Java Data types, Keywords, Identifiers, Variables, Operators

First Java Program

```
#include<stdio.h>
int main()
{
      printf(“Hello, world”);
      return 0;
}
```

C Program

```
// this is first java program
public class Hello {
    public static void main(String[] args) {
    System.out.println("Hello world");
 }}
```

$\longrightarrow$ Single line comment

$\longrightarrow$ Defining the class

$\longrightarrow$ Defining the main method

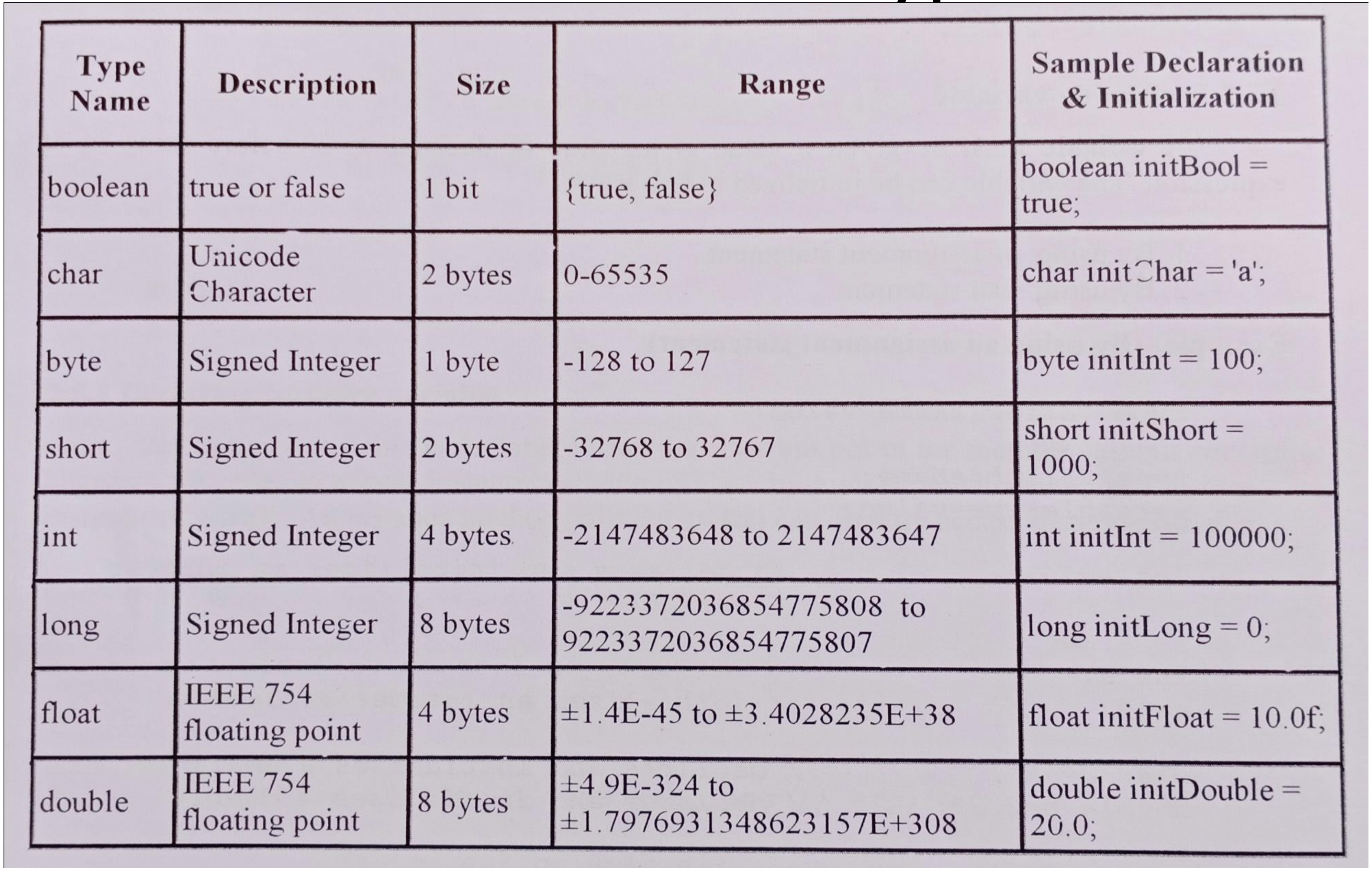
Data Types in Java

- Data types specify the different sizes and values that can be stored in the variable.

