java Calculator 2 + 3
  > java Calculator 2 - 3
  > java Calculator 2 / 3
  > java Calculator 2 . 3