### Topic 2.2 - Arrays

#### Introduction to Arrays

**Given the rainfall for 12 months of 2000 compute the average.**

double fall1, fall2, fall3, …. fall12; System.out.prinltn(“R.Fall month 1 : “); fall1 = console.nextDouble(); System.out.prinltn(“R.Fall month 2 : “); fall2 = console.nextDouble();

...

System.out.prinltn(“R.Fall month 12 : “); fall12 = console.nextDouble();

double total = fall1 + fall2 + … fall12; double average = total/12;

**fall1 fall1**

**fall12**

Too many variables. Prone to errors. Not practical!

Too many variables. Prone to errors. Not practical !

20

**Arrays help us avoid declaring variables**

double[] fall = new double[12]; System.out.prinltn(“R.Fall month 1 : “);

fall[0] = console.nextDouble();

System.out.prinltn(“R.Fall month 2 : “); fall[1] = console.nextDouble();

...

System.out.prinltn(“R.Fall month 12 : “); fall[11] = console.nextDouble();

double total = fall[0]+fall[1]... fall[11];

double average = total/12;

**fall**

**fall[0] fall[1] fall[2]**

**fall[11]**

Does it make life easy? Not really! Many repetitive statements!

21

#### Using a for loop to step through an array

double[] fall;

fall = new double[12]; // declaring array

// Reading values into array

**fall[0] fall[1] fall[2]**

for (int i = ; i < ; i++ ) {

System.out.prinltn(“R.Fall month”+i+”: “);

= console.nextDouble();

}

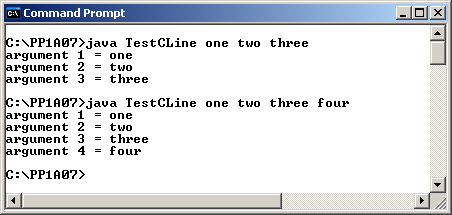
// Finding sum of array double total = ;

for (int i =

; i < ; i++)

**fall[11]**

total += ; average = total/12;



22

Command Line arguments

•**Recall the main method takes as arguments a reference to an array of String:** **public static void main(String[] args)**

•**Arguments can be passed from another method or command line.**

•**The following program reads all the arguments and displays them one after another. The length field is an instance of an array object.**

public class TestCLine{

public static void main (String[] args)

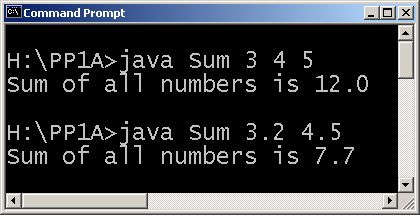
{ for (int i=0; i<args.length; i++) System.out.println("argument "+(i+1+" = "+ args[i]);

}

}

Sample Input/Output

23



Command Line arguments: Application

•Complete the program below for reading all numbers from command lines, adding and displaying sum.

•Hint: Double.parseDouble(String s) converts s to double equivalent.

Sample Input/Output

public class Sum{

public static void main (String[] args){ double total = 0;

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

...

System.out.println("Sum of all numbers is " + total);

}

}

24

|  |
| --- |
| What will be the output? (Trace) 25 |

|  |  |
| --- | --- |
| What will be the output? (Trace)   |  | | --- | | 26 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Arrays help us avoid declaring variables **The quantity of 5 items purchased and their unit prices are stored in arrays qty and unit respectively. Complete the program below to find the total cost.**   |  |  |  |  | | --- | --- | --- | --- | | qty[0] | 1.5 | unit[0] | 7.0 | | qty[1] | 5.0 | unit[1] | 6.0 | | qty[2] | 0.5 | unit[2] | 2.5 | | qty[3] | 2.5 | unit[3] | 4.5 | | qty[4] | 3.0 | unit[4] | 3.0 | |   import java.util.\*; public class Array2  { public static void main(String args[])  {  int vals[] = new int[5];  double qty[] = {1.5,5.0,0.5,2.5,3.0};  double unit[] = {7.0,6.0,2.5,4.5,3.0};  double tCost=0; … …  …  System.out.println("Total cost =“ + tCost);  }  }  27 |

