

Outline



Method OverLoading

printf method

Arrays

Declaring and Using Arrays

Arrays of Objects

Array as Parameters

Variable Length Parameter Lists

split() Method from String Class

Integer & Double Wrapper Classes

Two-Dimensional Arrays

Method Overloading

- *Method overloading* is the process of giving a single method name multiple definitions in a class
- If a method is overloaded, the method name is not sufficient to determine which method is being called
- The *signature* of each overloaded method must be unique
- The signature includes the number, type, and order of the parameters

Method Overloading

- The `indexOf` method of `String` class is overloaded

```
int indexOf(char ch)
```

Returns the index of the first occurrence of `char ch` in this string.

Returns -1 if not matched.

```
int indexOf(char ch, int fromIndex)
```

Returns the index of the first occurrence of `char ch` in this string after `fromIndex`. Returns -1 if not matched.

```
int indexOf(String s)
```

Returns the index of the first occurrence of `String cs` in this string.

Returns -1 if not matched.

```
int indexOf(String s, int fromIndex)
```

Returns the index of the first occurrence of `String s` in this string after `fromIndex`. Returns -1 if not matched.

Method Overloading

- The `abs` method of `Math` class is overloaded

```
double abs(double d)
float  abs(float f)
int    abs(int i)
long   abs(long l)
```

Method Overloading

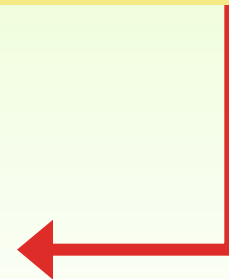
- The compiler determines which method is being invoked by analyzing the parameters

```
float tryMe(int x)
{
    return x + .375;
}
```

Invocation

```
result = tryMe(25, 4.32)
```

```
float tryMe(int x, float y)
{
    return x*y;
}
```



Overloading Methods

- The return type of the method is not part of the signature.
- That is, overloaded methods cannot differ only by their return type
- Constructors can be overloaded
- Overloaded constructors provide multiple ways to initialize a new object
 - Recall **PrintWriter** constructors

Overloading Example

- See [MyMaxOverloaded.java](#)

Outline

Method OverLoading



printf method

Arrays

Declaring and Using Arrays

Arrays of Objects

Array as Parameters

Variable Length Parameter Lists

split() Method from String Class

Integer & Double Wrapper Classes

Two-Dimensional Arrays

System.out.printf ("format string", data)

- The PrintWriter class has a printf method compatible with the fprintf method in C or Matlab.
- In the format string you can use format specifiers as well as leading text for the data you want to print.
- There should be exactly one data for each format specifier other than %n

Common Format Specifiers

- %d for integers
- %f for floating-point numbers
- %e for floating-point numbers (scientific notation)
- %s for string
- %c for char
- %b for boolean
- %n to advance to next line

Additional characters for format specifiers

- First of all, you can specify the length of data. If data doesn't fit, the space reserved is extended. Otherwise, numbers are aligned right, and strings are aligned left. To override this, you can additionally use a minus character.
- For numbers you can use zero character to fill the gap with 0's instead of blanks.
- For floating-point numbers you can use .digit to specify decimal places to digit.
- See [Example_printf.java](#)

Outline

Method OverLoading

printf method

Arrays



Declaring and Using Arrays

Arrays of Objects

Array as Parameters

Variable Length Parameter Lists

split() Method from String Class

Integer & Double Wrapper Classes

Two-Dimensional Arrays

Arrays

- Let's say we need to hold the temperature values of each day in Ankara in the past year
- Hard way: Create 365 different variables to hold each day's temperature

```
double tempDay1, tempDay2, ..., ..., tempDay365;
```

Ughhhhh!!!

Very difficult to even declare, use and manipulate!

- Easy way: use an array to hold all days temperatures

How to Declare an Array

- Create an array variable called **temperatures**
- Declared as follows:

```
double[] temperatures = new double[365];
```

- This sets up a location in memory for 365 double variables at once

Arrays Key Features

- Arrays provide an easy way to **hold related variables** at once (for example, temperature for the past year, gas prices for the last 30 days)
- It has some **ordering**, we can refer to array elements based on that ordering
- Homogenous, all data within a single array must share **the same data type** (for example, you can create an array of integer or boolean values, but not both)
- The **size of an array is fixed** once it is created.

