Save the code in Notepad as "Main.java". Open Command Prompt (cmd.exe), navigate to the directory where you saved your file, and type "javac Main.java":

C:\Users\Your Name>javac Main.java

This will compile your code. If there are no errors in the code, the command prompt will take you to the next line. Now, type "java Main" to run the file:

C:\Users\Your Name>java Main

Final Variables

If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

Example

final int myNum = 15;

myNum = 20; // will generate an error: cannot assign a value to a final variable

Scanner sc = new Scanner(System.in);

// System.out.println("enter a no. between 1 to 7 as a weekday");

// int a = sc.nextInt();

char op = sc.next().charAt(0); // to get single char viz +,- etc

sc.close();

Data types are divided into two groups:

Primitive data types - includes byte, short, int, long, float, double, boolean and char

Non-primitive data types - such as String, Arrays and Classes (you will learn more about these in a later chapter)

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type Size

byte 1 byte

short = char 2 bytes

int = float 4 bytes

long = double 8 bytes

boolean 1 bit

long myNum = 15000000000L;

The float and double data types can store fractional numbers. Note that you should end the value with an "f" for floats and "d" for doubles:

float myNum = 5.75f;

System.out.println(myNum);

The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits.

Therefore it is safer to use double for most calculations.

The main difference between primitive and non-primitive data types are:

Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).

Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.

A primitive type has always a value, while non-primitive types can be null.

A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.

The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt); // Outputs 9

System.out.println(myDouble); // Outputs 9.0

double myDouble = 9.78d;

int myInt = (int) myDouble; // Manual casting: double to int

System.out.println(myDouble); // Outputs 9.78

System.out.println(myInt); // Outputs 9

Java divides the operators into the following groups:

Arithmetic operators

Operator

+

-

\*

/

%

++

--

Assignment operators

Operator Example Same As Try it

= 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

Comparison operators

The return value of a comparison is either true or false

Operator Name Example Try it

== Equal to x == y

!= Not equal x != y

> Greater than x > y

< Less than x < y

>= Greater than or equal to x >= y

<= Less than or equal to x <= y

Logical operators

Operator Name Description Example Try it

&& Logical and Returns true if both statements are true x < 5 && x < 10

|| Logical or Returns true if one of the statements is true x < 5 || x < 4

! Logical not Reverse the result, returns false if the result is true !(x < 5 && x < 10)

Bitwise operators

// Initial values

int a = 5;

int b = 7;

// bitwise and

// 0101 & 0111=0101 = 5

System.out.println("a&b = " + (a & b));

Output

a&b = 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

String Operations

String str1 = "Rudra rao" ,str2 ="Welcome Rudra to java";

System.out.println("Uppercase =" + str1.toUpperCase());

System.out.println("Lowercase =" + str2.toLowerCase());

System.out.println("Length =" + str1.length());

System.out.println("Concatenate =" + str1.concat(str2));

System.out.println("Concatenate =" + str1+str2);

System.out.println("Equals = " + str1.equals(str2));//returns T or F, but also compares case.

System.out.println("Equals = " + str1.equals(str1));

System.out.println("Equals = " + (str1==str2));

System.out.println("Equals Ignore Case =" + str1.equalsIgnoreCase(str2));

System.out.println("Equals Ignore Case =" + str1.equalsIgnoreCase("rudra"));

System.out.println("Contains =" + str2.contains(str1) );//used to find reqd words from web application while testing

System.out.println("Contains =" + str2.contains("Rudra"));

System.out.println("Replace = " + str1.replace('u','o'));

System.out.println("Replace = " + str2.replace("Rudra","Ram"));

System.out.println("Replace = " + str1.replaceFirst(str1, str2));

System.out.println("Substring =" + str2.substring(9));

System.out.println("Substring =" + str2.substring(8, 13));// returns 0 to n-1 & used to get any substring from main string

System.out.println("CharAt =" + str1.charAt(0));

StringBuffer sb = new StringBuffer("Welcome");

System.out.println("Reverse =" +sb.reverse());