There are two types of data types in Java:

- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** The non-primitive data types include classes, interfaces and Arrays.

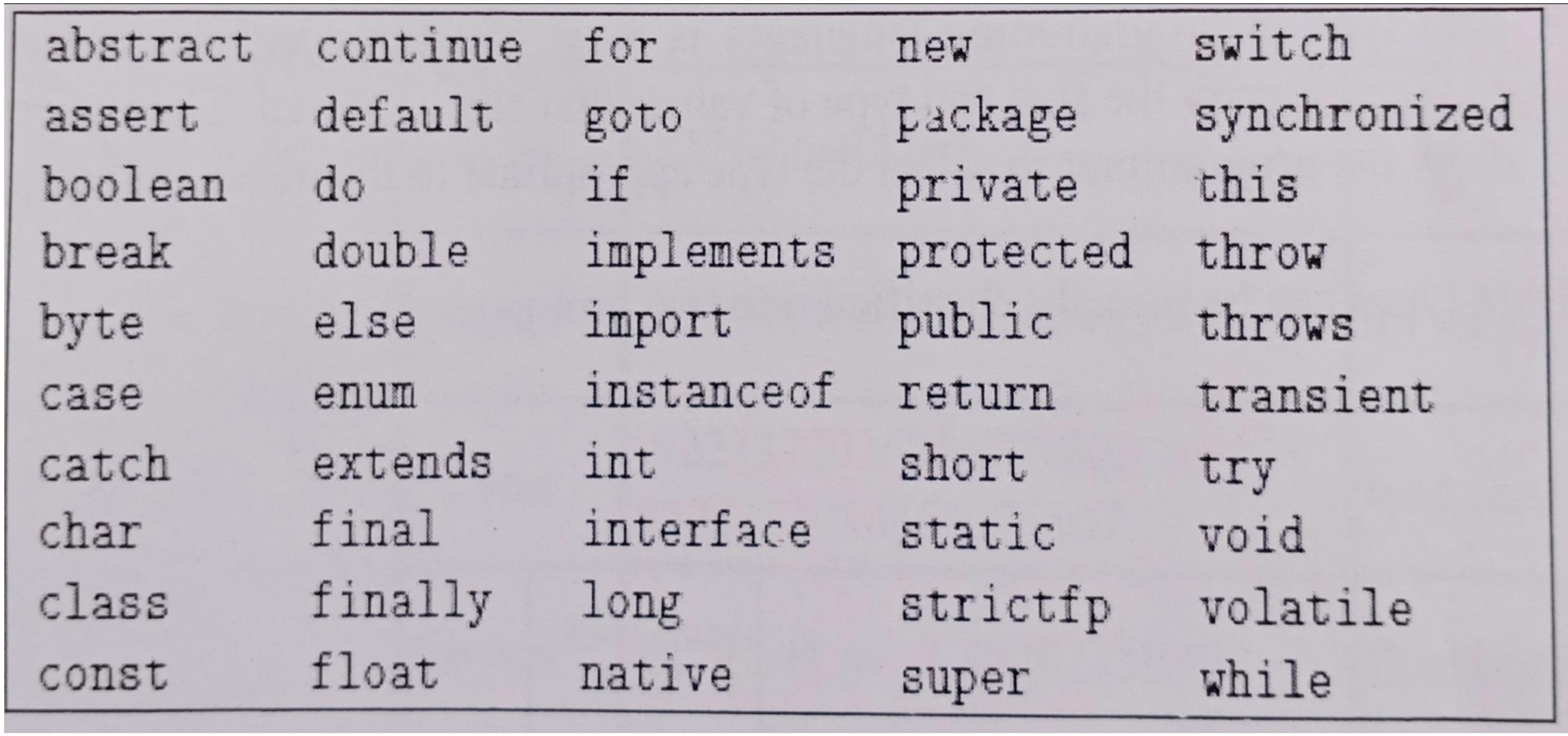
Primitive Data types



Type Name	Description	Size	Range	Sample Declaration & Initialization
boolean	true or false	1 bit	{true, false}	boolean initBool = true;
char	Unicode Character	2 bytes	0-65535	char initChar = 'a';
byte	Signed Integer	1 byte	-128 to 127	byte initInt = 100;
short	Signed Integer	2 bytes	-32768 to 32767	short initShort = 1000;
int	Signed Integer	4 bytes	-2147483648 to 2147483647	int initInt = 100000;
long	Signed Integer	8 bytes	-9223372036854775808 to 9223372036854775807	long initLong = 0;
float	IEEE 754 floating point	4 bytes	±1.4E-45 to ±3.4028235E+38	float initFloat = 10.0f;
double	IEEE 754 floating point	8 bytes	±4.9E-324 to ±1.7976931348623157E+308	double initDouble = 20.0;

https://www.codesdope.com/course/java-data-types/

Java Keywords



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

Identifiers

- **Rules for defining Java Identifiers**
- The only allowed characters for identifiers are all alphanumeric characters([**A-Z**],[**a-z**],[**0-9**]), ‘**$**‘(dollar sign) and ‘**_**‘ (underscore).

- Java identifiers are **case-sensitive**.

- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.

- **Reserved Words** can’t be used as an identifier.

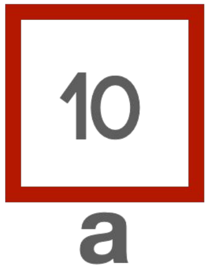
- These rules are also valid for other languages like C,C++.

Valid and Invalid identifiers

Valid identifiers	**Invalid identifiers**
MyClass	My Class
$amount	numberOf*s
totalGrades	final
TotalGrades	1Way
TAX_RATE	@abc
one	

Variables

- What are Variables?
- A variable is used to store some value. You can think of a variable as a storage which has a name and stores some value.



Variable a storing a value 10

- Declaring variables

type variable_name;

e.g. int x;

String name;

double area;

Initializing Variables

```
int a=10;
 or
int a;
a=10;

char c=‘x’;
Double b=2.33;
```

```
class Test {
        public static void main(String[] args) {
int n;
                n=5;
                System.out.println(n);
 }}
```