Array Elements Type

- The element type can be a primitive type or an object reference
- Therefore, we can create an array of integers, an array of characters, an array of boolean, an **array of `String` objects**

Declaring Arrays

- First you **declare an array** reference variable

```
int [] myFirstArray;           //declares an array
                               //variable for ints
```

- Then you instantiate the array, that is allocate the necessary amount of memory for the array and let the variable hold the starting address of the array.

```
myFirstArray = new int [10]; //allocates memory
```

- We can combine the declaration and instantiation lines into one line as follows:

```
int [] myFirstArray = new int [10];
```

Declaring Arrays

- The `scores` array could be declared as follows:

```
int[] scores = new int[10];
```

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

- The type is `int` (an array of integers)
- The `name` of the array is `scores`
- The `size of the array` is `10`
- All positions of the new array will automatically be `initialized` to `the default value` for the array's type.

Declaring Arrays

- Some other examples of array declarations:

```
int[] weights = new int[2000];
```

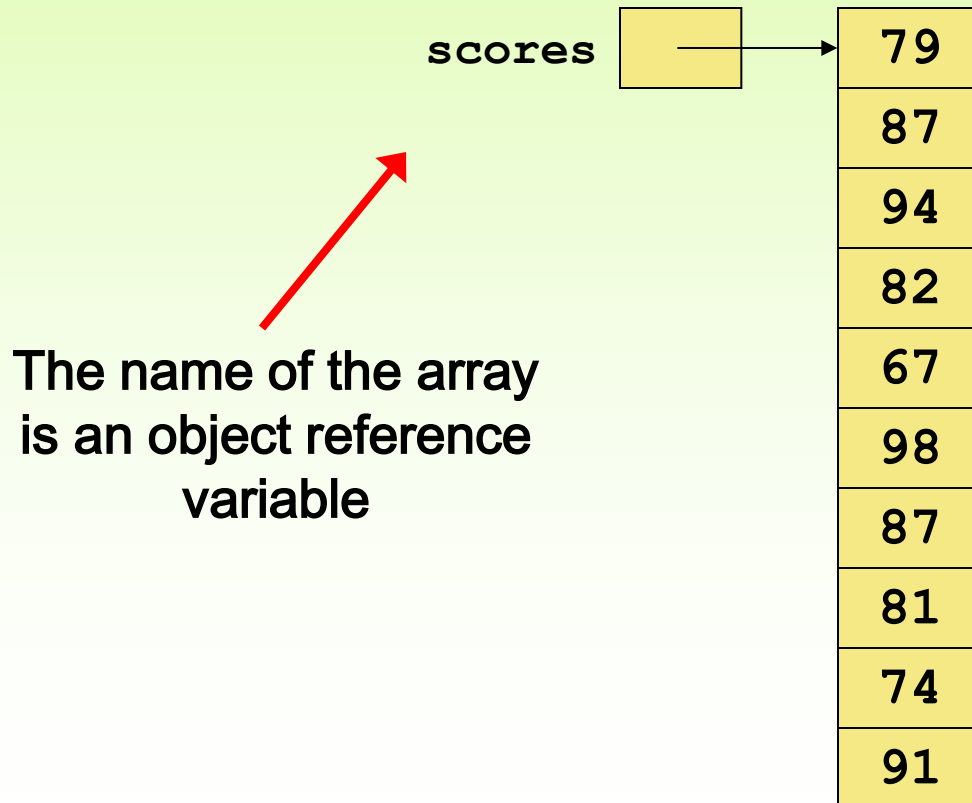
```
double[] prices = new double[500];
```

```
boolean[] flags;  
flags = new boolean[20];
```

```
char[] codes = new char[1750];
```

Arrays are Objects

- In Java, the array itself is an object that must be instantiated
- Another way to depict the `scores` array:



Array Basics

[Array1.java](#)

Array Elements

- Refers to the individual items represented by the array. For example,
 - an array of 10 integers is said to have 10 elements
 - an array of 5 characters has 5 elements
 - and so on...

