UNIT I - JAVA FUNDAMENTALS

- Java Data types
- Class – Object
- I / O Streams
- File Handling concepts
- Threads
- Applets
- Swing Framework
- Reflection

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

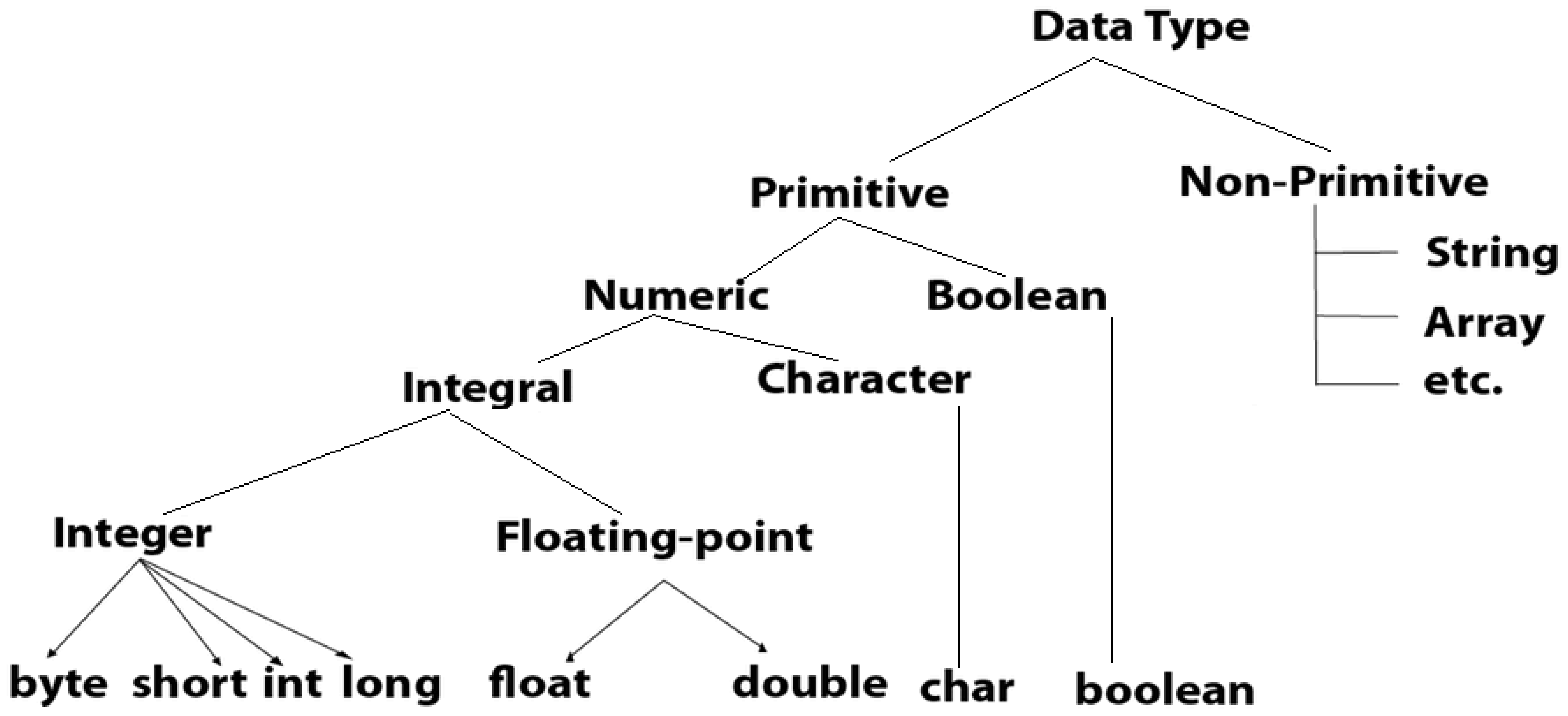
Topics to be covered…

- Java Data types
- Java Literals
- Java Keywords
- Java Variables/ Identifiers
- Java Naming conventions
- Java Operators
- Java Comments
- Reading data from keyboard
- Mathematical functions in Java
- Conditional statements in Java
- Loops in Java
- Java Break Statement
- Java continue Statement

Java Data types

Data Types in Java

- Java language has a rich implementation of data types
- **Data types specify the different sizes and values** that can be stored in the variable.
- In java, data types are classified into two categories :
  - ➢ Primitive Data type
  - ➢ Non-Primitive Data type

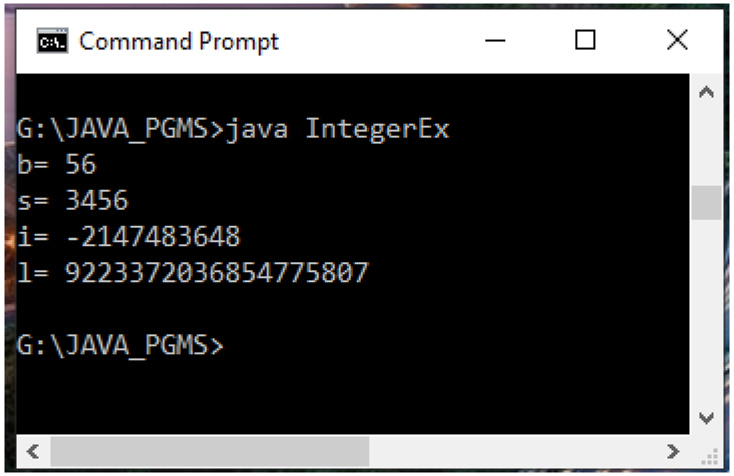


Primitive Data type

- **Integer**
- This group includes byte, short, int, long
  - **byte :**
    - It is 1 byte(8-bits) integer data type.
    - Value range from -128 to 127.
    - Default value zero. example: byte b=10;
  - **short :**
    - It is 2 bytes(16-bits) integer data type.
    - Value range from -32768 to 32767.
    - Default value zero. example: short s=11;
  - **int :**
    - It is 4 bytes(32-bits) integer data type.
    - Value range from -2147483648 to 2147483647.
    - Default value zero. example: int i=10;
  - **long :**
    - It is 8 bytes(64-bits) integer data type.
    - Value range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
    - Default value zero. example: long l=100012;

Integer type data Example

```
public class IntegerEx
{
    public static void main(String[] args)
        {
                // byte data type
                byte b = 56;
                System.out.println("b= "+b);

                // short data type
                short s = 3456;
                System.out.println("s= "+s);

                // int data type
                int i = -2147483648 ;
                System.out.println("i= "+i);

                // long data type
                long l =9223372036854775807L;
                System.out.println("l= "+l);
          }
}
```


Command Prompt

```
G:\JAVA_PGMS>java IntegerEx
b= 56
s= 3456
i= -2147483648
l= 9223372036854775807

G:\JAVA_PGMS>
```

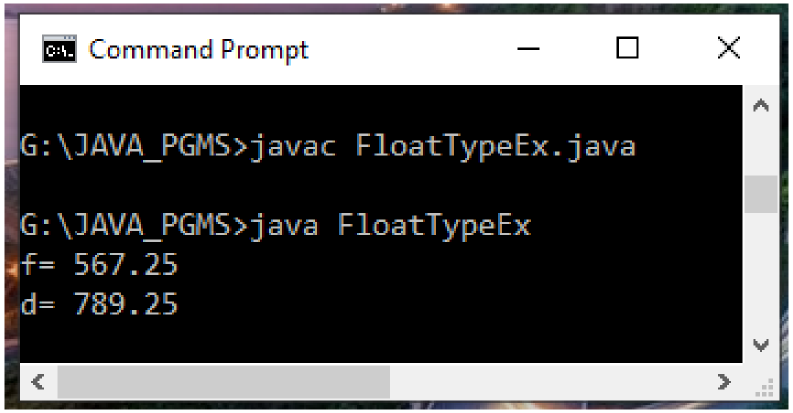
Notice, the use of L at the end of9223372036854775807. This represents that it's an integral literal of the long type.