https://www.codesdope.com/course/java-variables/

print() and println()

```
System.out.print("Hello World! ");
System.out.print("I will print on the same line.");
```

```
System.out.println("Hello World!");
System.out.println("I will print a new line.");
```

Printf() in Java

The `printf()` method outputs a formatted string.
Data from the additional arguments is formatted and written into placeholders
in the formatted string, which are marked by a `%` symbol. The way in which arguments are formatted depends on the sequence of characters that follows the `%` symbol.

```
System.out.printf("%2$,3.2f %1$s", "meters", 1260.5052);
```

This is how each part of the placeholder `%2$,3.2f` works:

- `2$` indicates that the value of the second argument is used
- `,` indicates that digits should be grouped (usually by thousands)
- `3` indicates that the representation of the data should be at least 3 characters long
- `.2` indicates that there should be two digits after the decimal point
- `f` indicates that the data is being represented as a floating point number

O/P- 1,260.50 meters

User input

- We take input with the help of the **Scanner** class.
- Java has a number of **predefined classes** which we can use in our java code.
- Predefined classes are organized in the form of **packages.**
- This Scanner class is found in the **java.util** package. So to use the Scanner class, we first need to include the **java.util** package in our program.
- Include a package in a program with the help of the **import** keyword.
- A package has many predefined classes.
- We can either import the **java.util.Scanner** class or the entire **java.util** package.
- import java.util.Scanner; // This will import just the Scanner class
- import java.util.*; // This will import the entire java.util package

Accept data from user

```
import java.util.*;
class UserInput {
    public static void main(String[] args)
    {
      Scanner s1 = new Scanner(System.in);
      System.out.println("Enter an integer");
      int a;
      a = s1.nextInt();
      System.out.println("The entered integer is" + a);
    }
}
```

Methods used for taking input of different data types.