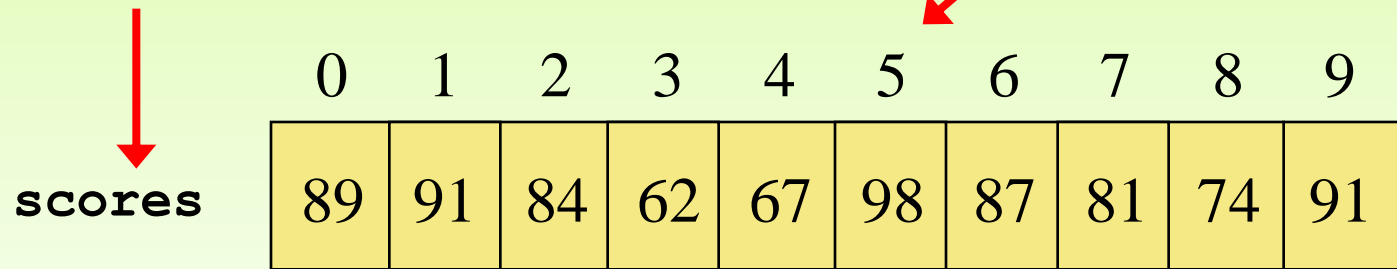
Array Index

- Refers to one particular element's position number in the array or more formally, as a *subscript*

```
int[] scores = new int[10];
```

The entire array
has a single name

Each value has a numeric *index*



An array of size N is indexed from 0 (zero) to N-1

This array holds 10 values that are indexed from 0 to 9

Array Element

0	1	2	3	4	5	6	7	8	9
79	87	94	82	67	98	87	81	74	91

- A particular value in an array is referenced using the array name followed by the index in brackets

scores [2]

refers to the value 94 (the 3rd value in the array)

BasicArray.java

The following example demonstrates the use of indices.

See [BasicArray.java](#)

Arrays

- An array element can be assigned a value, printed, or used in a calculation :

```
scores[2] = 89;
```

```
scores[first] = scores[first] + 2;
```

```
mean = (scores[0] + scores[1])/2;
```

```
System.out.println ("Top = " + scores[5]);
```

See [Array2.java](#)

Array Naming Considerations

- The rules for naming variables apply when selecting array variable names
 - Composed of letter, digit, \$ and underscore characters
 - Cannot start with a digit
 - Begins with a smallcase letter by convention

Array Length and Bounds

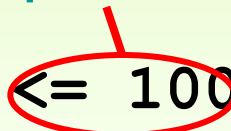
- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element.
- That is, if the array length is N , the index value must be in range 0 to $N-1$
- You will get the run-time error, **ArrayIndexOutOfBoundsException** if you use an invalid index

Bounds Checking

- For example, if the array `codes` can hold 100 values, it can be indexed from 0 to 99
- It's common to introduce *off-by-one errors* when using arrays:

```
for (int index=0; index <= 100; index++)  
    codes[index] = index*50 + epsilon;
```

problem



Array Length

- Each array object has a **public constant (not a method)** called `length` that stores the size of the array
- It is referenced using the array name:

`scores.length`

- `length` holds the number of elements (not the largest index!)
- Length is not a method so there is no parenthesis at the end unlike String class `length()` method

Basic Array Examples

[Array3.java](#)

[ReverseOrder.java](#)

```

//*****
//  ReverseOrder.java      Author: Lewis/Loftus
//
//  Demonstrates array index processing.
//*****

import java.util.Scanner;

public class ReverseOrder
{
    //-----
    //  Reads a list of numbers from the user, storing them in an
    //  array, then prints them in the opposite order.
    //-----
    public static void main (String[] args)
    {
        Scanner scan = new Scanner (System.in);

        double[] numbers = new double[10];

        System.out.println ("The size of the array: " + numbers.length);
    }
}

