Unit 2
Tokens, Expressions and Control Structures

Tokens

- A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:
  - Keywords
  - Identifiers
  - Constants
  - Special Symbols
  - Operators

Keywords

- Keywords are pre-defined or reserved words in a programming language.
- Each keyword is meant to perform a specific function in a program.
- Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed.

- Keywords supported by Java:

abstract	assert	boolean
break	byte	case
catch	char	class
const	continue	default
do	double	else
enum	exports	extends
final	finally	float
for	goto	if
implements	import	instanceof
int	interface	long
module	native	new
open	opens	package
private	protected	provides
public	requires	return
short	static	strictfp
super	switch	synchronized
this	throw	throws
to	transient	transitive
try	uses	void
volatile	while	with

Identifiers

- Identifiers are used as the general terminology for naming of variables, functions and arrays.
- These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character.
- Identifier names must differ in spelling and case from any keywords.
- We cannot use keywords as identifiers; they are reserved for special use.

- Valid Identifiers:
  - myVariable
  - MYVARIABLE
  - x
  - x1
  - _myvariable
  - $myvariable, etc
- Invalid Identifiers
  - my variable //contains space
  - 123abc //starts with a digit
  - a+c //+ sign is not alphanumeric character
  - sum_&_difference //ampersand is not an alphanumeric character

Constants/Literals

- Constants are also like normal variables.
- But, the only difference is, their values can not be modified by the program once they are defined.
- Constants refer to fixed values.
- They are also called as literals.
- Constants may belong to any of the data type.

Special Symbols

❖ The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.

- **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
- **Parentheses():** These special symbols are used to indicate function calls and function parameters.
- **Braces{}:** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
- **comma (,):** It is used to separate more than one statements like for separating parameters in function calls.
- **semi colon :** It is an operator that essentially invokes something called an initialization list.
- **asterick (*):** It is used to create pointer variable.
- **assignment operator:** It is used to assign values.

Primitive Data Types in Java

- The primitive data types are the predefined data types. They specify the size and type of any standard values.
- Java has 8 primitive data types.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Non-Primitive Data Types In Java

- Non-primitive data types are called **reference types** because they refer to objects.
- Some non primitive data types are String, Arrays, Class.

User Defined Data Types in Java

- Java User-defined data types are the customized data types created by the developers by exploiting the special features offered by the Java language.
- User-defined datatypes allow us to define datatypes that model the structure and behavior of the data in our applications.
- These data types have versatile characteristics and behavior and it provides greater flexibility to developers in improving their productivity and maintainability of the programs.
- Two major User define data types are:
  1. Class
  2. Interface

Variables in Java

- A variable is a named location in memory to hold data
- Declaration Syntax:

  data_type identifier;

  e.g. int x;
- Java is a statically-typed language. It means that all variables must be declared before they can be used
- The value of a variable can be changed in the program, hence the name variable. For example:

```
int speed=40;

speed=80;

System.out.println(speed);  //prints 80
```

Variables in Java-Example

FirstJavaApplication - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- FirstJavaApplication
  - Source Packages
    - firstjavaapplication
      - FirstJavaApplication.java
  - Libraries

main - Navigator

Members <empty>

- FirstJavaApplication
  - main(String[] args)

Start Page | FirstJavaApplication.java

Source History

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package firstjavaapplication;
7
8  public class FirstJavaApplication {
9
10     public static void main(String[] args) {
11         int x=10;
12         int y=20;
13         int sum = x+y;
14         System.out.println("Sum is "+sum);
15     }
16
17 }
18
```

firstjavaapplication.FirstJavaApplication > main

Output - FirstJavaApplication (run)

```
run:
Sum is 30
BUILD SUCCESSFUL (total time: 5 seconds)
```

Literals In Java

- Literals are data used for representing fixed values, which can be used directly in the code

a. Boolean Literal: Has two values, false and true

b. Integer Literal: An integer literal is a numeric value without any fractional or exponential part.

c. Floating-point Literals: A floating-point literal is a numeric literal that has either a fractional form or an exponential form

d. Character Literals: Character literals are Unicode character enclosed inside single quotes

e. String Literals: String literals are enclosed inside double quotes

Literals In Java-Example

FirstJavaApplication - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- FirstJavaApplication
  - Source Packages
    - firstjavaapplication
      - FirstJavaApplication.java
  - Test Packages
  - Libraries
  - Test Libraries

FirstJavaApplication - Navigator

Members <empty>

- FirstJavaApplication
  - main(String[] args)

Start Page | FirstJavaApplication.java

Source | History