Primitive Data type Cont'd

- **Floating-Point Number**
- Floating type is **useful to store decimal point values.**
- This group includes float, double
  - **float :**
    - It is 4 bytes(32-bits) float data type.
    - Default value 0.0f. example: float ff=10.3f;
  - **double :**
    - It is 8 bytes(64-bits) float data type.
    - Default value 0.0d. example: double db=11.123;
- Notice that, we have used 10.3f instead of 10.3in the above example. It's because 10.3 is a double literal. To tell the compiler to treat 10.3 as float rather than double, we need to use f or F

Floating point type data Example

```
public class FloatTypeEx
{
   public static void main(String[] args)
   {
            // float data type
            float f = 567.25f;
            System.out.println("f= "+f);

            // double data type
            double d = 789.25;
            System.out.println("d= "+d);

   }
}
```



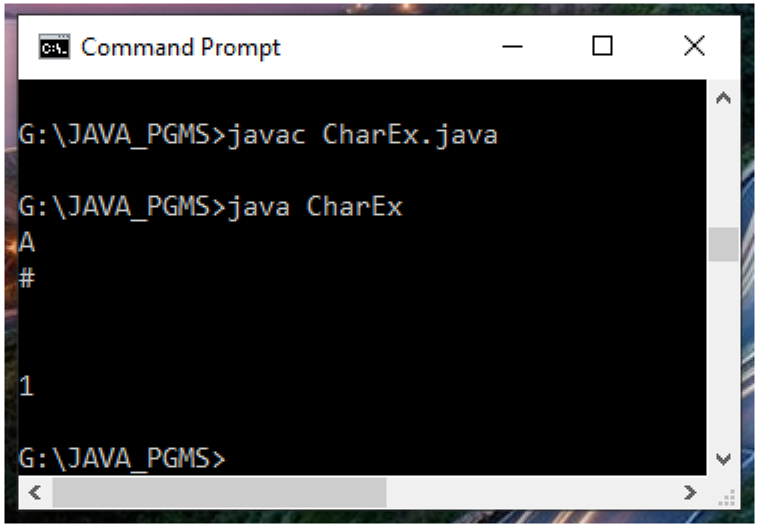
```
G:\JAVA_PGMS>javac FloatTypeEx.java

G:\JAVA_PGMS>java FloatTypeEx
f= 567.25
d= 789.25
```

Primitive Data type Cont’d

- **Characters**
- This group represent char, which represent symbols in a character set, like letters and numbers.
  - **char :**
    - It is 2 bytes(16-bits) unsigned unicode character.
    - Minimum value is '\u0000' (or 0)
    - Maximum value is '\uffff' (or 65,535 inclusive)
    - Char data type is used to store any character
    - Example: char c='a';

Char type data Example

```
public class CharEx
 {
   public static void main(String[] args)
    {
      char ch1 = 'A';
      System.out.println(ch1);

      char ch2 = '#';
      System.out.println(ch2);

      System.out.println('\n');
      char ch3 = '1';
      System.out.println(ch3);
   }
}
```



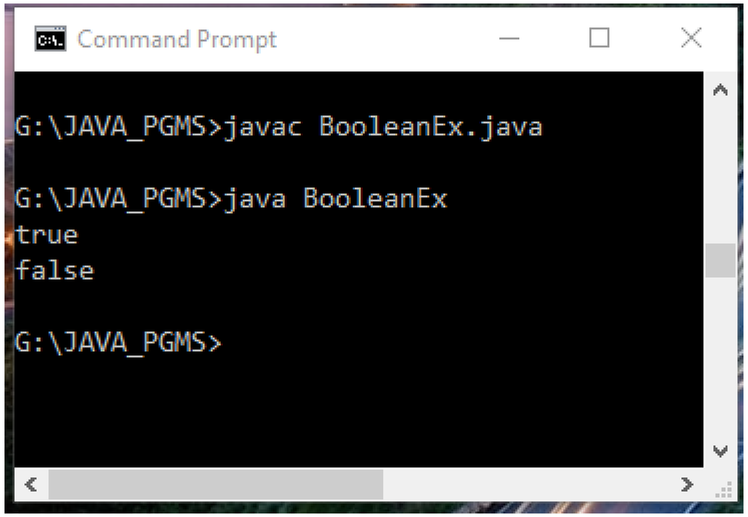
```

G:\JAVA_PGMS>javac CharEx.java

G:\JAVA_PGMS>java CharEx
A
#


1

G:\JAVA_PGMS>
```

Primitive Data type Cont’d

- **Boolean**
  - This group represent boolean
  - They are defined constant of the language.
  - ➢ boolean
    - ▪ The boolean data type has two possible values, either true or false.
    - ▪ Default value: false.
    - ▪ example: boolean b=true;

boolean type data Example

```
public class BooleanEx
 {
   public static void main(String[] args)
   {

      boolean t = true;
      System.out.println(t);

      boolean f = false;
      System.out.println(f);

   }

}
```



```
G:\JAVA_PGMS>javac BooleanEx.java

G:\JAVA_PGMS>java BooleanEx
true
false

G:\JAVA_PGMS>
```

Java Literals

Java literals

- To understand literals, let's take an example to assign value to a variable.

  int x=253

- Here,
  - int →is a data type.
  - x → is variable
  - 253→ is literal.
- A Literal is the source code representation of a fixed value.
- Values like 12.5, 7, true, '\u0050' that appear directly in a program without requiring computation are literals.

Java literals

- **Integer Literals**
  - Integer literals are used to initialize variables of integer data types byte, short, int and long.
  - If an integer literal ends with l or L, it's of type long.
  - Tip: it is better to use L instead of l.

```
IntegerEx.java:19: error: integer number too large: 9223372036854775807
                long l =9223372036854775807;
                        ^
1 error
```

  - Integer literals can be expressed in decimal, hexadecimal and binary number systems.
  - The numbers starting with prefix 0x represents hexadecimal.
  - Similarly, numbers starting with prefix 0b represents binary.

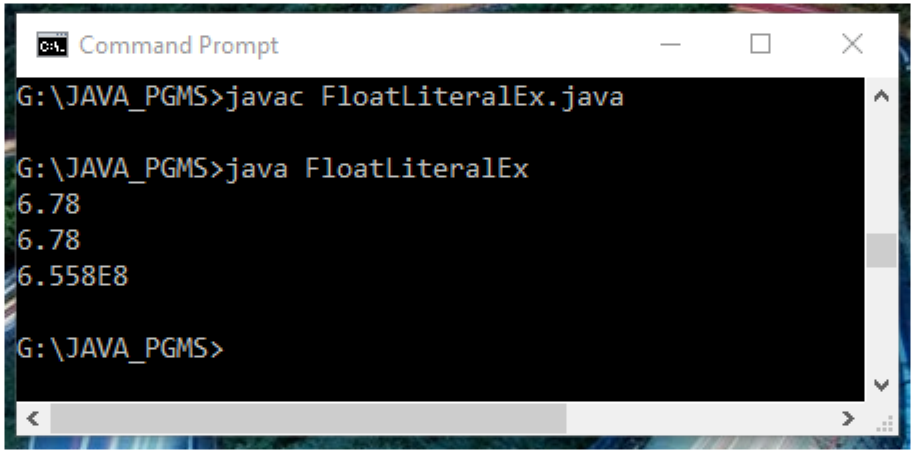
```
// decimal int decNumber = 34;
// 0x represents hexadecimal int hexNumber = 0x2F;
// 0b represents binary int binNumber = 0b10010;
```

Java literals Cont’d

- **Floating-point Literals**
  - ➢ Floating-point literals are used to initialize variables of data type float and double.
  - ➢ If a floating-point literal ends with f or F, it's of type float. Otherwise, it's of type double.
  - ➢ A double type can optionally end with D or d. However, it's not necessary.
  - ➢ They can also be expressed in scientific notation using E or e.