```

continue

continue

```
for (int index = 0; index < numbers.length; index++)
{
    System.out.print ("Enter number " + (index+1) + ": ");
    numbers[index] = scan.nextDouble();
}

System.out.println ("The numbers in reverse order:");

for (int index = numbers.length-1; index >= 0; index--)
    System.out.print (numbers[index] + " ");

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

Sample Run

The size of the array: 10

Enter number 1: 18.36

Enter number 2: 48.9

Enter number 3: 53.5

Enter number 4: 29.06

Enter number 5: 72.404

Enter number 6: 34.8

Enter number 7: 63.41

Enter number 8: 45.55

Enter number 9: 69.0

Enter number 10: 99.18

The numbers in reverse order:

99.18 69.0 45.55 63.41 34.8 72.404 29.06 53.5 48.9 18.36

Array Initializers

- An array initializer combines the declaration, creation, and initialization of an array in one statement using the following syntax:

```
elementType [] arrayReferenceVariable = {value0, value1, ... , valueK};
```

Initializer Lists

- An *initializer list* can be used to instantiate and fill an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] units = {147, 323, 89, 933, 540,  
              269, 97, 114, 298, 476};
```

```
char[] grades = {'A', 'B', 'C', 'D', 'F'};
```

Initializer Lists

- Note that when an initializer list is used:
 - the `new` operator is not used
 - no size value is specified
- The size of the array is determined by the number of items in the list
- An initializer list can be used only in the array declaration
- See [Primes.java](#)

```

//*****
// Primes.java      Author: Lewis/Loftus
//
// Demonstrates the use of an initializer list for an array.
//*****

public class Primes
{
    //-----
    // Stores some prime numbers in an array and prints them.
    //-----
    public static void main (String[] args)
    {
        int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};

        System.out.println ("Array length: " + primeNums.length);

        System.out.println ("The first few prime numbers are:");

        for(int i=0; i < primeNums.length; i++){

            System.out.print (primeNums[i] + " ");

        }
    }
}

```

Output

```
Array length: 8
The first few prime numbers are:
2 3 5 7 11 13 17 19
```

```

//*****
// Primes.java
//
// Demonstrate
//*****

public class Primes
{
    //-----
    // Stores some prime numbers in an array and prints them.
    //-----
    public static void main (String[] args)
    {
        int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};

        System.out.println ("Array length: " + primeNums.length);

        System.out.println ("The first few prime numbers are:");

        for(int i=0; i < primeNums.length; i++){

            System.out.print (primeNums[i] + " ");

        }

    }
}

*****
array.
*****
```

Example:

Find the frequency of the result of throwing a dice 6000 times

- See [RollDice.java](#)

for-each loops

- for-each loop (enhanced for loop) enables you to traverse the array sequentially without using an index variable

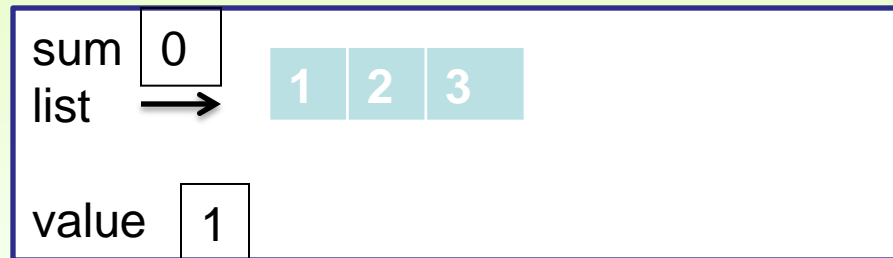
```
for (double d: array)
    System.out.println(d);
```

is equivalent to

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

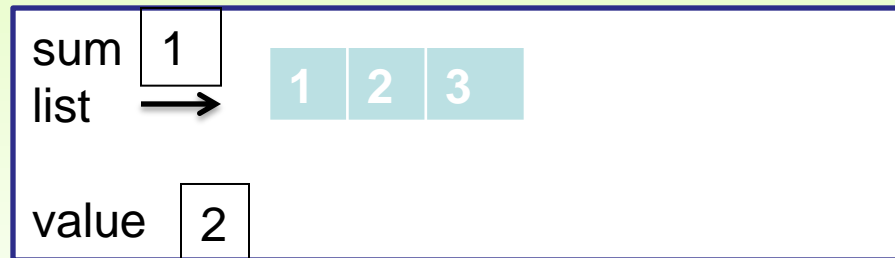
Another example

```
int sum = 0;  
int[] list = {1, 2, 3};  
  
for (int value : list){  
    sum += value;  
}
```