```
 * and open the template in the editor.
 */
package firstjavaapplication;

public class FirstJavaApplication {

    public static void main(String[] args) {
        boolean isOn = true;
        boolean isOff = false;
        int valuel = 50;
        float floatValue= (float) 55.54;
        char gender='M';
        String name="John";
        System.out.println("Boolean literal is "+ isOn+" and "+isOff);
        System.out.println("Integer Literal is "+ valuel+" and Floating point Literal is "+floatValue);
        System.out.println("Character Literal is "+ gender+" and String Literal is "+ name);

    }

}
```

firstjavaapplication.FirstJavaApplication

Output - FirstJavaApplication (run)

```
run:
Boolean literal is true and false
Integer Literal is 50 and Floating point Literal is 55.54
Character Literal is M and String Literal is John
BUILD SUCCESSFUL (total time: 0 seconds)
```

Type Casting in Java

- The process of converting the value of one data type (int, float, double, etc.) to another data type is known as typecasting.
- Type casting of the primitive datatypes can be done in two ways namely:
  1. Widening
  2. Narrowing

Widening

- Converting a lower datatype (having smaller size) to a higher datatype (having larger size) is known as widening.
- In this case the casting is done automatically (since there is no loss in data)therefore, it is known as implicit type casting

byte -> short -> char -> int -> long -> float -> double

Type Casting in Java

- The process of converting the value of one data type (int, float, double, etc.) to another data type is known as typecasting.
- Type casting of the primitive datatypes can be done in two ways namely:
  1. Widening
  2. Narrowing

Widening

- Converting a lower datatype (having smaller size) to a higher datatype (having larger size) is known as widening.
- In this case **the casting is done automatically (since there is no loss in data)therefore, it is known as implicit type casting**

byte -> short -> char -> int -> long -> float -> double

Type Casting in Java

Narrowing

- Converting a higher datatype to a lower datatype is known as widening.
- This needs to be done manually by a programmer since there can be loss of data; therefore, it is also called explicit type casting

double -> float -> long -> int -> char -> short -> byte

Type Casting in Java-Example

Start Page | FirstJavaApplication.java

Source | History

```java
public class FirstJavaApplication {

    public static void main(String[] args) {
        //widening example
        int integerValuel= 20;
        float floatingValue= integerValuel;
        System.out.println("Integer Value = "+ integerValuel+" after widening is "+ floatingValue);
        //narrowing example
        double doubleValue = 10.2345;
        int integerValue = (int) doubleValue;
        System.out.println("Double Value = "+ doubleValue+" after narrowing is "+ integerValue);

    }

}
```

firstjavaapplication.FirstJavaApplication > main

Output - FirstJavaApplication (run)

```
run:
Integer Value = 20 after widening is 20.0
Double Value = 10.2345 after narrowing is 10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Default Variable Initializations

- Variable initialization means assigning value to a variable for the first time.
- When we declare a variable without assigning it an explicit value, the Java compiler will assign a default value.

FirstClass.java

```java
public class FirstClass {

        int number;         // number will have default value: 0
        float ratio;        // default value: 0.0
        boolean success;    // default value: false
        String name;        // default: null
        public void print() {
        System.out.println("number = " + number);
        System.out.println("ratio = " + ratio);
        System.out.println("success = " + success);
        System.out.println("name = " + name);
         }

          public static void main(String[] args) {
           FirstClass fc = new FirstClass();
           fc.print();
    }

}
```

firstclass.FirstClass > main

Terminal - localhost | Output - FirstClass (run) | Exception Reporter

```
run:
number = 0
ratio = 0.0
success = false
name = null
BUILD SUCCESSFUL (total time: 2 seconds)
```

