Java Cheatsheet

Haris Ali Khan • January 4, 2023 • 15 min read

Basics

Basic syntax and functions from the Java programming language.

Boilerplate

```
class HelloWorld
{
        public static void main(String args[])
        {
              System.out.println("Hello World");
        }
}
```

Showing Output

It will print something to the output console.

```
class HelloWorld
{
        public static void main(String args[])
        {
              System.out.println("Hello World");
        }
}
```

Taking Input

It will take string input from the user

```
import java.util.Scanner;
class HelloWorld
{
        public static void main(String args[])
```

```
        {
              Scannner sc=new Scanner(System.in);
              String name=sc.nextLine();
              System.out.println(name);
        }
}
```

It will take integer input from the user

```
import java.util.Scanner;
class HelloWorld
{
        public static void main(String args[])
        {
              Scannner sc=new Scanner(System.in);
              int x=sc.nextInt();
              System.out.println(x);
        }
}
```

It will take float input from the user

```
import java.util.Scanner;
class HelloWorld
{
        public static void main(String args[])
        {
              Scannner sc=new Scanner(System.in);
              int x=sc.nextFloat();
              System.out.println(x);
        }
}
```

It will take double input from the user

```
import java.util.Scanner;
class HelloWorld
{
        public static void main(String args[])
        {
              Scannner sc=new Scanner(System.in);
              double x=sc.nextDouble();
              System.out.println(x);
        }
}
```

Primitive Type Variables

The eight primitives defined in Java are int, byte, short, long, float, double, boolean, and char those aren't considered objects and represent raw values.

byte

byte is a primitive data type it only takes up 8 bits of memory.

```
class HelloWorld
{
       public static void main(String args[])
      {
            byte age=18;
            System.out.println(age);
      }
}
```

long

long is another primitive data type related to integers. long takes up 64 bits of memory.

```
class HelloWorld
{
       public static void main(String args[])
      {
            long var=900.0;
            System.out.println(age);
      }
}
```

float

We represent basic fractional numbers in Java using the float type. This is a single-precision decimal number. Which means if we get past six decimal points, this number becomes less precise and more of an estimate.

```
class HelloWorld
{
       public static void main(String args[])
      {
            float price=100.05;
            System.out.println(price);
      }
```

```
}
```

char

Char is a 16-bit integer representing a Unicode-encoded character.

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

int

int holds a wide range of non-fractional number values.

```
class HelloWorld
{
   public static void main(String args[])
   {
         int var1=256;
         System.out.println(var1);
    }
}
```

short

If we want to save memory and byte is too small, we can use short.

```
class HelloWorld
{
   public static void main(String args[])
   {
         short var2=5666;
         System.out.println(var2);
   }
}
```

Comments

A comment is the code that is not executed by the compiler, and the

programmer uses it to keep track of the code.

Single line comment

```
// It's a single line comment
```

Multi-line comment

```
/* It's a
multi-line
comment
*/
```

Constants

Constants are like a variable, except that their value never changes during program execution.

```
public class Declaration {

  final double PI = 3.14;

  public static void main(String[] args) {
    System.out.println("Value of PI:   " + PI);
  }
}
```

Arithmetic Expressions

These are the collection of literals and arithmetic operators.

Addition

It can be used to add two numbers

```
public class HelloWorld
{
    public static void main(String args[])
    {
          int x=10+3;
          System.out.println(x);
    }
}
```

Subtraction

It can be used to subtract two numbers

```
public class HelloWorld
{
    public static void main(String args[])
    {
          int x=10-3;
          System.out.println(x);
    }
}
```

Multiplication

It can be used to multiply add two numbers

```
public class HelloWorld
{
    public static void main(String args[])
    {
          int x=10*3;
          System.out.println(x);
    }
}
```

Division

It can be used to divide two numbers

```
public class HelloWorld
{
    public static void main(String args[])
    {
          int x=10/3;
          System.out.println(x);
    }
}
```

Modulo Remainder

It returns the remainder of the two numbers after division

```
public class HelloWorld
{
    public static void main(String args[])
    {
          int x=10%3;
          System.out.println(x);
    }
}
```

Augmented Operators

Addition assignment

```
public class HelloWorld
{
    public static void main(String args[])
    {
          var=1;
          var+=10;
          System.out.println(var);
    }
}
```

Subtraction assignment

```
public class HelloWorld
{
    public static void main(String args[])
    {
          var=1;
          var-=10;
          System.out.println(var);
    }
}
```

Multiplication assignment

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

```
            var=1;
            var*=10;
            System.out.println(var);
    }
}
```

Division assignment

```
public class HelloWorld
{
    public static void main(String args[])
    {
            var=1;
            var/=10;
            System.out.println(var);
    }
}
```

Modulus assignment

```
public class HelloWorld
{
    public static void main(String args[])
    {
            var=1;
            var%=10;
            System.out.println(var);
    }
}
```

Escape Sequences

It is a sequence of characters starting with a backslash, and it doesn't represent itself when used inside string literal.