## 

|  |
| --- |
| Exercise (Manipulating Array)   * The Fibonacci series s given by * 1 1 2 3 5 8 …   (3rd and subsequent terms sum of previous terms)   * You are required to find the first 20 elements using an array which has first two elements set to 1. * Finally print the array using a loop.   28 |

|  |
| --- |
| Exercise (Manipulating Array)    29 |

|  |
| --- |
| Array of Strings  * You are required to write a program that allows users to enter up to 20 titles of books storing the values entered in array of String references. * Then prompt user to enter the title he/she is searching. Display all matching titles (either partial or complete). * For partial matching you may use the method   int indexOf(String s) .  30 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array of Strings (cont)  **import java.util.\*;**  **public class ArrayOfString{**   |  |  | | --- | --- | | Book[0] | “Intro to IT” | | Book[1] | “Java for beginners” | | Book[2] | “ Statistics 1” | | Book[3] | “The Double Helix” | | Book[4] | “Chemistry for Dummies” |   **public static void main(String args[]){**  **String Book[] = new String[5];**  **Book[0] = “Intro to IT”;**  **Book[1] = “Java for Beginners”;**  **Book[2] = “Statistics 1”;**  **Book[3] = “The Double Helix”;**  **Book[4] = “Chemistry for Dummies”;**  **System.out.println("The Book titles are:\n”);**  **for ( int i=0;i<Book.length;i++){**  **System.out.println(Book[i]+ “ ”);**  **}**  **}**  **}**  31 |

|  |
| --- |
| Array of Strings (cont)  Displaying book titles and book authors stored in different arrays  **import java.util.\*;**  **public class ArrayOfStringEx2{**  **public static void main(String args[]){**  **String Book[] = new String[5];**  **Book[0] = “Intro to IT”;**  **Book[1] = “Java for Beginners”;**  **Book[2] = “Statistics 1”;**  **Book[3] = “The Double Helix”;**  **Book[4] = “Chemistry for Dummies”;**  **String[] Authors={"Christene. C", "Peter. M ","Graig**  **K.","Graeme D.","Alex S."};**  **for ( int i=0;i<Book.length;i++){**  **System.out.println("The Author of Book '"+Book[i]+**  **"' is "+ Authors[i]);**  **}**  **}**  **}**  32 |

|  |
| --- |
| Array of Strings (cont)  Searching for all available books using a search keyword of a given length  **import java.util.\*;**  **public class ArrayOfStringEx2{**  **public static void main(String args[]){**  **Scanner sc=new Scanner (system.in);**  **String Book[] = new String[5];**  **Book[0] = “Java for Dummies”;**  **Book[1] = “Chemistry for Dummies”;**  **Book[2] = “Statistics 1”;**  **Book[3] = “The Double Helix”;**  **Book[4] = “Java for Advanced Programmers”;**  **String[] Authors={"Christene. C", " Alex S.", "Graig**  **K.","Graeme D."," Peter. M"};**  **System.out.println("Enter the search keyword”);**  **String keyword=sc.nextLine();**  **for ( int i=0;i<Book.length;i++){**  **if(keyword.substring(0,4).equals(Book[i].substring(0,4))){**  **System.out.println("Book '"+Book[i]+**  **"' Author is "+ Authors[i]);**  **}**  **}**  **}**  33 |

|  |
| --- |
| Array of Strings (cont)  Searching for all available books using the first letter ( alphabet)  **import java.util.\*;**  **public class ArrayOfStringEx2{**  **public static void main(String args[]){**  **Scanner sc=new Scanner (system.in);**  **String Book[] = new String[6];**  **Book[0] = “Java for Dummies”;**  **Book[1] = “Chemistry for Dummies”;**  **Book[2] = “Intro to Statistics”;**  **Book[3] = “The Double Helix”;**  **Book[4] = “Java for Advanced Programmers”;**  **Book[5] = “Intro to Business Statistics”;**  **String[] Authors={"Christene. C", " Alex S.", "Graig**  **K.","Graeme D."," Peter. M", “Craig H.”};**  **System.out.println("Enter the first letter of the book ”);**  **String first\_letter=sc.nextLine();**  **for ( int i=0;i<Book.length;i++){**  **if (first\_letter.charAt(0)==(Book[i].charAt(0))){ System.out.println("Book '"+Book[i]+**  **"' Author is "+ Authors[i]);**  **}**  **}**  **}**  **}**  34 |

|  |
| --- |
| Methods in Java  A Java method is a collection of statements that are grouped together to perform an operation.  **public static int methodName (int a, int b) {**  **//body**  **}**  **public static -> modifiers**  **int-> return type**  **methodName -> name of the method**  **a,b -> formal parameters**  **int a, int b -> list of parameters**  Method definition consist of a method header and a method body  **modifier returnType nameOfMethod(Parameter List) {**  **// method body**  **}**  35 |

|  |
| --- |
| Methods in Java (cont…)  **modifier:** defines the access type of the method and it is optional to use.  **returnType:** a value that may be returned by a method.    **nameOfMethod:** This is the method name. The method name and the parameter list form the *method signature.*  **Parameter List:** The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain *zero* parameters.  **method body:** The method body defines what the method does with statements.  36 |