Command Line Argument in Java

- Command Line Argument in Java is the information that is passed to the program when it is executed.
- In the command line, the arguments passed from the console can be received in the java program and they can be used as input.
- The users can pass the arguments during the execution bypassing the command-line arguments inside the main() method.
- When command-line arguments are supplied to JVM, **JVM wraps these and supplies them to args[].**
- We can specify the command line arguments as

  java class_name arg1 arg2 arg3 ……..
- <u>We need to pass the arguments as space-separated values.</u>

Command Line Argument in Java-Steps to do

1. Create a text file on any location you want let it be Desktop
2. Write your java code in that text file and save it with .java extension. Let the file name be Abcd.java.
3. Now open command prompt. Using command prompt, compile Abcd.java file using the command below:

   javac Abcd.java

4. After compilation a .class file named Abcd.class will be created. Now we run that file as

   java Abcd Apple Ball Cat Dog

Where Apple, Ball, Cat, Dog are the arguments passed

Command Line Argument in Java-Example

Java File

Abcd - Notepad

File Edit Format View Help

```
public class Abcd {
    public static void main(String[] args) {
        // TODO code application logic here
        int i;
              for(i=0;i<args.length;i++){
                  System.out.println("Passed Arguments are: "+ args[i]);
              }
    }

}
```

Compiling

Command Prompt

```
C:\Users\Sudar\Desktop>javac Abcd.java

C:\Users\Sudar\Desktop>_
```

Running Through Command Line

Command Prompt

```
C:\Users\Sudar\Desktop>javac Abcd.java

C:\Users\Sudar\Desktop>java Abcd Apple Ball Cat
Passed Arguments are: Apple
Passed Arguments are: Ball
Passed Arguments are: Cat

C:\Users\Sudar\Desktop>_
```

Arrays of Primitive Data Types

- An array is a collection of similar data types.
- Array is a container object that hold values of homogenous type.
- It is also known as static data structure because size of an array must be specified at the time of its declaration.
- An array can be either primitive or reference type.
- It gets memory in heap area. Index of array starts from zero to size-1.
- Syntax:

```
datatype[]  identifier;
or
datatype identifier[];
```

Arrays of Primitive Data Types

- Examples:

```
int[] arr;
int arr[];
char[] arr;
char arr[]
etc
```

Arrays of Primitive Data Types-Initialization of Array

❖ Array can be initialized as:

int arr[] ={10,20,30,40}

Or

Using new keyword as:

int arr[] = new int[10]

Where 10 is the size of array

Arrays of Primitive Data Types-Initialization and Accessing Integer Array

```
 * @author Sudar
 */
public class FirstClass {

        public static void main(String[] args) {

            int arr[]= {10,20,30};
            System.out.println("First Element is "+ arr[0]);
            System.out.println("First Element is "+ arr[1]);
            System.out.println("First Element is "+ arr[2]);

        }

    }
```

```
run:
First Element is 10
First Element is 20
First Element is 30
BUILD SUCCESSFUL (total time: 0 seconds)
```

Arrays of Primitive Data Types-Initialization and Accessing Character Array

Assignment

Comment Syntax

```
4   public class CommentSyntax {
5
6
7       public static void main(String[] args) {
8           //Hi i am writing comment
9           /*In order to comment
10          more than one lines you cna do this as well*/
11          System.out.println("Tis is not comment. If u want to write comment do as above");
12      }
13
14  }
15
```

Output - CommentSyntax (run)

```
run:
Tis is not comment. If u want to write comment do as above
BUILD SUCCESSFUL (total time: 1 second)
```

Garbage Collection in Java

- Garbage Collection in Java is a process by which the programs perform memory management automatically.
- The Garbage Collector(GC) finds the unused objects and deletes them to reclaim the memory.
- Sometimes, the programmer may forget to destroy useless objects, and the memory allocated to them is not released.
- The used memory of the system keeps on growing and eventually there is no memory left in the system to allocate.
- Such applications suffer from "*memory leaks*".
- After a certain point, sufficient memory is not available for creation of new objects, and the entire program terminates abnormally due to OutOfMemoryErrors.