Tab

It gives a tab space

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

```
        System.out.print("\t");
    }
}
```

Backslash

It adds a backslash

```
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.print("\\");
    }
}
```

Single quote

It adds a single quotation mark

```
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.print("\'");
    }
}
```

Question mark

It adds a question mark

```
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.print("\?");
    }
}
```

Carriage return

Inserts a carriage return in the text at this point.

```
public class HelloWorld
{
   public static void main(String args[])
   {
        System.out.print("\r");
   }
}
```

Double quote

It adds a double quotation mark

```
public class HelloWorld
{
   public static void main(String args[])
   {
        System.out.print("\"");
   }
}
```

Type Casting

Type Casting is a process of converting one data type into another

Widening Type Casting

It means converting a lower data type into a higher

```
class HelloWorld
{
   public static void main(String args[])
   {
        int x = 45;
        double var_name = x;
        System.out.println(var_name);
   }

}
```

Narrowing Type Casting

It means converting a higher data type into a lower

```
class HelloWorld
```

```
{
   public static void main(String args[])
   {
        double x = 40005;
        int var_name = x;
        System.out.println(var_name);
   }

}
```

Decision Control Statements

Conditional statements are used to perform operations based on some condition.

if Statement

```
if (condition) {
// block of code to be executed if the condition is true
}
```

if-else Statement

```
if (condition) {
// If condition is True then this block will get executed
} else {
// If condition is False then this block will get executed
}
```

if else-if Statement

```
if (condition1) {
// Codes
}
else if(condition2) {
// Codes
}
else if (condition3) {
// Codes
}
else {
// Codes
```

```
}
```

Ternary Operator

It is shorthand of an if-else statement.

Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Example

```
public class TernaryOperatorExample
{
  public static void main(String args[])
  {
    int x, y;
    x = 20;
    y = (x == 1) ? 61: 90;
    System.out.println("Value of y is: " +  y);
    y = (x == 20) ? 61: 90;
    System.out.println("Value of y is: " + y);
  }
}
```

Switch Statements

It allows a variable to be tested for equality against a list of values (cases).

```
class SwitchExample
{
  public static void main(String args[])
  {
  int day = 4;
  switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
```

```
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
```

Iterative Statements

Iterative statements facilitate programmers to execute any block of code lines repeatedly and can be controlled as per conditions added by the coder.

while Loop

It iterates the block of code as long as a specified condition is True

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

for Loop

for loop is used to run a block of code several times

```
class HelloWorld
{
   public static void main(String args[])
   {
        int i;
        for(i=1;i<100;i++)
         {
              System.out.println(i);
         }
   }

}
```

for-each Loop

```
public class HelloWorld
{
   public static void main(String args[])
   {
     int[] arr = {2,4,5,7,8,0,3,5}
     for (int i : arr) {
     System.out.println(i);
   }
}
```

do-while Loop

It is an exit controlled loop. It is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the condition is False

```
public class HelloWorld
{
   public static void main(String args[])
   {
        int i=1;
        do
        {
             System.out.println(i);
             i++;
        }while(i<=100);
   }
}
```

Break statement

break keyword inside the loop is used to terminate the loop

```
class HelloWorld
{
   public static void main(String args[])
   {
        int i;
        for(i=1;i<100;i++)
         {
              System.out.println(i);
```

```
                if(i==50)
                break;
            }
    }

}
```

Continue statement

continue keyword skips the rest of the current iteration of the loop and returns to the starting point of the loop

```
class HelloWorld
{
   public static void main(String args[])
   {
        int i;
        for(i=1;i<100;i++)
         {
              System.out.println(i);
              if(i==50)
              continue;
         }
   }

}
```