Method	Inputs
nextInt()	Integer
nextFloat()	Float
nextDouble()	Double
nextLong()	Long
nextShort()	Short
next()	Single Word
nextLine()	Line of Strings
nextBoolean()	Boolean

https://www.codesdope.com/course/java-input/

Using Command line Arguments

```
public class SumOfNumbers4
 { public static void main(String args[])
{ int x = Integer.parseInt(args[0]); //first arguments
int y = Integer.parseInt(args[1]); //second arguments
int sum = x + y;
System.out.println("The sum of x and y is: " +sum); } }
```

Operators

- Java provides a rich set of built-in operators which can be categorized as follows.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Assignment Operators
  - Increment and Decrement Operators

Arithmetic operators

Operator	Description	Example
+	Adds operands	a + b = 5
-	Subtracts right operand from left operand	a - b = 1
*	Multiplies both operands	a * b = 6
/	Quotient of division of left operand by right operand	a / b = 1.5
%	Remainder of division of left operand by right operand	a % b = 1

Command line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.

- Javac filename.java
- Java filename arg1 arg2…….
- 12 56 abc 3.9

Java Relational Operators

Operator	Description	Example
==	Equal to	(a == b) is false
!=	Not equal to	(a != b) is true
>	Greater than	(a > b) is true
<	Less than	(a < b) is false
>=	Greater than or equal to	(a >= b) is true
<=	Less than or equal to	(a <= b) is false

Java Logical Operators

Operator	Description	Example
&&	Logical AND. It returns true if both the operands are true. Otherwise, it returns false.	(a && b) is false
||	Logical OR. It returns true if either one or both the operands are true. Otherwise, it returns false.	(a || b) is true
!	Logical NOT. It is used to reverse a condition. So, if a condition is true, ! makes it false and vice versa.	!a is false, !b is true

Java Assignment Operators

Operator	Description	Example
=	Assigns value of right operand to left operand	C = A + B
+=	Adds the value of right operand to left operand and assigns the final value to the left operand	C += A is same as C = C + A
-=	Subtracts the value of right operand from left operand and assigns the final value to left operand	C -= A is same as C = C - A
*=	Multiplies the value of right operand to left operand and assigns the final value to left operand	C *= A is same as C = C * A
/=	Divides the value of left operand by right operand and assigns the quotient to left operand	C /= A is same as C = C / A
%=	Divides the value of left operand by right operand and assigns the remainder to left operand	C %= A is same as C = C % A

Java Increment and Decrement Operators

- ++ and -- are called increment and decrement operators respectively.
- **++** adds 1 to the operand whereas **--** subtracts 1 from the operand.
- **a++** increases the value of a variable a by 1 and **a--** decreases the value of a by 1.
- Similarly, **++a** increases the value of a by 1 and **--a** decreases the value of a by 1.
- In **a++** and **a--**, ++ and -- are used as **postfix** whereas in **++a** and **--a**, ++ and -- are used as **prefix**.
- For example, suppose the value of a is 5, then *a++* and *++a* will change the value of a to 6. Similarly *a*-- and --*a* will change the value of *a* to 4.

Difference between Prefix and Postfix operators

- While both a++ and ++a increases the value of a, the only difference between these is that a++ returns the value of a before the value of a is incremented and ++a first increases the value of a by 1 and then returns the incremented value of a.

```
m=5;
y=++m;
Value of m & y is 6

m=5;
y=m++;
Value of m = 6 & y is 5
```

- Similarly, a-- first returns the value of a and then decreases its value by 1 and --a first decreases the value of a by 1 and then returns the decreased value.

https://www.codesdope.com/course/java-operators/

Java Control Statements

- Provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.
- Java provides three types of control flow statements.
- Decision Making statements
  - if statements
  - switch statement
- Loop statements
  - do while loop
  - while loop
  - for loop
- Jump statements
  - break statement
  - continue statement

Syntax for decision making statements

If statement:

```
if(condition) {
statement 1; //executes when condition is true
}
```

If- else statement:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

Experiment no 2

- Accept 3 numbers from user. Compare them and declare the largest number using if-else statement.
- PL: Accept marks from user, if marks greater than 40 declare student as “pass” else “Fail”.