Example

```
class FloatLiteralEx
 {
   public static void main(String[] args)
  {
      double db = 6.78;
      float f = 6.78F;

      // 6.558*10^8
      double dbSci = 6.558e8;

      System.out.println(db);
      System.out.println(f);
      System.out.println(dbSci);
   }
}
```


Command Prompt

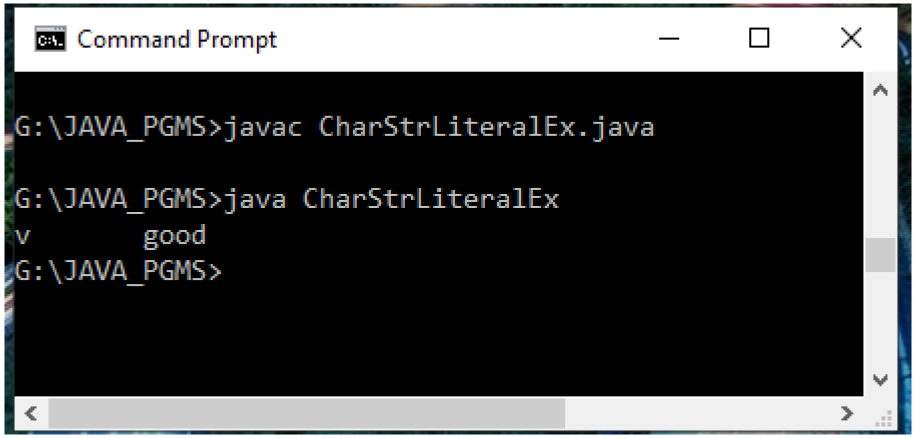
```
G:\JAVA_PGMS>javac FloatLiteralEx.java

G:\JAVA_PGMS>java FloatLiteralEx
6.78
6.78
6.558E8

G:\JAVA_PGMS>
```

Java literals Cont’d

- **Character and String Literals**
  - They contain Unicode (UTF-16) characters.
  - For char literals, single quotations are used. For example, 'a', '\u0111' etc.
  - For String literals, double quotation is used. For example, "programming", "Java 8"
  - Java also supports a few special escape sequences. For example, \b (backspace), \t (tab), \n (line feed), \f (form feed), \r (carriage return), \" (double quote), \' (single quote), and \\ (backslash).

Example

```
class CharStrLiteralEx
{
    public static void main(String[] args)
    {
        char ch = 'v';
        char es = '\t';
        String str = "good";

        System.out.print(ch);
        System.out.print(es);
        System.out.print(str);
    }
}
```

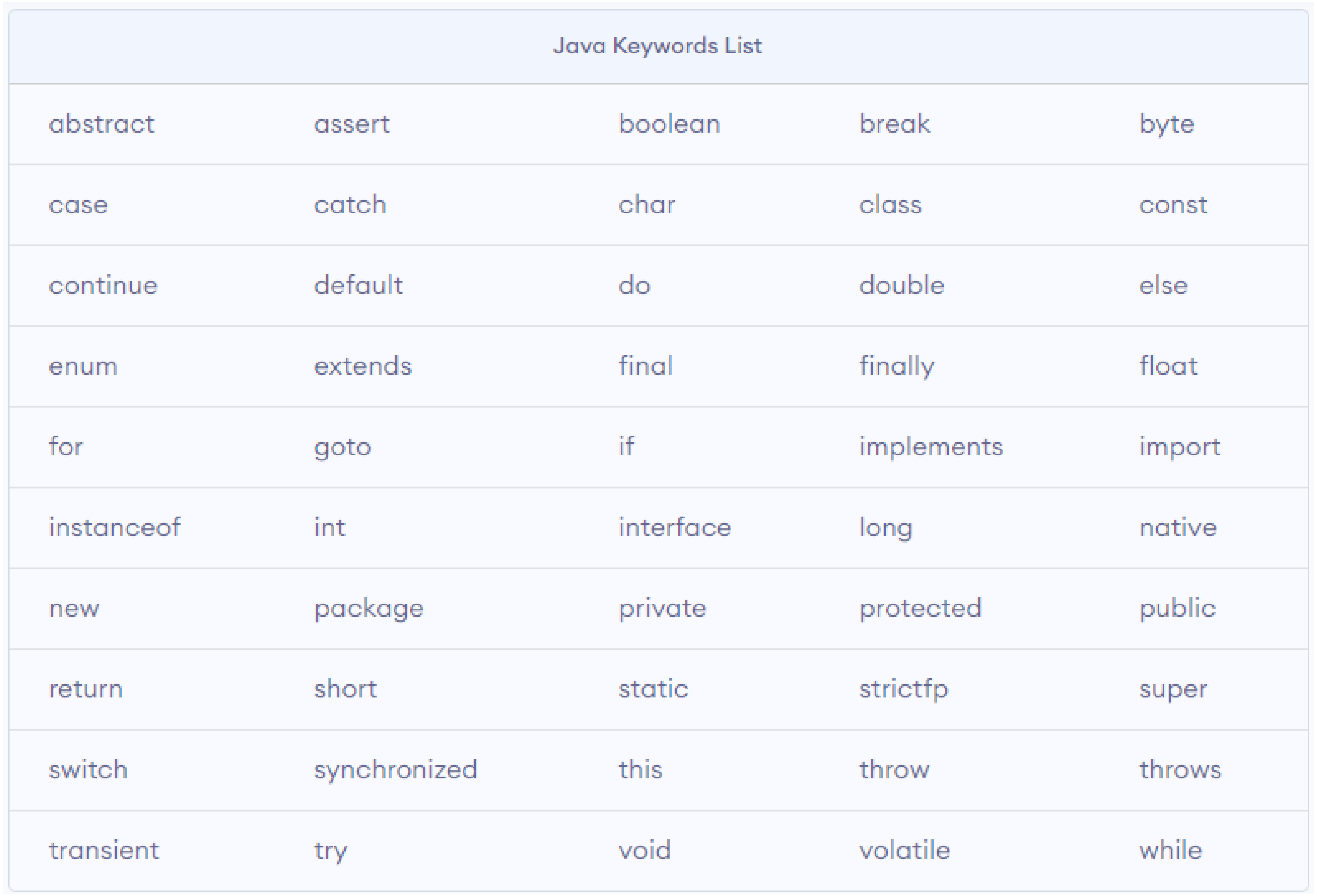


```
G:\JAVA_PGMS>javac CharStrLiteralEx.java

G:\JAVA_PGMS>java CharStrLiteralEx
v       good
G:\JAVA_PGMS>
```

Java Keywords

Java Keywords

- Keywords are **predefined, reserved words** used in Java programming that **have special meanings** to the compiler
- For example: int age;Here, **int is a keyword**



Java Keywords List				
abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

Java Variables/ Identifiers

Java Variables

- A variable is a location in memory (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier)
- To declare the variable in Java, we can use following syntax

  datatype variableName;

- Here, **datatype** refers to type of variable which can any like: **int, float** etc. and **variableName** can be any like: **empId, amount, price** etc.
- For example, int age=21;
- In the example, we have assigned value to the variable during declaration. However, it's not mandatory.
- we can declare variables without assigning the value, and later we can store the value as we wish.