Arrays

Arrays are used to store multiple values in a single variable

Declaring an array

Declaration of an array

```
public class HelloWorld
{
  public static void main(String args[])
  {
     String [] var_name;
  }
}
```

Defining an array

Defining an array

```
public class HelloWorld
{
  public static void main(String args[])
  {
     String [] var_name={"harry","rohan","aakash"}
  }
}
```

Accessing an array

Accessing the elements of an array

```
public class HelloWorld
{
  public static void main(String args[])
  {
    String[] var_name = {''Harry", "Rohan", "Aakash"};
    System.out.println(var_name[index]);
  }
}
```

Changing an element

Changing any element in an array

```
public class HelloWorld
{
  public static void main(String args[])
  {
    String[] var_name = {''Harry", "Rohan", "Aakash"};
    var_name[2]="Shubham";
  }
}
```

Array length

It gives the length of the array

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

```
    System.out.println(var_name.length);
  }
}
```

Loop through an array

It allows us to iterate through each array element

```
public class HelloWorld
{
  public static void main(String args[])
  {
    String[] var_name = {''Harry", "Rohan", "Aakash"};
    for (int i = 0; i < var_name.length; i++) {
     System.out.println(var_name[i]);
     }
  }
}
```

Multi-dimensional Arrays

Arrays can be 1-D, 2-D or multi-dimensional.

```
// Creating a 2x3 array (two rows, three columns)
int[2][3] matrix = new int[2][3];
matrix[0][0] = 10;
// Shortcut
int[2][3] matrix = {
{ 1, 2, 3 },
{ 4, 5, 6 }
};
```

Methods

Methods are used to divide an extensive program into smaller pieces. It can be called multiple times to provide reusability to the program.

Declaration

Declaration of a method

```
returnType methodName(parameters) {
//statements
}
```

Calling a method

Calling a method

```
methodName(arguments);
```

Example

```
public static void findEvenOdd(int num)
{
  //method body
  if(num%2==0)
    System.out.println(num+" is even");
  else
    System.out.println(num+" is odd");
}
import java.util.Scanner;
public class EvenOdd
{
  public static void main (String args[])
  {
    //creating Scanner class object
    Scanner scan=new Scanner(System.in);
    System.out.print("Enter the number: ");
    //reading value from the user
    int num=scan.nextInt();
    //method calling
    findEvenOdd(num);
}
```

Method Overloading

Method overloading means having multiple methods with the same name, but different parameters.

```
class Calculate
{
  void sum (int x, int y)
  {
    System.out.println("Sum is: "+(a+b)) ;
  }
  void sum (float x, float y)
  {
    System.out.println("Sum is: "+(a+b));
```

```
  }
public static void main (String[] args)
{
   Calculate calc = new Calculate();
   calc.sum (5,4); //sum(int x, int y) is method is called.
   calc.sum (1.2f, 5.6f); //sum(float x, float y) is called.
}
}
```

Recursion

Recursion is when a function calls a copy of itself to work on a minor problem. And the function that calls itself is known as the Recursive function.

```
void recurse()
{
recurse();
}
```

Strings

It is a collection of characters surrounded by double quotes.

Creating String Variable

```
String var_name = "Hello World";
```

String Length

Returns the length of the string

```
public class str
{
  public static void main(String args[])
  {
      String var_name = "Harry";
      System.out.println("The length of the string is: " + var_name.
  }
}
```

String Methods toUpperCase()

Convert the string into uppercase

```
public class str
{
  public static void main(String args[])
  {
      String var_name = "Harry";
      System.out.println(var_name.toUpperCase());
  }
}
```

toLowerCase()

Convert the string into lowercase

```
public class str
{
  public static void main(String args[])
  {
      String var_name = "Harry";
      System.out.println(var_name.toLowerCase());
  }
}
```

indexOf()

Returns the index of specified character from the string

```
public class str
{
  public static void main(String args[])
  {
      String var_name = "Harry";
      System.out.println(var_name.indexOf("a"));
  }
}
```

concat()

Used to concatenate two strings

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

```
    String var1 = "Harry";
    String var2 = "Bhai";
    System.out.println(var1.concat(var2));
  }
}
```

Math Class

Math class allows you to perform mathematical operations.

