Java Basics cont..

Lecture 3

Content

- Relational Operators
- Control Statements
- Arrays
- Input/Output

Relational Operators (conditional statements and loop tests)

- Java has six relational operators that compare two numbers and return a boolean value. The relational operators are <, >, <=, >=, ==, and !=.

x < y	Less than	True if x is less than y, otherwise false.
x > y	Greater than	True if x is greater than y, otherwise false.
x <= y	Less than or equal to	True if x is less than or equal to y, otherwise false.
x >= y	Greater than or equal to	True if x is greater than or equal to y, otherwise false.
x == y	Equal	True if x equals y, otherwise false.
x != y	Not Equal	True if x is not equal to y, otherwise false.

Relational Operators (conditional statements and loop tests)

- Here are some code snippets showing the relational operators.

```
boolean test1 = 1 < 2;  // True. One is less that two.
boolean test2 = 1 > 2;  // False. One is not greater than two.
boolean test3 = 3.5 != 1;  // True. One does not equal 3.5
boolean test4 = 17*3.5 >= 67.0 - 42; //True. 59.5 is greater than 5
boolean test5 = 9.8*54 <= 654; // True. 529.2 is less than 654
boolean test6 = 6*4 == 3*8; // True. 24 equals 24
boolean test7 = 6*4 <= 3*8; // True. 24 is less than or equal to 24
boolean test8 = 6*4 < 3*8; // False. 24 is not less than 24
```

Control Statements

- Decision Making Statements
  - If-else
  - If - else if
  - switch
- Loop
  - For
  - While
  - Do while
- Branching
  - Break
  - continue

Control Statements

- Decision making statements
  - if statement
    - The statements under the ‘if’ block are executed only when the condition evaluates to true.
    - Syntax:
      - If (condition)
      - {
      - //statements
      - }

Decision making statements

- **Class** Great{
-
- **Public static void** main(**String** args[]){
-
- **int** a = 10, b = 5;
-
- **if**(a>b)
-
- {
-
-    System.out.println(“a **is** greater than b”);
-
- }
-
- }
- Output:
- a **is** greater than b

Decision making statements

- If-else
  - If then else statement provides two paths. The if block is executed when the condition holds true. When the condition evaluates to false, the statements inside the else block are executed.
  - Syntax:
    - if(condition)
    - { //statements, when condition is true
    - }
    - else
    - {
    - //statements, when condition is false
    - }

Decision Making Statements

- Class Great{
- Public static void main(String args[]){
-             int a = 3, b = 5;
- if(a>b)
- {
-    System.out.println(“a is greater than b”);
- }
- else
- {
-    System.out.println(“b is greater than a”);
- }
- }
- Output:
    - b is greater than a

Decision Making Statement

- If – else if
- **Class** Grade {
- **Public static void** main(**String** args[]) {
- **int** marks = 88;
- **if**(marks > 89)
- {
- System.out.println(“Distinction”);
- }
- **else if**(marks >79)
- {
- System.out.println(“First **class**”);
- }
- **else if**(marks >69)
- {
- System.out.println(“Second **class**”);
- }
- **else if**(marks >49)
- {
- System.out.println(“Pass”);
- }
- **else**   // executed when all the above conditions are false
- {
- System.out.println(“Fail”);
- }
- Output:
    - First class

Decision Making Statements

- Switch
  - ‘switch’ statement can have multiple execution paths. It is similar to that of ‘if else-if’ statement except that switch can handle expressions which results to any primitive data type and if statements handle only boolean expressions.
- Syntax:
- switch (expression)
- {
- case ‘value1’:
- //statements
- break;
- case ‘value2’:
- //statements
- break;
- case ‘value3’:
- //statements
- break;
- default:
- //statements
- }

Loop

- For loop
    - For loop executes a set of statements repeatedly until a specified condition evaluates to false
    - Syntax:
    - for (initialization; condition; increment/decrement)
    - {
    - //statements to be repeated
    - }

Loop

- **for**(**int** i = 0; i < 5; i++)
- {
- System.out.println(“Number”+i);
- }
- Output:
  - Number:0
  - Number:1
  - Number:2
  - Number:3
  - Number:4

Loop

- While
    - While loop executes a set of statements repeatedly until a given condition remains true.
    - Syntax:
    - while(condition)
    - {
    - //statements
    - }