  For example,

  int age;

  age = 21;

Java Variables Cont'd

- The value of a variable can be changed in the program, hence the name 'variable'.

For example,

```
int age = 21;
 ... .. ...
age= 45;
```

- **Java is a statically-typed language**. It means that all variables must be declared before they can be used.
- Also, you cannot change the data type of a variable in Java within the same scope.

```
int age = 21;
... .. ...
 float age;
```

Java Variables Cont'd

- Java Programming language defines mainly three kind of variables.
  - ➢ Local Variables
  - ➢ Instance Variables
  - ➢ Static Variables (Class Variables)
- **Local Variable**
  - ➢ A variable declared **inside the body of the method** is called local variable.
  - ➢ we can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
  - ➢ A local variable cannot be defined with "static" keyword.
- **Instance Variable**
  - ➢ A variable **declared inside the class but outside the body of the method**, is called instance variable.
  - ➢ It is not declared as static.
  - ➢ It is called instance variable because its value is instance specific and is not shared among instances.

Java Variables Cont'd

- **Static variable**
  - A variable which is declared as static is called static variable.
  - It cannot be local.
  - We can create a single copy of static variable and share among all the instances of the class.
  - Memory allocation for static variable happens only once when the class is loaded in the memory.

```
class  Example
{
        int x=25; //instance variable
        static int ct=2; //static variable
        void method()
        {
                int a=45; //local variable
         }
}
```

Java identifiers (variable name)

- Identifiers are the name given to variables, classes, methods, etc.
- Example: int age;
- Here, age is a variable (an identifier).
- **Rules for Naming an Identifier**
  - ➢ Identifiers **cannot be a keyword**.
  - ➢ We cannot also use true, false and null as identifiers. It is because they are literals.
  - ➢ Identifiers are **case-sensitive**.
  - ➢ It can have a sequence of letters and digits. However, it must begin with a letter, $ or _. **The first letter of an identifier cannot be a digit.**
  - ➢ It's a convention to start an identifier with a letter rather and $ or _.
  - ➢ **Whitespaces are not allowed**.
  - ➢ Similarly, we **cannot use symbols** such as @, #, and so on.

Java Naming conventions

Java Naming conventions

- **Java naming conventions** are sort of guidelines
- It helps the application programmers to decide what to name the identifiers such as class, package, variable, constant, method, etc.
- It helps to produce a consistent and readable code throughout the application.
- But, it is not forced to follow. So, it is known as convention not rule.
- These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.
- Java follows **camel-case syntax** for naming the class, interface, method, and variable.
  - If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.

Java Naming conventions Cont’d

- Some other rules that should be followed by identifiers.
- **Variables** (Already explained)
- **Class**
  - It should **start with the uppercase letter**.
  - It should be a **noun** such as Color, Button, System, Thread, etc.
  - Use appropriate words, instead of acronyms.
  - **Example: -**

```
public class Employee {}
 public class Record {}
```

- **Interface**
  - It should **start with the uppercase letter**.
  - It should be an **adjective** such as Runnable, Remote, ActionListener.
  - Use appropriate words, instead of acronyms.
  - **Example: -**

```
public interface Serializable {}
 public interface Clonable {}
```

Java Naming conventions Cont’d

- **Method**
  - It should **start with lowercase letter.**
  - It should be a **verb** such as main(), print(), println().
  - If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
  - **Example :-**

    public Long getId() {}

    public void remove(Object o) {}
- **Package**
  - It should be **a lowercase letter** such as java, lang.
  - If the name contains multiple words, it should be **separated by dots (.)** such as java.util, java.lang.
  - **Example :-**

    package com.google.search.common;
- **Constant**
  - It should be in **uppercase letters** such as RED, YELLOW.
  - If the name contains **multiple words, it should be separated by an underscore(_)** such as MAX_PRIORITY.
  - It may contain digits but not as the first letter.

    Example :-

    public final int INITIAL_SIZE = 16;

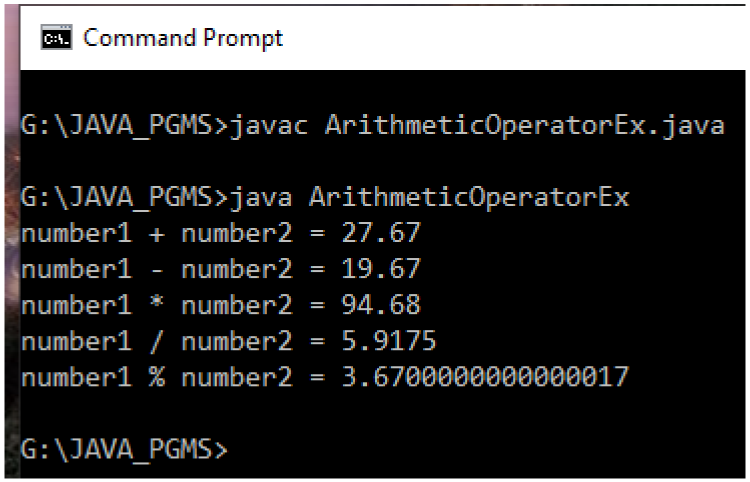
Java Operators

Java Operators

- Operator is a **symbol which tells to the compiler to perform some operation**
- Java provides a rich set of operators do deal with various types of operations
- Java operators can be divided into following categories:
  - Arithmetic operators→+(addition), -(subtraction), *(multiplication), /(division), %(modulus)
  - Increment & Decrement operators→++(increment), -- (decrement)
  - Relation operators→<(less than), <=(less than or equal to), >(greater then), >= (greater than or equal to) ,== (equal to), !=(not equal)
  - Logical operators→&&(Logical AND), || (Logical OR),! (NOT)
  - Bitwise operators→~(1's complement), &(Bitwise AND), |(Bitwise OR), ^ (EXOR), <<(Left Shift), >>(Right Shift),>>> (Unsigned right Shift)
  - Assignment operators→=, += , -=,*= , /= , %=
  - Conditional operators→ ?:
  - Misc operators

Arithmetic Operator Example

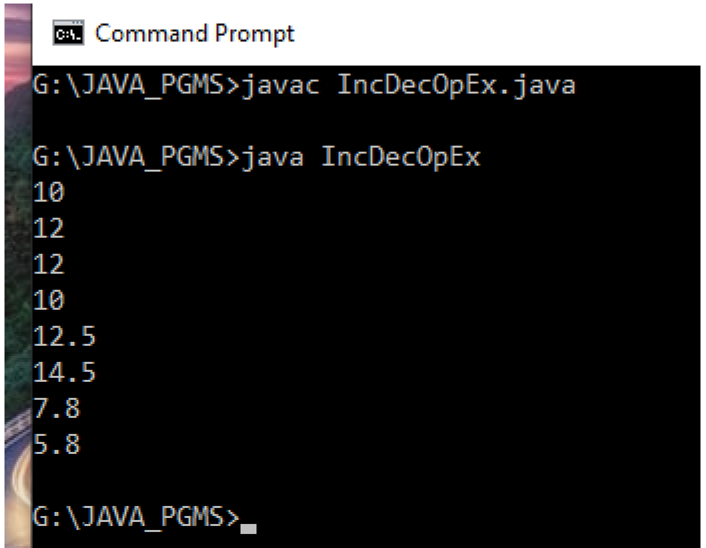
```
class ArithmeticOperatorEx
{
    public static void main(String[] args)
            {
                        double num1 = 23.67,result;
                        int num2 = 4;
                         // Using addition operator
                             result = num1 + num2;
                             System.out.println("number1 + number2 = " + result);
                        // Using subtraction operator
                             result = num1 - num2;
                             System.out.println("number1 - number2 = " + result);
                        // Using multiplication operator
                             result = num1 * num2;
                             System.out.println("number1 * number2 = " + result);
                        // Using division operator
                             result = num1 / num2;
                             System.out.println("number1 / number2 = " + result);
                        // Using remainder operator
                             result = num1 % num2;
                             System.out.println("number1 % number2 = " + result);
             }
}
```



```
G:\JAVA_PGMS>javac ArithmeticOperatorEx.java

G:\JAVA_PGMS>java ArithmeticOperatorEx
number1 + number2 = 27.67
number1 - number2 = 19.67
number1 * number2 = 94.68
number1 / number2 = 5.9175
number1 % number2 = 3.6700000000000017

G:\JAVA_PGMS>
```