|  |
| --- |
| Why use methods?   * It is common (good) practice to break up the implementation of a program into different methods, where each method performs a specific task that the larger program requires. * Methods often require some information to be supplied in order to perform the required task and can optionally return a result to the caller. * Any methods that you define must be part of a class – they cannot exist on their own. * Many common tasks have been implemented as methods in the Java API classes!   37 |

|  |
| --- |
| Method implementation   * The formal parameters in a method are essentially variables containing information that the caller has supplied. * You can (and should) declare local variables within a method if it needs to store or “remember” values as part of the task it is performing. * If the method has a non-void return type then it must return a literal value or variable of the corresponding type.   **public int multiply (int a, int b){**  **int product; // declaring local variable**  **// code to perform required task**  **product = a \* b;**  **// return statement needed here**  **return product;**  **}**  38 |

|  |
| --- |
| Calling methods  When creating a method, one defines what a method is to do. To use a method to perform its function, one has to call or invoke it.  There are two ways to call a method; the choice is based on whether the method returns a **value** or not and whether the method you call is in the **same class** or not ( *to be discussed later*)  When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method‐ending closing brace is reached.   * If the method returns a value, a call to the method is usually treated as a value. * If the method returns void, a call to the method must be a statement.   39 |

|  |
| --- |
| An example to demonstrate how to define a method and how to call it:  **class MethodDemo1 {**  **public static void main(String [] args){**  **int x = 5, y = 2;**  **int result;**  **result = multiply(x, y); // method call System.out.println(x + “ \* “ + y + “ = “ + result);**  **}**  **// method must be static because main() is static public static int multiply (int a, int b) {**  **int product;**  **product = a \* b;**  **return product; // return value to caller**  **}**  **}**  40 |

|  |
| --- |
| Parameter passing / return values  * In the previous example the variables ‘a’ and ‘b’ were passed along as actual parameters to the method call. * For primitive type parameters a copy of the value stored in the each actual parameter is passed along to the corresponding formal parameter within the method. * A copy of the value being returned is sent back to be stored in the corresponding variable   **result = multiply(x, y);**  41 |

|  |
| --- |
| More on return values  * It is possible to ignore the value being returned by a method.   **class MethodDemo2 {**  **public static void main(String [] args) {**  **int x = 5, y = 2;**  **multiply(x, y); // return value is ignored**  **}**  **// method must be static because main() is static public static**  **int multiply (int a, int b) {**  **int product;**  **product = a \* b;**  **return product;**  **}**  **}**  42 |

|  |
| --- |
| Methods without return values  * It is also possible to have methods which do not return a value to the caller by specifying the return type as ‘**void**’.   **class MethodDemo3 {**  **public static void main(String [] args) {**  **int x = 5, y = 2;**  **multiply(x, y); // no return value**  **}**  **// use void for the return type if the method**  **// does not need to return a value**  **public static void multiply (int a, int b) {**  **int product;**  **product = a \* b;**  **System.out.println(a + “ \* “ + b + “ = “ + product);**  **}**  **}**  43 |

|  |
| --- |
| Passing Parameter by Value **public class swappingExample {**  **public static void main(String[] args) {**  **int x = 30; int y = 45;**  **System.out.println("Before swapping, x = " + x + "**  **and y = " + y);**  **// Invoke the swap method**  **swapFunction(x, y);**  **System.out.println("\n Now, Before and After swapping values**  **will be same here :");**  **System.out.println("After swapping, x = " + x + "**  **and y is " + y);**  **}**  **public static void swapFunction(int x, int y) {**  **System.out.println("Before swapping(Inside), x = " + x + "**  **y= " + y);**  **// Swap x with y**  **int z = x;**  **x = y;**  **y = z;**  **System.out.println("After swapping(Inside), a = " + a + "**  **b = " + b);**  **}**  **}** 44 |

|  |  |
| --- | --- |
| |  | | --- | | * When a method is called, the actual parameter values are computed and passed into the formal parameter variables. * Thus, when a primitive is passed, any changes made to the formal parameter (the copy) will not affect the actual parameter (original) in the calling method. * However, when a reference to an object is passed, the formal parameters in the called method can be used to directly manipulate the original object. |  Primitives & Object references asarguments 45 |

|  |
| --- |
| Sample Program: Passing Primitives & Object references **class MyInt {**  **public MyInt(int n) { val = n; }**  **public void add(int n) { val += n; }**  **public int getVal() { return val; }**  **private int val;**  **}**  **public class PrimsAndObjects {**  **public static void main (String[] args) {**  **int num1 = 10;**  **MyInt num2 = new MyInt(10);**  **addOne(num1, num2);**  **System.out.println("primitive num1 = " + num1+**  **" object referred by num2 = " + num2.getVal());**  **}**  **public static void addOne(int x, MyInt y) {**  **x += 1;**  **y.add(1);**  **}**  **}**  46 |