Loop

- **int** i = 5;
- **while** (i != 0)
- {
- System.out.println(“value:”+i);
- --i;
- }
- Output:
  - value:5
  - value:4
  - value:3
  - value:2
  - Value:1

Loop

- Do while loop is similar to that of the while loop except that the condition is evaluated at the end of the loop in do-while whereas in while loop, it is evaluated before entering into the loop.
- Syntax:
- do
- {
- //statements
- }while(condition)

Loop

- **int** i = 0;
- **do**
- {
- System.out.println(“value:”+i);
- i++;
- } **while** (i<6)
- Output:
  - Value: 0
  - Value: 1
  - Value: 2
  - Value: 3
  - Value: 4
  - Value: 5

Branching – Break Statement

- When a program executes a break statement, then the loop (for/switch/while/do while) which contains the break statement is terminated. The flow of the execution jumps to the outside of the loop.

Branching – Break statements

```
Class Find{

public static void main(String args[]) {

int arr[] = {1,2,3,4,5,6,7};

boolean found = false;

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

{ if(arr[i]== 5)

   { found = true;

     break;

    }

}

if(found)

{System.out.println("5 is found");

}

}
```

Output:
5 is found

Branching – Continue statement

- Continue statement skips the current iteration of the loop and evaluates the loop’s condition for the next iteration. Rest of the statements of the loop after the continue statement is not executed and the next iteration is followed.
- The difference between break and continue is that continue statement breaks the current iteration of the loop whereas break statement terminates the loop itself.

Branching - Continue Statement

```
public class ContinueExample {
    public static void main(String a[]){
        for(int i=1;i<=10;i++){
            if(i%2 == 0){
             continue;
         }
         System.out.println(i);
      }
   }
}
```

- Output:

```
1
3
5
7
9
```

Array

- Array is an object, which stores a group of elements of the same data type.
- Array is index based and the first element of an array starts with the index 0.
- The size of array is fixed.
- Array is Index based and hence accessing a random element and performing any operations over the elements such as sorting, filling etc, can be easily performed.
- The size of an array is fixed and hence cannot grow or shrink at runtime.
- Single Dimensional Arrays
- Multidimensional Arrays

Array – Single Dimension

- Syntax for declaration
  - DataType[] Array_name;
  - **Example:**
  - int[ ] Roll;
  - Char[ ] Name;
- Syntax for instantiation
  - Array_name = new datatype[size];
  - **Example:**
  - Roll = new int[10];   //can store 10 integer values in the array ‘Roll’
  - Declaration and instantiation can be done in a single line.
  - **Example:**   int Roll[ ] = new int[5];

Array – Single Dimension

- Array initialization
  - Roll[1] = 101;
  - Roll[2] = 102;
- Array declaration, instantiation and initialization can be done in a single line.
  - **Example:**
  - int arr[ ] = {45,56,67,65,44}; // arr[] has 5 elements
- ACCESSING ELEMENTS OF AN ARRAY
  - Array elements can be accessed by using the index value of the element.
  - Roll[4] denotes the element in the 5th position since arrays starts with the index 0. Generally, ‘for’ loop is used to iterate over the elements of an array.

Array – Single Dimension

```
class ArrayExample{
public static void main(String args[]){
int arr[] = new int[5];                    //declaration, instantiation

//initialization of array using for loop

for(int i = 0; i<arr1.length; i++) {       //length is a property of array
(returns size)
arr[i] = i;
}

for(int i=0;i<arr.length;i++)              //printing array
System.out.println(arr[i]);

}

}
```

Output:

```
0
1
2
3
4
```

Array - Multidimension

- Arrays can have more than one dimension, which means arrays can themselves contain arrays i.e. Array of arrays.
- Declaration, Instantiation, Initialization
- **Example:**
- int[][] a = new int[2][2];   //2 Dimensional array with 2 rows & 2 columns
- a = {{10,20},{30,40}};   //initialization
  - a[0][0] = 10, a[0][1] = 20, a[1][0] = 20 & a[1][1] = 40
- Two dimensional arrays are treated like matrix. The first index represents the row and the second represents the column.

Array - Multidimension

```
class MultiDimArray{
public static void main(String args[]){
int marks[][]={{98,90,95},{53,56,57}};     //declaring and
initializing 2D array

for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
System.out.print(arr[i][j]+" ");
}
System.out.println();
}
}
}
```

Output:

```
98 90 95
53 56 57
```