Increment Decrement Operator Example

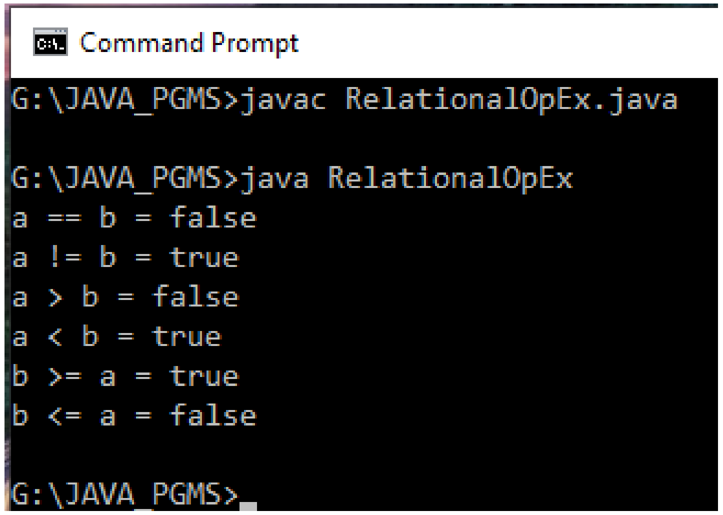
```
class IncDecOpEx
{
	public static void main(String args[])
	{
		int x=10;
		System.out.println(x++);//10 (11)
		System.out.println(++x);//12
		System.out.println(x--);//12 (11)
		System.out.println(--x);//10
		double d=12.5;
		System.out.println(d++);//12.5 (13.5)
		System.out.println(++d);//14.5
		float f=7.8f;
		System.out.println(f--);//7.8 (6.8)
		System.out.println(--f);//5.8
	}
}
```



```
G:\JAVA_PGMS>javac IncDecOpEx.java

G:\JAVA_PGMS>java IncDecOpEx
10
12
12
10
12.5
14.5
7.8
5.8

G:\JAVA_PGMS>
```

Relational Operator Example

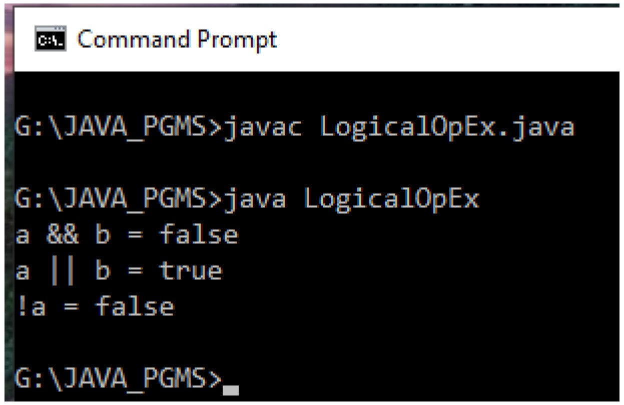
```
class RelationalOpEx
{
 public static void main(String as[])
  {
     int a, b;
     a=45;
     b=90;
     System.out.println("a == b = " + (a == b) );
     System.out.println("a != b = " + (a != b) );
     System.out.println("a > b = " + (a > b) );
     System.out.println("a < b = " + (a < b) );
     System.out.println("b >= a = " + (b >= a) );
     System.out.println("b <= a = " + (b <= a) );
  }
}
```



```
G:\JAVA_PGMS>javac RelationalOpEx.java

G:\JAVA_PGMS>java RelationalOpEx
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false

G:\JAVA_PGMS>
```

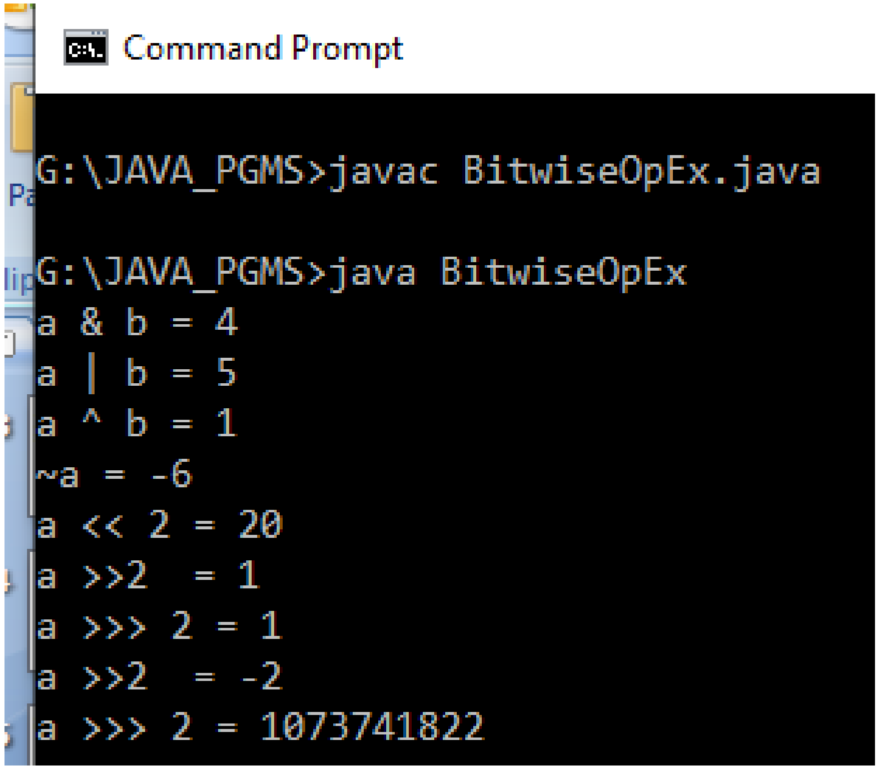
Logical Operator Example

```
class LogicalOpEx
{
public static void main(String as[])
   {
      boolean a = true;
      boolean b = false;
      System.out.println("a && b = " + (a&&b));
      System.out.println("a || b = " + (a||b) );
      System.out.println("!a = " + !a);
   }
}
```


Command Prompt

```
G:\JAVA_PGMS>javac LogicalOpEx.java

G:\JAVA_PGMS>java LogicalOpEx
a && b = false
a || b = true
!a = false

G:\JAVA_PGMS>
```

```
class BitwiseOpEx
{
public static void main(String as[])
   {
      int a = 5;
      int b = 4;
      int c = 0;
      c = a & b;
      System.out.println("a & b = " + c );
      c = a | b;
      System.out.println("a | b = " + c );
      c = a ^ b;
      System.out.println("a ^ b = " + c );
      c = ~a;
      System.out.println("~a = " + c );
      c = a << 2;
      System.out.println("a << 2 = " + c );
      c = a >> 2;
      System.out.println("a >>2  = " + c );
      c = a >>> 2;
      System.out.println("a >>> 2 = " + c );
      a=-5;
      c = a >> 2;
      System.out.println("a >>2  = " + c );
      c = a >>> 2;
      System.out.println("a >>> 2 = " + c );
   }
}
```