Output

Uppercase =RUDRA RAO

Lowercase =welcome rudra to java knknk

Length =9

Concatenate =Rudra raoWelcome Rudra to java knknk

Concatenate =Rudra raoWelcome Rudra to java knknk

Equals = false

Equals = true

Equals = false

Equals Ignore Case =false

Equals Ignore Case =false

Contains =false

Contains =true

Replace = Rodra rao

Replace = Welcome Ram to java knknk

Replace = Welcome Rudra to java knknk

Substring =udra to java knknk

Substring =Rudra

CharAt =R

Reverse =emocleW

The indexOf() method returns the index (the position) of the first occurrence of a specified text in a string (including whitespace):

Example

String txt = "Please locate where 'locate' occurs!";

System.out.println(txt.indexOf("locate")); // Outputs 7

Java counts positions from zero.

0 is the first position in a string, 1 is the second, 2 is the third ...

String x = "10";

int y = 20;

int z = x + y;..............INVALID ERROR

String x = "10";

int y = 20;

String z = x + y;..............1020

String txt = "We are the so-called "Vikings" from the north.";...error

The solution to avoid this problem, is to use the backslash escape character.

String txt = "We are the so-called \"Vikings\" from the north.";

Escape character Result Description

\' ' Single quote

\" " Double quote

\\ \ Backslash

public class Main {

public static void main(String[] args) {

String txt = "Hello\nWorld!";

System.out.println(txt);

}

}

Integer myInt = 5;

Double myDouble = 5.99;

Character myChar = 'A';

System.out.println(myInt.doubleValue()); 5.0

System.out.println(myDouble.intValue()); 5

System.out.println(myChar.charValue()); A

Math. max(x,y), min, sqrt,

Math.random() returns a random number between 0.0 (inclusive), and 1.0 (exclusive):

int time = 20;

String result = (time < 18) ? "Good day." : "Good evening.";

System.out.println(result);

SWITCH

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

int day = 4;

switch (day)

{

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

default:

stmt;

}

LOOPS

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}

int i = 0;

do {

System.out.println(i);

i++;

}

while (i < 5);

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

// Outer loop

for (int i = 1; i <= 2; i++) {

System.out.println("Outer: " + i); // Executes 2 times

// Inner loop

for (int j = 1; j <= 3; j++) {

System.out.println(" Inner: " + j); // Executes 6 times (2 \* 3)

}

}

There is also a "for-each" loop, which is used exclusively to loop through elements in an array:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

break stmt

public class Main {

public static void main(String[] args) {

for (int i = 0; i < 10; i++) {

if (i == 4) {

break;

}

System.out.println(i);

}

}

}

o/p = 0 to 3

continue

public class Main {

public static void main(String[] args) {

for (int i = 0; i < 10; i++) {

if (i == 4) {

continue;

}

System.out.println(i);

}

}

}

The continue statement breaks one iteration (in the loop), if a specified condition occurs,

and continues with the next iteration in the loop.

This example skips the value of 4:

o/p = 0 to 9 except 4

ARRAYS

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars.length);

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

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

System.out.println(cars[i]);

}

int a[] = new int [5];

int n = a.length;

a[0]=10; a[1]=20; a[2]=30; a[3]=40; a[4]=50;

for (int i=0;i<n;i++)

{

System.out.println(a[i]);..................// here i refers index value

}

int a[]= {10,900,30,40,50}; // direct way of array declrn

for (int i:a) //for-each loop is used to fetch elements from an array

{ //this will get each element frm a & will store in i.

System.out.println(i);........// here i refers value and not index

}

METHODS

1.STATIC (NO PARAM NO RETURN)

public class Main {

static void myMethod() {

System.out.println("I just got executed!");

}

public static void main(String[] args) {

myMethod();......................................// no need to create obj of the class to call static mthd

}

}

A method can also be called multiple times:

2. WITH PARAM NO Returns

public class Main {

static void myMethod(String fname, int age) {

System.out.println(fname + " is " + age);

}

public static void main(String[] args) {

myMethod("Liam", 5);

myMethod("Jenny", 8);

myMethod("Anja", 31);

}

}

fname is a parameter, while Liam, Jenny and Anja are arguments.