Garbage Collection in Java

- In Java, garbage collection happens automatically during the lifetime of a program. This eliminates the need to de-allocate memory and therefore avoids memory leaks.
- Advantages:

1. It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
2. It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Expressions

- An expression is a construct made up of variables, operators, and method invocations.
- The data type of the value returned by an expression depends on the elements used in the expression.
- The Java programming language allows you to construct compound expressions from various smaller expressions as long as the data type required by one part of the expression matches the data type of the other.
- Java has 3 different types of expressions:

1. **Expressions that Produce a Value**
2. **Expressions that Assign a Variable**
3. **Expressions with No Result**

Java Output

- In Java, we can send output to standard output device (screen) by using:
  - System.out.println(); Prints string inside the quote and then the cursor moves to the beginning of next line

    or
  - System.out.print(); Prints string inside the quotes

    or
  - System.out.printf(); provides the string formatting just in c and c++
- We can print concatenated string using + operator in print function

Reading Input From Console

- To read an input from console, we need to use the Scanner class which is inside java.util.Scanner package.
- So first we need to import that class as

  import java.util.Scanner
- Now we create an object of the class and use any of the available methods found inside the Scanner class

- Methods to read different types of data are:

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Reading Input From Console-Example

```java
package inputexample;
import java.util.Scanner;

public class Inputexample {

    public static void main(String[] args) {
        int num1, num2, sum;
        Scanner sc1 = new Scanner(System.in);
        System.out.println("Enter first number");
        num1 = sc1.nextInt();
        System.out.println("Enter second number");
        num2 = sc1.nextInt();
        sum = num1+num2;
        System.out.println("Sum is "+ sum);
    }

}
```

Output - Inputexample (run) | Exception Reporter

```
run:
Enter first number
30
Enter second number
40
Sum is 70
BUILD SUCCESSFUL (total time: 13 seconds)
```

Operators in Java

- Operators are symbols that perform operations on variables and values.
- Operators in Java can be classified into 6 types:
  - Arithmetic Operators
  - Assignment Operators
  - Relational Operators
  - Logical Operators
  - Unary Operators
  - Bitwise Operators

Operators in Java-Arithmetic Operators

❖ Arithmetic operators are used to perform arithmetic operations on variables and data.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

Operators in Java-Arithmetic operators-Example

```java
package arithmeticoperator;
import java.util.Scanner;
public class ArithmeticOperator {

    public static void main(String[] args) {
        int num1, num2, sumValue, differenceValue, productValue,divisionValue, modulusValue;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter First Number");
        num1 = sc.nextInt();
        System.out.println("Enter Second Number");
        num2 = sc.nextInt();
        sumValue= num1+num2;
        differenceValue= num1-num2;
        productValue= num1*num2;
        divisionValue= num1/num2;
        modulusValue=num1%num2;
        System.out.println("Sum is "+ sumValue+" Difference is "+ differenceValue+" Prdouct Value is "+ productValue
        +" Divison Value is "+ divisionValue+" and Remainder value is "+modulusValue);

    }

}
```

Output

Output - ArithmeticOperator (run) | Exception Reporter

```
run:
Enter First Number
50
Enter Second Number
20
Sum is 70 Difference is 30 Prdouct Value is 1000 Divison Value is 2 and Remainder value is 10
BUILD SUCCESSFUL (total time: 11 seconds)
```

Operators in Java-Assignment Operators

❖ Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

Operators in Java-Assignment Operators-Example

Find the output of following program.

```java
package arithmeticoperator;

import java.util.Scanner;

public class AssignmentOperator {
    public static void main(String[] args) {
        int num1=10, num2=20;
        num1+=num2;
        num1-=25;
        num2*=num1;
        num2/=num1;
        num2%=num1;
        System.out.println("Final Value of num1 is "+ num1+" and num2 is "+num2);
    }
}
```

Operators in Java-Assignment Operators-Example

Find the output of following program.

```java
package arithmeticoperator;

import java.util.Scanner;

public class AssignmentOperator {
    public static void main(String[] args) {
        int num1=10, num2=20;
        num1+=num2;
        num1-=25;
        num2*=num1;
        num2/=num1;
        num2%=num1;
        System.out.println("Final Value of num1 is "+ num1+" and num2 is "+num2);
    }
}
```