Command Prompt

```
G:\JAVA_PGMS>javac BitwiseOpEx.java

G:\JAVA_PGMS>java BitwiseOpEx
a & b = 4
a | b = 5
a ^ b = 1
~a = -6
a << 2 = 20
a >>2  = 1
a >>> 2 = 1
a >>2  = -2
a >>> 2 = 1073741822
```

Difference between >> (Right shift)and >>> (Unsigned right Shift/right shift with zero fill)

```
                      5-->  0000 0000 0000 0000 0000 0000 0000 0101
                     ~5-->1111 1111 1111 1111 1111 1111 1111 1010
                                                                    +
                                                                  1
2's complement of 5(-5)--> 1111 1111 1111 1111 1111 1111 1111 1011
```

-5>>2→ 1111 1111 1111 1111 1111 1111 1111 1110
To represent in Decimal, Since this is negative number (MSB is 1) take 2's complement ,so that it becomes

```
0000 0000 0000 0000 0000 0000 0000 0001
                                     +1
-------------------------------------------------------------
0000 0000 0000 0000 0000 0000 0000 0010 →-2
```

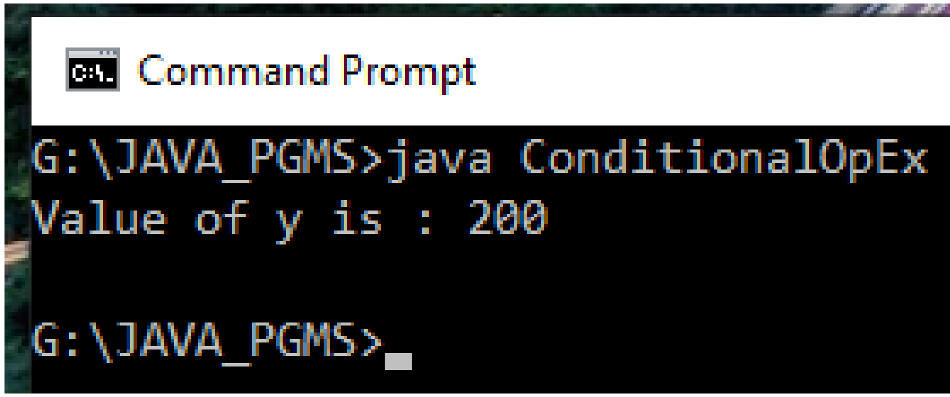
-5>>>2→0011 1111 1111 1111 1111 1111 1111 1110 ==1073741822

Conditional operator

- It is also known as ternary operator because it works with **three operands.**
- **It is short alternate of if-else statement.**
- It can be used to evaluate Boolean expression and **return either true or false** value

epr1 ? expr2 : expr3

```
class ConditionalOpEx
{
public static void main(String as[])
  {
        int x=1,y;
        y=(x==2)?100:200;
        System.out.println( "Value of y is : " +  y );
   }
}
```

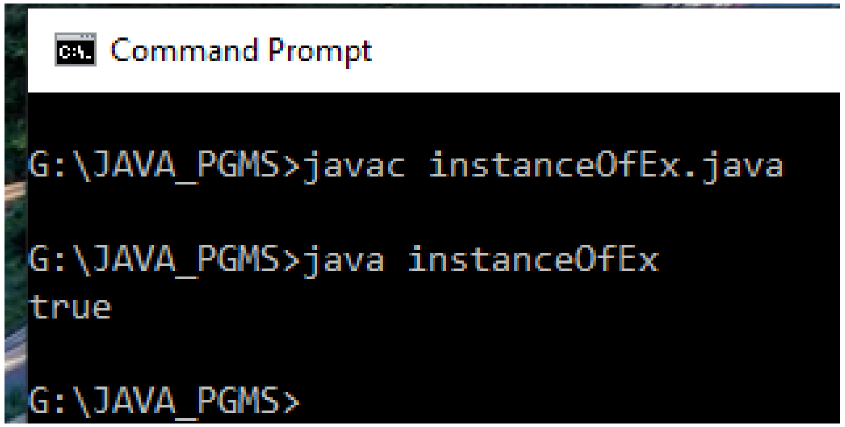


```
G:\JAVA_PGMS>java ConditionalOpEx
Value of y is : 200

G:\JAVA_PGMS>_
```

Misc. operator

- instanceOf operator
  - It is a java **keyword** and used to test whether the given **reference belongs to provided type** or not.
  - Type can be a class or interface.
  - **It returns either true or false**.

```
class instanceOfEx
{
public static void main(String as[])
 {
     String str= "Welcome";
     System.out.println( str instanceof String );
  }
}
```



```
G:\JAVA_PGMS>javac instanceOfEx.java

G:\JAVA_PGMS>java instanceOfEx
true

G:\JAVA_PGMS>
```

Java Comments

Java Comments

- In computer programming, comments are **a portion of the program that are completely ignored by Java compilers.**
- They are mainly used to **help programmers to understand the code.**
- In Java, there are two types of comments:
    - **single-line comment(or) End of Line comment:** starts and ends in the same line. To write a single-line comment, we can use the //
    - **multi-line comment(or) Traditional Comment:** When we want to write comments in multiple lines, we can use the multi-line comment. To write multi-line comments, we can use the /*….*/ symbol

Reading data from keyboard

Reading data from keyboard

- There are many ways to read data from the keyboard.
- For example:
  - ➢ InputStreamReader
  - ➢ Console
  - ➢ Scanner
  - ➢ DataInputStream etc.
- We will learn to get input from user using the object of **Scanner class**.
- The Scanner class is used to read input data from different sources like input streams, users, files, etc.

Scanner class

- To read input using Scanner class,

1) We need to import java.util.Scanner package

```
import java.util.Scanner;
```

2) Create a Scanner Object

Example:

```
// To read input from the input stream
 Scanner sc1 = new Scanner(InputStream input);
// To read input from files
Scanner sc2 = new Scanner(File file);
// To read input from a string
Scanner sc3 = new Scanner(String str);
```

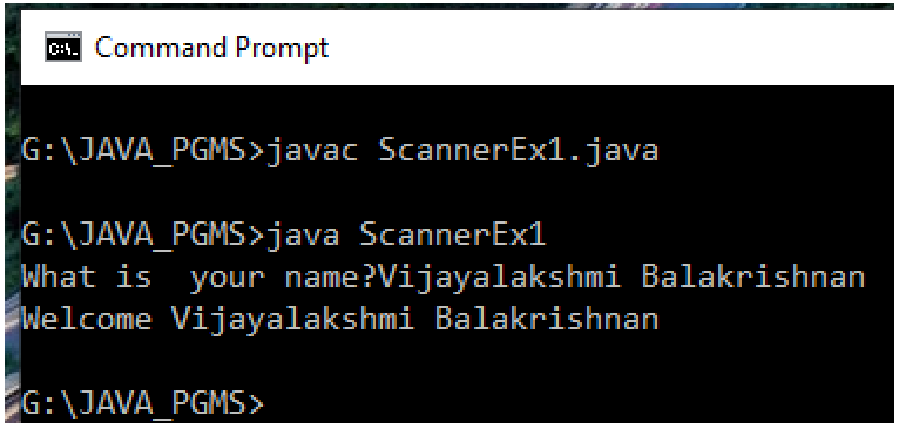
3) Use Scanner methods to get input from the user