Note that when you are working with multiple parameters, the method call must have the same number of arguments as

there are parameters,

and the arguments must be passed in the same order.

3. WITH PARAM WITH RETURN

public class Main {

static int myMethod(int x, int y) {

return x + y;

}

public static void main(String[] args) {

int z = myMethod(5, 3);

System.out.println(z);

}

}

4. NO PARAM WITH RETURN ....INVALID

=========================

how to write mthd

accessModifier static(optional) returnType mthdname(params optional inside mthd)

| | | | |

public static int dum (int a, int b)

{

stmts;

return x; //optional

}

Method Overloading

With method overloading, multiple methods can have the same name with different parameters:

Note: Multiple methods can have the same name as long as the number and/or type of parameters are different.

OOPS

- possible to reuse and maintain the codes

-You should extract out the codes that are common for the application,

and place them at a single place and reuse them instead of repeating it.

Like

Class is a logical template to create objects that share common properties and methods,

and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

Class is a blueprint which defines some properties and behaviors. An object is an instance of a class which has those properties and behaviours attached. A class is not allocated memory when it is defined.

An object is allocated memory when it is created. Class is a logical entity whereas objects are physical entities.

Create obj

public class Main {

int x = 5;

public static void main(String[] args) {

Main myObj = new Main();

System.out.println(myObj.x);

}

}

You can also create an object of a class and access it in another class.

This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name.

In this example, we have created two files in the same directory/folder:

Main.java

Second.java

Main.java >>>>

public class Main {

int x = 5;

}

Second.java >>>>

class Second {

public static void main(String[] args) {

Main myObj = new Main();

System.out.println(myObj.x);

}

}

public class Main {

public void myMethod() {

System.out.println("Hello World!");

}

public static void main(String[] args) {

Main b = new Main();

b.myMethod();

}

}

================

class attributes are variables within a class:

Example

Create a class called "Main" with two attributes: x and y:

public class Main {

int x = 5;

int y = 3;

}

You can access attributes by creating an object of the class, and by using the dot syntax (.):

The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...).

The final keyword is called a "modifier".

==========================================================================================

Constructor

Same name as class name

Used to initialize the values of class attributes

The constructor is called when an object of a class is created.

Main myObj = new Main(); // Create an object of class Main (This will call the constructor)

It cannot have a return type (like void).

A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

Default constructor (NON PARAM constructor)

Parameterized constructor

// Create a Main class

public class Main {

int x; // Create a class attribute

// Create a class constructor for the Main class

public Main() {

x = 5; // Set the initial value for the class attribute x

}

public static void main(String[] args) {

Main myObj = new Main(); // Create an object of class Main (This will call the constructor)

System.out.println(myObj.x); // Print the value of x.............// here obj is used to call the variable x , bcauz that varb is non-static

}

}

// Outputs 5

=========== used static varb . so myObj is not used while calling it in syso.=================

// Create a Main class

public class Main {

static int x;

// Create a class constructor for the Main class

public Main() {

x = 5;

}

public static void main(String[] args) {

Main myObj = new Main();

System.out.println(x);

}

}

1. To call non static stuffs (methods / variables) into static method, we hv to call them using object.

public class Main {

int z = 10;

public int add()

{

int x = int a + int b;

return x;

}

public static void main(String[] args) {

Main myObj = new Main();

myObj.add();

System.out.println(myObj.z);

}

}

1. To call static stuffs (methods / variables) into static method, we don’t need to create obj, we can call them directly.

public class Main {

static int z = 10;

public static int add()

{

int x = int a + int b;

return x;

}

public static void main(String[] args) {

add();

System.out.println(z);

}

}

1. Non static mthds can access every thin (static / non static stuffs) directly without creating any object.

public class Main {

int a = 5; int b = 10;

public static int add()

{

int x = a+b;

return x;

}

public int sub()

{

int y = a-b;

return y;

}

public void calculator()

{

add();

sub();

Syso( “add” = x , “sub” = y);

}

public static void main(String[] args) {

Main myObj = new Main();

myObj.calculator();

}