Output

```
arithmeticoperator.AssignmentOperator > main >
Output - ArithmeticOperator (run)   Exception Reporter
run:
Final Value of num1 is 5 and num2 is 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operators in Java-Relational Operators

- Relational operators are used to check the relationship between two operands.

Operator	Description	Example
==	Is Equal To	3 == 5 returns **false**
!=	Not Equal To	3 != 5 returns **true**
>	Greater Than	3 > 5 returns **false**
<	Less Than	3 < 5 returns **true**
>=	Greater Than or Equal To	3 >= 5 returns **false**
<=	Less Than or Equal To	3 <= 5 returns **true**

Operators in Java-Relational Operators-Example

```java
package arithmeticoperator;

public class RelationalOperator {

    public static void main(String[] args){
        int a = 7, b = 11;
        // == operator
        System.out.println(a == b);  // false
        // != operator
        System.out.println(a != b);  // true
        // > operator
        System.out.println(a > b);  // false
        // < operator
        System.out.println(a < b);  // true
        // >= operator
        System.out.println(a >= b);  // false
        // <= operator
        System.out.println(a <= b);  // true
    }
}
```

Output

```
run:
false
true
false
true
false
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operators in Java-Logical Operators

- Logical operators are used to check whether an expression is true or false.

Operator	Example	Meaning
&& (Logical AND)	expression1 **&&** expression2	true only if both expression1 and expression2 are true
|| (Logical OR)	expression1 || expression2	true if either expression1 or expression2 is true
! (Logical NOT)	**!**expression	true if expression is false and vice versa

Operators in Java-Logical Operators-Example

```java
package arithmeticoperator;

public class LogicalOperator {
    public static void main(String[] args) {
        System.out.println("Logical Operator Demo");
    // && operator
    System.out.println((5 > 3) && (8 > 5));  // true
    System.out.println((5 > 3) && (8 < 5));  // false
    // || operator
    System.out.println((5 < 3) || (8 > 5));  // true
    System.out.println((5 > 3) || (8 < 5));  // true
    System.out.println((5 < 3) || (8 < 5));  // false
    // ! operator
    System.out.println!(5 == 3));  // true
    System.out.println(!(5 > 3));  // false
  }
}
```

Output - ArithmeticOperator (run) | Exception Reporter

```
run:
Logical Operator Demo
true
false
true
true
false
true
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operators in Java-Unary Operators

❖ Unary operators are used with only one operand.

Operator	Meaning
+	**Unary plus**: not necessary to use since numbers are positive without using it
-	**Unary minus**: inverts the sign of an expression
++	**Increment operator**: increments value by 1
--	**Decrement operator**: decrements value by 1
!	**Logical complement operator**: inverts the value of a boolean

Operators in Java-Unary Operators-example

```java
package arithmeticoperator;

public class UnaryOperator {
    public static void main(String[] args) {
    int a = 12, b = 12;
    int result1, result2;
    System.out.println("Value of a: " + a);
    result1 = ++a;
    System.out.println("After increment: " + result1);
    System.out.println("Value of b: " + b);
    result2 = --b;
    System.out.println("After decrement: " + result2);
  }
}
```

```
run:
Value of a: 12
After increment: 13
Value of b: 12
After decrement: 11
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operators in Java-Bitwise Operators

- Bitwise operators in Java are used to perform operations on individual bits.
- These operators are not generally used in Java

Operator	Description
~	Bitwise Complement i.e. converts 0 to 1 and 1 to 0
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

Operators in Java

❖ Other Operators

1. Instanceof
   - The instanceof operator checks whether an object is an instanceof a particular class
2. Ternary Operator
   - The ternary operator (conditional operator) is shorthand for the if-then-else statement
   - Example:

     variable = Expression ? expression1 : expression2
   - Here's how it works. If the Expression is true, expression1 is assigned to the variable. If the Expression is false, expression2 is assigned to the variable.

Operators in Java-Example

WAP to read two integer numbers from user and display the largest one using Ternary operator.