Scanner Methods

Method	Description
nextInt()	reads an int value from the user
nextFloat()	reads a float value form the user
nextBoolean()	reads a boolean value from the user
nextLine()	reads a line of text from the user
next()	reads a word from the user
nextByte()	reads a byte value from the user
nextDouble()	reads a double value from the user
nextShort()	reads a short value from the user
nextLong()	reads a long value from the user

Read input from the keyboard Example1

```
//1.Import package
import java.util.Scanner;
class ScannerEx1
 {
   public static void main(String[] args)
    {
      // 2.Creates an object of Scanner
      Scanner in = new Scanner(System.in);
      System.out.print("What is  your name?");
      // 3.Read a line of text from the user
      String name = in.nextLine();
      //4.print output
      System.out.println("Welcome " + name);
      // 5.Closes the scanner
      in.close();
   }
}
```



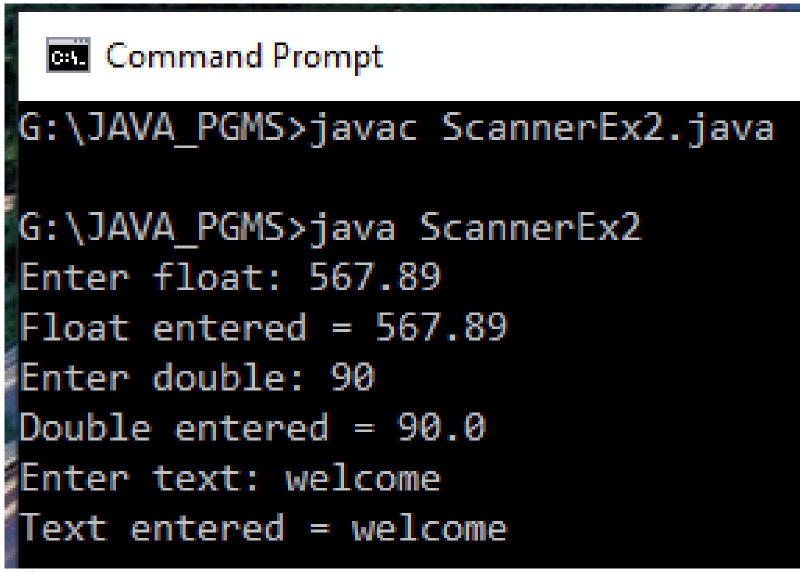
```
G:\JAVA_PGMS>javac ScannerEx1.java

G:\JAVA_PGMS>java ScannerEx1
What is  your name?Vijayalakshmi Balakrishnan
Welcome Vijayalakshmi Balakrishnan

G:\JAVA_PGMS>
```

The System.in parameter is used to take input from the standard input. It works just like taking inputs from the keyboard.

Note: We have used the close() method to close the object. It is recommended to close the scanner object once the input is taken.

Read input from the keyboard Example2

```
import java.util.Scanner;
class ScannerEx2
 {
   public static void main(String[] args)
    {
         Scanner sc = new Scanner(System.in);
         // Getting float input
         System.out.print("Enter float: ");
         float f = sc.nextFloat();
         System.out.println("Float entered = " + f);
         // Getting double input
        System.out.print("Enter double: ");
        double d= sc.nextDouble();
        System.out.println("Double entered = " + d);
         // Getting String input
       System.out.print("Enter text: ");
       String str = sc.next();
       System.out.println("Text entered = " + str);
   }
}
```



```
G:\JAVA_PGMS>javac ScannerEx2.java

G:\JAVA_PGMS>java ScannerEx2
Enter float: 567.89
Float entered = 567.89
Enter double: 90
Double entered = 90.0
Enter text: welcome
Text entered = welcome
```

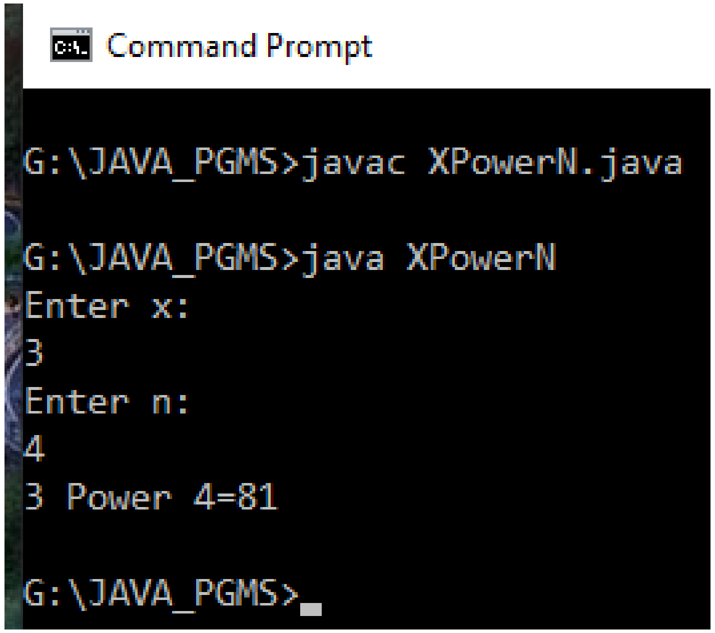
Mathematical functions in Java

Mathematical functions in Java

- Java has a lot of Math methods that allow us to perform mathematical computations.
- The **java.lang.Math** class contains various methods
  - ➢ **Basic Math methods** like abs(), min(), max(), sqrt(), pow(), etc...
  - ➢ **Logarithmic Math Methods** like log(), log10(), exp(), etc..
  - ➢ **Trignometric Math Methods** like sin(), cos(), tan(), asin(), acos(), etc...
  - ➢ **Hyperbolic Math Methods** like sinh(), cosh(),t anh()
  - ➢ **Angular Math Methods** like toDegrees(), toRadians()

Math Function Example

```
//x^n
import java.util.*;
import java.math.*;
class XPowerN
{
 public static void main(String a[])
   {
          int x,n,res;
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter x:");
          x=sc.nextInt();
          System.out.println("Enter n:");
          n=sc.nextInt();
          res=(int)Math.pow(x,n);
          System.out.println(x+" Power "+n+"="+res);
   }
}
```



```
G:\JAVA_PGMS>javac XPowerN.java

G:\JAVA_PGMS>java XPowerN
Enter x:
3
Enter n:
4
3 Power 4=81

G:\JAVA_PGMS>
```

Conditional/Decision making statements in Java

Conditional statements in Java

- In Java, **if statement** is used for testing the conditions.
- The condition matches the statement it returns true else it returns false.
- There are four types of If statement they are:
  - if statement
  - if-else statement
  - if-else-if ladder
  - nested if statement
- switch statement is used for executing one statement from multiple conditions and it is similar to an if-else-if ladder.

Types of If statement

if Statement

The if statement is a single conditional based statement that executes only if the provided condition is true.

Syntax:

```
if(condition)
{
        //code
}
```

if-else Statement

•The if-else statement is used for testing condition. If the condition is true, if block executes otherwise else block executes.

•It is useful in the scenario when we want to perform some operation based on the false result.

•The else block execute only when condition is false.

Syntax:

```
if(condition)
{
        //code for true
}
else
{
        //code for false
}
```

Types of If statement Cont'd

Syntax:

```
if(condition1)
{
        //code for if condition1 is true
}
else if(condition2)
{
        //code for if condition2 is true
}
else if(condition3)
{
        //code for if condition3 is true
}
...
else
{
        //code for all the false conditions
}
```

<u>if-else-if ladder Statement</u>

•When we have multiple conditions to execute then it is recommend to use if-else-if ladder.