if-else-if ladder

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

Program to check rating

- Accept rating of the product from the user.
- Compare rating and print remark
  - If rating is
    - <5 then print “bad rating”
    - <8 then print “average rating”
    - else print “good rating”

HW-Accept grade and print the remark

```
Scanner sc = new Scanner(System.in);
char grade = sc.next().charAt(0);
```

- Grade==‘A’- Outstanding
- Grade== ‘B’- Excellent
- Grade== ‘C’- Good
- Grade== ‘D’- can do better
- Grade== ‘E’- Pass
- Grade== ‘F’- Fail
- For any other grade(apart from A-F)- Invalid grade

```
• class Test {
    public static void main(String[] args)
      {
   Scanner sc = new Scanner(System.in);
   System.out.println(“Enter Grade:”);
   char grade = sc.next().charAt(0);
           if (grade == 'A') {
              System.out.println("Excellent !");
           } else if (grade == 'B')  {
             System.out.println("Outstanding !");
           }   else if (grade == 'C') {
              System.out.println("Good !");
           }   else if (grade == 'D') {
              System.out.println("Can do better");
           }   else if (grade == 'E') {
              System.out.println("Just passed");
             }  else if (grade == 'F') {
                System.out.println("You failed");
            }    else  {
                 System.out.println("Invalid grade");
            }
   }}
```

Nested if statement

- Nested if statements means an if statement inside an if statement.
  Syntax:

```
if (condition1)
{
  // Executes when condition1 is true
 if (condition2)
    {
       // Executes when condition2 is true
         }
 }
```

- Define 3 variables (x,y,z) and assign integer values
- Check whether x is greater or not
  - If (x>y)
    - If(x>z) then
      - Print-X is a greater no
    - Else print x is not a greater no
  - Else print-x is not a greater no

While loop

The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

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

Program to print 1-10 nos in reverse order

HW- print odd numbers between 1-20

- WAP to display table of 15 using while loop
- PL: display sum of first 10 even numbers using do while loop.

- HW: print odd nos between 1-20 using while loop

For Loop

- The Java *for loop* is used to iterate a part of the program several times. If the number of iterations is **fixed**, it is recommended to use for loop.

  ```
  for(initialization; condition; increment/decrement
  )
    {
  //statement or code to be executed
  }
  ```

WAP to print following patters

```
*
**
***
****
*****
```

<u>Accept no from user and check whether it is prime nos or not.</u>

<u>0 1 2 3 4 5 6 7 8 9</u>

Do-while loop

- The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.
- **Syntax**

```
do{
 //code to be executed / loop body
 //update statement
   }while (condition);
```

Program to print 1-10 nos

```
public class DoWhileExample {
public static void main(String[] args) {
   int i=1;
   do{
      System.out.println(i);
   i++;
   }while(i<=10);
}
}
```

HW: WAP to print Fibonacci series for first 20 numbers

Fibonacci series is: 0 1 1 2 3 5 8 13 21…

```
1.   Initialize first 2 nos (a,b)
2.   Initialize x=0,i=0 and n=18
3.   Print first 2 nos
4.   X= a+b;
5.   Print x
6.   a=b and b=x
7.   Increment I
8.   Check (i<=18) if yes then goto step 4
9.   Stop.
```

Switch Statement

- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.
- The switch statement tests the equality of a variable against multiple values.
  - There can be ***one or N number of case values*** for a switch expression.
  - The **case value must be of switch expression type only**. The case value must be *literal or constant*. It doesn't allow variables.
  - **The case values must be *unique*.** In case of duplicate value, it renders compile-time error.
  - Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
  - The case value can have a *default label* which is optional.