Operators in Java-Example

WAP to read two integer numbers from user and display the largest one using Ternary operator.

```java
package arithmeticoperator;
import java.util.Scanner;
public class TernaryOperator {
    public static void main(String[] args){
        int num1;
        int num2;
        int val;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter First Number");
        num1=sc.nextInt();
        System.out.println("Enter Second Number");
        num2=sc.nextInt();
        val = (num1>num2)?num1:num2;
        System.out.println("Larget Number is "+ val);
    }
}
```

Output - ArithmeticOperator (run) #3 | Exception Reporter

```
run:
Enter First Number
50
Enter Second Number
20
Larget Element is 50
BUILD SUCCESSFUL (total time: 8 seconds)
```

Java Control Flow

1. Branching Statement
   - If statement
   - Switch statement
2. Looping Statement
   - While
   - Do-while
   - For
3. Jump Statement
   - Break
   - Continue
   - Return

Java Control Flow-Branching Statement

1. Simple if statement

❖ Syntax :

if(*condition*)
statement;

```
int num=40;
if(num%2==0){
    System.out.println("Number is even");
}
```

Java Control Flow-Branching Statement

2. if….else statement

❖ Syntax :

if(*condition*) *statement1*;

else statement2

```java
package controlstructure;

public class ControlStructure {

    public static void main(String[] args) {
        int num =25;
        if(num%2==0){
            System.out.println("Number is even");
        }else{
            System.out.println("Number is odd");
        }

        }
}
```

Java Control Flow-Branching Statement

3. else…if ladder

❖ Syntax :

if(*condition*) *statement1*;

else if (condition2) statement2;

else statement3;

```java
package controlstructure;

public class ControlStructure {

    public static void main(String[] args) {
        int num =20;
        if(num>20){
            System.out.println("Number is greater than 20");
        }else if(num<20){
            System.out.println("Number is less than 20");
        }else{
            System.out.println("Number is exactly 20");
        }

        }
}
```

Java Control Flow-Branching Statement

4. Nested if….else statement

```
public class ControlStructure {
    public static void main(String[] args) {
        int num1=20,num2=30,num3=40;
        if(num1>num2){
            if(num1>num3){
                System.out.println("Num1 is largest");
            }else{
                 System.out.println("Num3 is largest");
            }
        }else{
         if(num2>num3){
             System.out.println("Num2 is largest");
         }else{
             System.out.println("Num3 is largest");
         }
        }

         }
}
```

Java Control Flow-Branching Statement-Switch Statement

Syntax:

```
switch(expression){
    case x:
        //code block
    break;
    case y:
      //code block
    break;
    default:
       //code block
}
```

Java Control Flow-Branching Statement-Switch Statement

```java
package controlstructure;
import java.util.Scanner;
public class SwitchStatement {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String day;
        System.out.println("Enter today's day");
        day = sc.nextLine();
        switch(day){
            case "Sunday":
                System.out.println("Today is Sunday");
                break;
            case "Monday":
                System.out.println("Today is Monday");
                break;
            case "Tuesday":
                System.out.println("Today is Tuesday");
                break;
            default:
                System.out.println("Today is neither Sunday,Monday nor Tuesday");
        }
    }
}
```

Output - ControlStructure (run) | Exception Reporter

```
run:
Enter today's day
Friday
Today is neither Sunday,Monday nor Tuesday
BUILD SUCCESSFUL (total time: 5 seconds)
```

Java Control Flow-Looping Statement-while loop

Syntax:

```
while(condition){
    //code to be executed
}
```

```java
package controlstructure;

public class WhileLoop {
    public static void main(String[] args) {
        int i=1;
        while(i<=5){
            System.out.println("Hello World");
            i++;
        }
    }
}
```

Variables | Breakpoints | Output - ControlStructure (run)

```
run:
Hello World
Hello World
Hello World
Hello World
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java Control Flow-Looping Statement-(do-while loop)

Syntax:

```
do{
    //code to be executed
}while(condition);
```

```java
package controlstructure;

public class WhileLoop {
    public static void main(String[] args) {
        int i=1;
        do{
            System.out.println("Hello World");
            i++;
        }while(i<=5);
    }
}
```

Variables | Breakpoints | Output - ControlStructure (run)

```
run:
Hello World
Hello World
Hello World
Hello World
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java Control Flow-Looping Statement-(for loop)

Syntax:

```
for(initialization; condition; increment/decrement){

    //statement or code to be executed
}
```

```java
package controlstructure;

public class ForLoop {
    public static void main(String[] args) {
        int i;
        for(i=1;i<=5;i++){
            System.out.println("Hello World");
        }
    }
}
```