•It contains multiple conditions and execute if any condition is true otherwise executes else block.

Types of If statement Cont’d

<u>Nested if statement</u>

In this, one if block is created inside another if block when the outer block is true then only the inner block is executed.

Syntax:

```
if(condition)
{
   //statement
          if(condition)
          {
          //statement
          }
}
```

Java Switch Statement

The switch statement is used for executing one statement from multiple conditions. it is similar to an if-else-if ladder.

Syntax:

```
switch(expression)
{
        case value1:
                //code for execution;
                break;  //optional
        case value2:
                // code for execution
                break;  //optional
        ......
        ......
        case value n:
                // code for execution
                 break;  //optional

        default:
                 code for execution when none of the case is true;
}
```

Loops in Java

Loops in Java

- loops are **used to repeat a specific block** of code until a certain condition is met.
- There are three types of loops in Java.
  - for loop
  - while loop
  - do-while loop

Loops in Java Cont'd

- **for Loop**
- The for loop is used for executing a part of the program **repeatedly**.
- When the number of execution is fixed then it is suggested to use for loop.
- for loop can be categories into two type.
  - ➢ for loop
  - ➢ for-each loop
- **for Loop :**

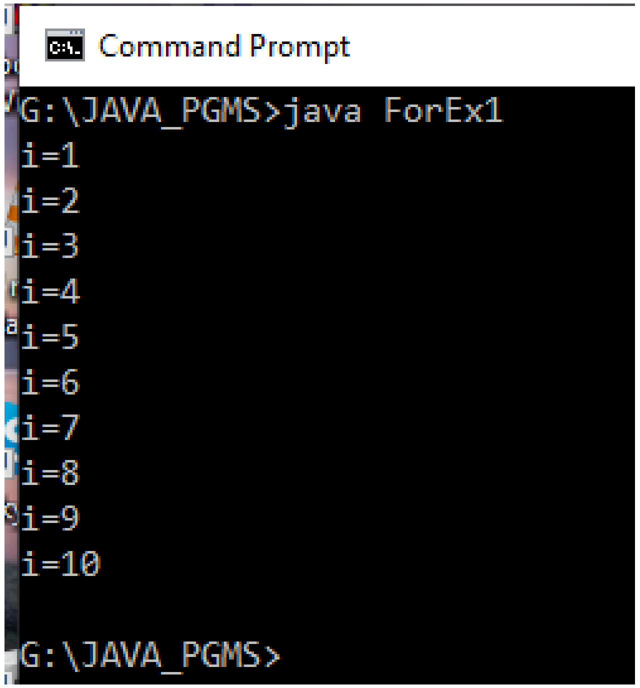
```
for(initialization;condition;increment/decrement)
{
//statement
}
```

- **for-each Loop**
- In Java, for each loop is used for traversing array or collection elements. In this loop, there is no need for increment or decrement operator.
- For-each loop syntax

```
for(Type var:array)
 {
//code for execution
}
```

Simple for example

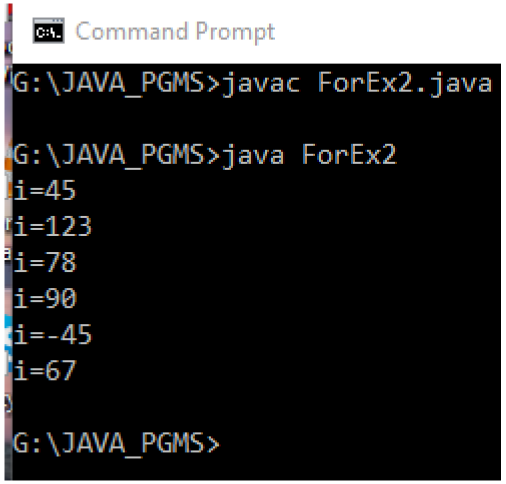
```
class ForEx1
{
        public static void main(String a[])
        {
                for(int i=1;i<=10;i++)
                {
                        System.out.println("i="+i);
                }
        }
}
```



```
G:\JAVA_PGMS>java ForEx1
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10

G:\JAVA_PGMS>
```

for each example

```
class ForEx2
{
        public static void main(String a[])
        {
                int x[]={45,123,78,90,-45,67};
                for(int i:x)
                {
                        System.out.println("i="+i);
                }
        }
}
```



```
G:\JAVA_PGMS>javac ForEx2.java

G:\JAVA_PGMS>java ForEx2
i=45
i=123
i=78
i=90
i=-45
i=67

G:\JAVA_PGMS>
```

Loops in Java Cont'd

- **Java While Loop**
- If the number of iteration is not fixed, it is recommended to use while loop.
- **Syntax:**

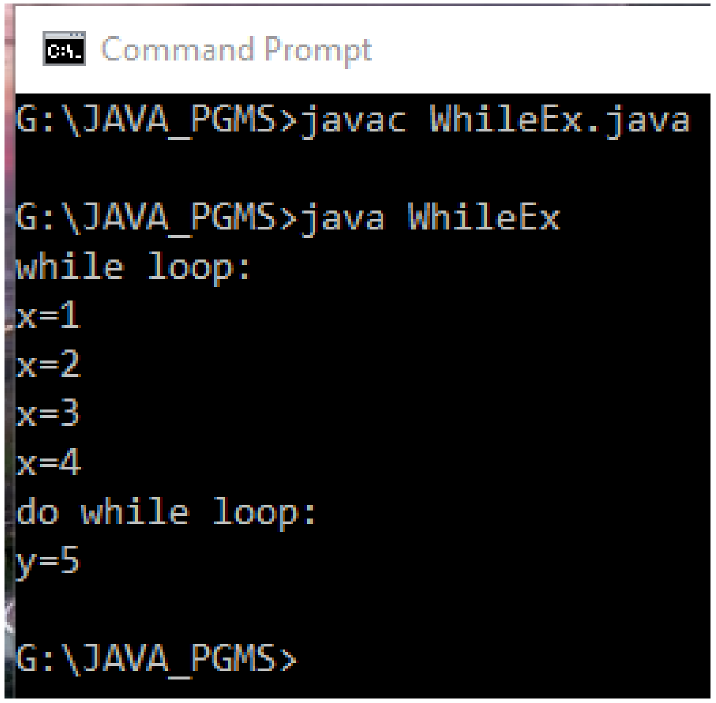
```
while(condition)
{
  //code to be executed
}
```

- **Java do-while Loop**
- If the number of iteration is not fixed and we **must have to execute the loop at least once**, it is recommended to use do-while loop.
- *do-while loop* is executed at least once because condition is checked after loop body.
- **Syntax:**

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

While and do while example

```
class WhileEx
{
  public static void main(String a[])
   {
          int i=4,x=1,y=5;
          System.out.println("while loop:");
          while(x<=i)
          {
                    System.out.println("x="+x);
                    x++;
          }
          System.out.println("do while loop:");
          do
          {
                    System.out.println("y="+y);
          }while(y<=i);
   }
}
```



Command Prompt

```
G:\JAVA_PGMS>javac WhileEx.java

G:\JAVA_PGMS>java WhileEx
while loop:
x=1
x=2
x=3
x=4
do while loop:
y=5

G:\JAVA_PGMS>
```

Java Break & continue Statement

Java Break Statement

- The Java **break statement is used to break loop or switch statement**
- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- In case of inner loop, it breaks only inner loop.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Java continue Statement

- The continue statement in Java skips the current iteration of a loop (for, while, do...while, etc) and the control of the program moves to the end of the loop and, the test expression of a loop is evaluated.