Syntax: Switch statement

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
  code to be executed if all cases are not matched;
}
```

-
  ```
  class Test {
    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter Grade:");
    char grade = sc.next().charAt(0);
     switch (grade)
        {
          case 'A':
                 System.out.println("Excellent !");
                 break;
          case 'B':
                  System.out.println("Outstanding !");
                  break;
          case 'C':
                  System.out.println("Good !");
                  break;
          case 'D':
                  System.out.println("Can do better");
                  break;
          case 'E':
                  System.out.println("Just passed");
                  break;
          case 'F':
                 System.out.println("You failed");
                   break;
          default:
                 System.out.println("Invalid grade");
        } } }
  ```

- Write a program to display basic calculator using Switch Statement.

Jump and Continue Statement

- To jump out of a loop or jump to the starting condition of a loop whenever of break and continue statements are used.
- **Java break Statement**

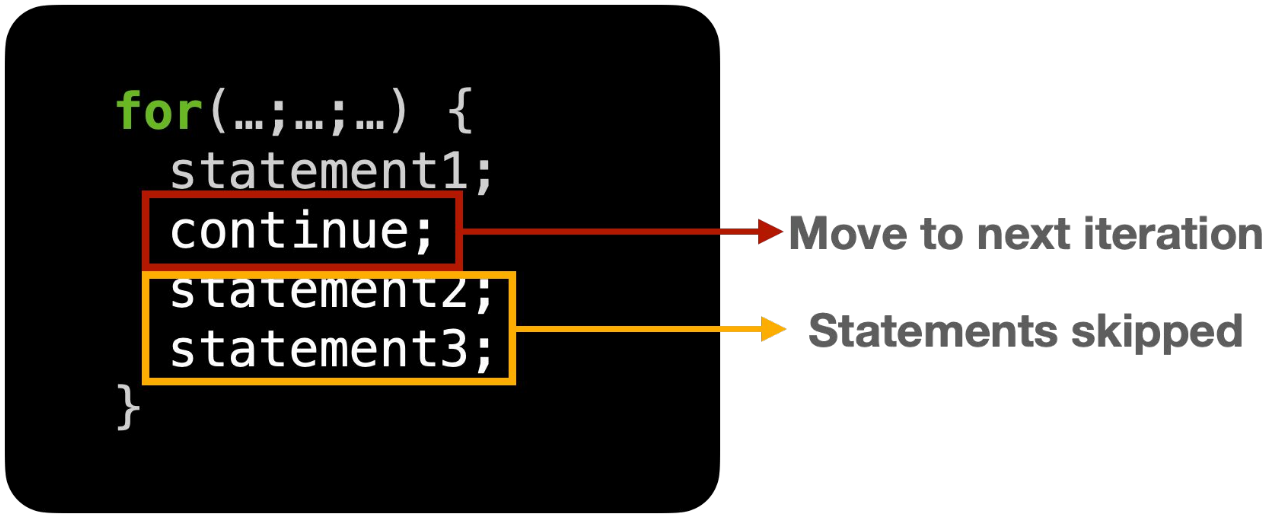
  break is used to break or terminate a loop whenever we want.
- Just type break; after the statement after which you want to break the loop.
- Syntax

  break;

```
class BreakExample {
public static void main(String[] args) {
    for (int n = 1; n <= 5; n++) {
      System.out.println("*");
      if (n == 2) {
         break;
  } } } }
```

Continue statement

- While the break statement **terminates** the loop, the continue statement skips the remaining statements in the loop and starts the next iteration.
- In short, it ends the execution of the current iteration without executing the rest of the statements in the loop. After that, the f**low proceeds to the next iteration as usual**.



```
for(…;…;…) {
  statement1;
  continue;
  statement2;
  statement3;
}
```

- Java continue Syntax

```
continue;

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

Java -Array

- An array is a collection of similar types of data. For example, an array of *int* is a collection of integers, an array of *double* is a collection of doubles, etc.

Declaration of an Array in Java

type[] arrayName = **new** type[array_size];

e.g. int[] arr = new int[6];

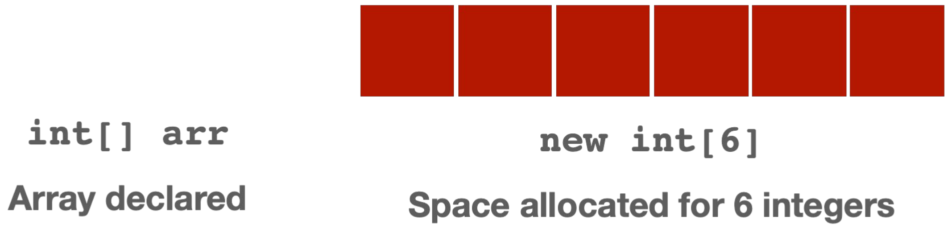
Or

type array_name[];

array_name=new type[array_size];

e.g. int arr[];

arr=new int[6];