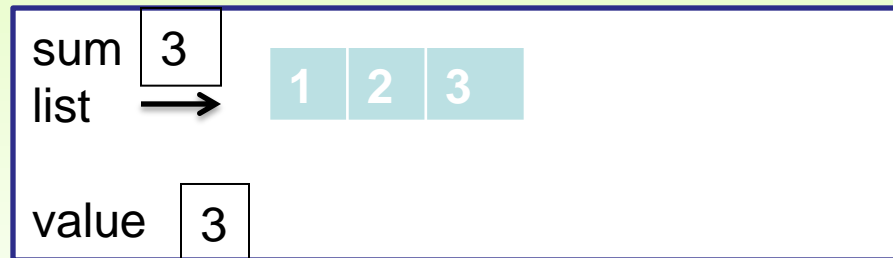
Another example

```
int sum = 0;  
int[] list = {1, 2, 3};  
  
for (int value : list){  
    sum += value;  
}
```



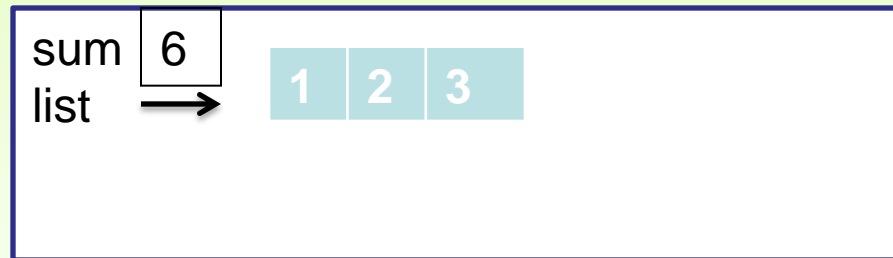
Another example

```
int sum = 0;  
int[] list = {1, 2, 3};  
  
for (int value : list){  
    sum += value;  
}
```



Another example

```
int sum = 0;  
int[] list = {1, 2, 3};  
  
for (int value : list){  
    sum += value;  
}
```



Example

- [InitializerList.java](#) demonstrates the usage of array initializers and for-each loops.

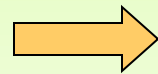
Outline

Method OverLoading

printf method

Arrays

Declaring and Using Arrays



Arrays of Objects

Array as Parameters

Variable Length Parameter Lists

split() Method from String Class

Integer & Double Wrapper Classes

Two-Dimensional Arrays

Arrays of Objects

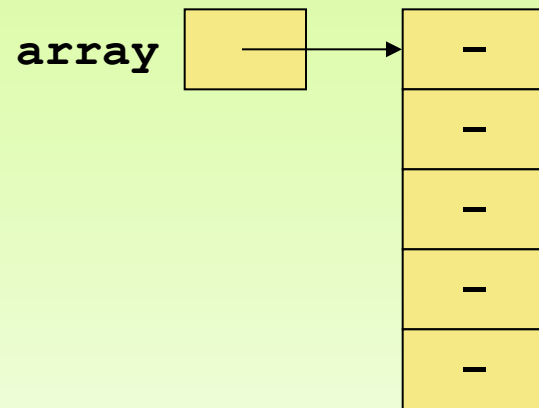
- The elements of an array can be object references
- The following declaration reserves space to store 5 references to `String` objects

```
String[] array = new String[5];
```

- It does NOT create the `String` objects themselves
- Initially an array of objects holds `null` references
- Each object stored in an array must be instantiated separately

Arrays of Objects

- The `array` array when initially declared:



- At this point, the following line of code would throw a `NullPointerException`:

```
System.out.println(array[0].length() );
```

- See [ArrayOfStrings.java](#)

Arrays of Objects

- Keep in mind that `String` objects can be created using literals
- The following declaration creates an array object called `verbs` and fills it with five `String` objects created using string literals

```
String[] verbs = {"play", "work", "eat",  
                 "sleep", "run"};
```

See [StringArr.java](#)