Variables | Breakpoints | Output - ControlStructure (run)

```
run:
Hello World
Hello World
Hello World
Hello World
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java Control Flow-Looping Statement-(for loop)-Example

```java
package controlstructure;

public class ForLoop {
    public static void main(String[] args) {
        int arr[] = {10,20,30,10,20,40},i;
        for(i=0;i<arr.length;i++){
            System.out.println("arr["+i+"] = "+ arr[i]);
        }
    }
}
```

Variables | Breakpoints | Output - ControlStructure (run)

```
run:
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 10
arr[4] = 20
arr[5] = 40
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java Control Flow-Looping Statement-(for each loop)

```java
package controlstructure;

public class ForEachLoop {
    public static void main(String[] args) {
        int arr[] = {10,20,30,10,20,40};
            for(int a:arr){     //for each loop
                System.out.println("Array elements: "+a);
            }
        }
}
```

Variables | Breakpoints | Output - ControlStructure (run) | Exc

```
run:
Array elements: 10
Array elements: 20
Array elements: 30
Array elements: 10
Array elements: 20
Array elements: 40
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java break statement

- The break statement in Java terminates the loop immediately, and the control of the program moves to the next statement following the loop.
- It is almost always used with decision-making statements

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
while (testExpression) {
    // codes
    while (testExpression) {
        // codes
        if (condition to break) {
            break;
        }
        // codes
    }
    // codes
}
```

break in nested loop

```
label:
for (int; testExpresison, update) {
    // codes
    for (int; testExpression; update) {
        // codes
        if (condition to break) {
            break label;
        }
        // codes
    }
    // codes
}
```

Labelled break in nested loop

Java break statement-Example

```java
package breakstatement;

public class BreakStatement {

    public static void main(String[] args) {
        int i;
        for(i=1;i<=10;i++){
            if(i==5){
                break;
            }
            System.out.println(i);
        }
    }
}
```

Output - BreakStatement (run) | Exception Reporter

```
run:
1
2
3
4
BUILD SUCCESSFUL (total time: 1 second)
```

Java break statement-Example

```java
package breakstatement;
public class BreakStatement {
    public static void main(String[] args) {
        int i,j;

        for(i=1;i<=5;i++){
            for(j=1;j<=5;j++){
                if(j==3){
                    break;
                }
                System.out.println("Value of i is "+i+" and j is "+ j);
            }
        }
    }
}
```

Output??

Java break statement-Example

```java
package breakstatement;
public class BreakStatement {
    public static void main(String[] args) {
        int i,j;

        for(i=1;i<=5;i++){
            for(j=1;j<=5;j++){
                if(j==3){
                    break;
                }
                System.out.println("Value of i is "+i+" and j is "+ j);
            }
        }
    }
}
```

Output

Output - BreakStatement (run) | Exception Reporter

```
run:
Value of i is 1 and j is 1
Value of i is 1 and j is 2
Value of i is 2 and j is 1
Value of i is 2 and j is 2
Value of i is 3 and j is 1
Value of i is 3 and j is 2
Value of i is 4 and j is 1
Value of i is 4 and j is 2
Value of i is 5 and j is 1
Value of i is 5 and j is 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java continue statement

❖ Continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
while (testExpression) {
    // codes
    while (testExpression) {
        // codes
        if (testExpression) {
            continue;
        }
        // codes
    }
    // codes
}
```

continue in nested loop

```
label:
while (testExpression) {
    // codes
    while (testExpression) {
        // codes
        if (testExpression) {
            continue label;
        }
        // codes
    }
    // codes
}
```

labelled continue in nested loop

Java continue statement-Example

```java
package breakstatement;

class ContinueStatement {
    public static void main(String[] args) {
        int i ;
        for(i=1;i<=5;i++){
            if(i==3){
                continue;
            }
            System.out.println("Valie of i = "+i);
        }
    }
}
```

Output - BreakStatement (run) | Exception Reporter

```
Valie of i = 1
Valie of i = 2
Valie of i = 4
Valie of i = 5
BUILD SUCCESSFUL (total time: 1 second)
```