Arrays

- Similarly, an array named words to store 5 English words can be declared as:

```
String[] words = new String[5];
```

- To Declare and initialize an array at the same time:

```
int[] arr = new int[]{8, 10, 2, 7, 4, 56};
```

Or

```
int[] arr = {8, 10, 2, 7, 4, 56};
```

element	8	10	2	7	4	56
index	0	1	2	3	4	5

An element of an array can be accessed using the following syntax.

arrayName[index]

Accessing Elements of an Array in Java

```
class Test {
      public static void main(String[] args)
      {
        int[] myarray = { 10, 20, 30 };
        System.out.println("First element: " + myarray[0]);
System.out.println("Second element: " + myarray[1]);
        System.out.println("Third element: " + myarray[2]);
      }
}
```

WAP to accept elements of array and print it.

- Write a program in java that reads an array of N elements and then calculate the sum of those N elements and display it to the standard output or screen.
- PL : Find maximum and minimum elements from the array.

2D Arrays

- A 2-dimensional array is generally known as a matrix. It stores data in the form of rows and columns as shown below.

	Column 0	Column 1	Column 2	Column 3
Row 0				
Row 1				

- Declaration of a 2D Array

type[][] arrayName = **new** type[row_size][column_size];

- E.g. int[][] a = new int[2][4];
- **int** a[][] = **new int**[][]{ {1, 2, 3}, {4, 5, 6} };

- This is a 2-dimensional array of type int having 2 rows and 4 columns as shown below.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]

2D Arrays

// Program to print elements of 2D array

```
class TwoDimensionalArray
{
public static void main(String args[])
{
int[][] a={{10,20},{30,40},{50,60}};//declaration and initialization
System.out.println("Two dimensional array elements are");
for (int i = 0; i < 3; i++)
{
        for (int j = 0; j < 2; j++)
  {
          System.out.println(a[i][j]);
        }
}
}
}
```

- Write a program in java such that it accepts two matrices using 2D array of m x n size and then calculate the multiplication of elements of matrices. Display it to the standard output or screen.
- PL: Write a program to implement Matrix addition and subtraction.

To accept elements from user

```
for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
        {
          System.a[i][j]=s.nextInt();
        }
  }
```

1	2	3
2	3	4
2	3	4

2	2	3
1	2	2
3	2	4

- C[i][j]=a[i][j]+b[i][j]

$$\begin{bmatrix} a_{11}, & a_{12} \\ a_{21}, & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11}, & b_{12} \\ b_{21}, & b_{22} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} \times b_{11} + a_{12} \times b_{21} , & a_{11} \times b_{12} + a_{12} \times b_{22} \\ a_{21} \times b_{11} + a_{22} \times b_{21} , & a_{21} \times b_{12} + a_{22} \times b_{22} \end{bmatrix}$$

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \times \begin{array}{|c|c|}\hline 5 & 6 \\ \hline 7 & 8 \\ \hline \end{array}$$

$$\begin{bmatrix} 1 \times 5 + 2 \times 7 , & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 , & 3 \times 6 + 4 \times 8 \end{bmatrix} = \begin{bmatrix} 5 + 14, & 6 + 16 \\ 15 + 28, & 18 + 32 \end{bmatrix}$$

$$= \begin{array}{|c|c|}\hline 19 & 22 \\ \hline 43 & 50 \\ \hline \end{array}$$

Matrix Multiplication

```
for(i=0;i<2;i++)
  {
      for(j=0;j<2;j++)
      {
              c[i][j]=0;
              for(k=0;k<2;k++)
              {
                   c[i][j]= c[i][j]+ (a[i][k] *b[k][j]);

              }
              System.out.print(c[i][j]+"\t");

         }
             System.out.println();

  }
```

$$\begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} a00*b00+a01*a10 & a00*b01+a01*a11 \\ a10*b00+a11*a10 & a10*b01+a11*a11 \end{bmatrix}$$