Methods max() method

It is used to find the greater number among the two

```
public class Demo
{
  public static void main(String[] args)
  {
    // using the max() method of Math class
    System.out.print("The maximum number is: " + Math.max(9,7));
  }
}
```

min() method

It is used to find the smaller number among the two

```
public class Demo
{
  public static void main(String[] args)
  {
    // using the min() method of Math class
    System.out.print("The maximum number is: " + Math.min(9,7));
  }
}
```

sqrt() method

It returns the square root of the supplied value

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

```
    // using the sqrt method of Math class
    System.out.print("The maximum number is: " + Math.sqrt(144));
  }
}
```

random() method

It is used to generate random numbers

```
Math.random(); //It will produce random number b/w 0.0 and 1.0
```

```
public class Demo
{
  public static void main(String[] args)
  {
    // using the random() method of Math class
    int random_num = (int)(Math.random() * 101); //Random num b/w 0
    System.out.println(random_num);
  }
}
```

Object-Oriented Programming

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

```
class ClassName {
// Fields
// Methods
// Constructors
// Blocks
}
```

object of class

An object is an instance of a Class.

```
className object = new className();
```

Encapsulation

Encapsulation is a mechanism of wrapping the data and code acting on the data together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class.

```
public class Person
 {
   private String name; // using private access modifier

   // Getter
   public String getName()
   {
    return name;
   }

   // Setter
   public void setName(String newName)
   {
     this.name = newName;
   }
}
```

Inheritance

Inheritance can be defined as the process where one class acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.

```
class Subclass-name extends Superclass-name
{
//methods and fields
}
```

Example

```
class Employee
{
  float salary=40000;
}
class Programmer extends Employee
```

```
{
  int bonus=10000;
  public static void main(String args[])
  {
    Programmer p=new Programmer();
    System.out.println("Programmer salary is:"+p.salary);
    System.out.println("Bonus of Programmer is:"+p.bonus);
  }
}
```

Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

```
// A class with multiple methods with the same name
public class Adder
{
// method 1
  public void add(int a, int b)
  {
  System.out.println(a + b);
  }

  // method 2
  public void add(int a, int b, int c)
  {
  System.out.println(a + b + c);
  }

  // method 3
  public void add(String a, String b)
  {
  System.out.println(a + " + " + b);
  }
}
```

File Operations

File handling refers to reading or writing data from files. Java provides some functions that allow us to manipulate data in the files.
Assume that we have created the file "D:\\Example.txt"

canRead method

Checks whether the file is readable or not

```
import java.io.*;

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

        // Get the file
        File f = new File("D:\\Example.txt");

        // Check if the specified file
        // can be read or not
        if (f.canRead())
            System.out.println("Can be Read");
        else
            System.out.println("Cannot be Read");
    }
}
```

createNewFile method

It creates an empty file

```
import java.io.*;

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

        try {

            // Get the file
            File f = new File("D:\\Example.txt");

            // Create new file
            // if it does not exist
            if (f.createNewFile())
                System.out.println("File created");
            else
                System.out.println("File already exists");
        }
        catch (Exception e) {
            System.err.println(e);
```

canWrite method

Checks whether the file is writable or not

```
import java.io.*;

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

        // Get the file
        File f = new File("D:\\Example.txt");

        // Check if the specified file
        // can be written or not
        if (f.canWrite())
            System.out.println("Can be written");
        else
            System.out.println("Cannot be written");
    }
}
```

exists method

Checks whether the file exists

```
import java.io.*;

// Main class
public class FileOperations {

    public static void main(String args[])
    {
        File f = new File("D:\\Example.txt");

        // Checking if the specified file exists or not
        if (f.exists())

            // Show if the file exists
            System.out.println("Exists");
        else

            // Show if the file does not exists
            System.out.println("Does not Exists");
    }
```

```
}
```

delete method

It deletes a file

```
import java.io.*;

public class FileOperations {
    public static void main(String[] args)
    {
        File file= new File("D:\\Example.txt");

        if (file.delete()) {
            System.out.println("File deleted successfully");
        }
        else {
            System.out.println("Failed to delete the file");
        }
    }
}
```

getName method

It returns the name of the file

```
import java.io.*;

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

        // try-catch block to handle exceptions
        try {

            // Create a file object
            File f = new File("D:\\Example.txt");

            // Get the Name of the given file f
            String Name = f.getName();

            // Display the file Name of the file object
            System.out.println("File Name : " + Name);
        }
        catch (Exception e) {
```

```
            System.err.println(e.getMessage());
        }
```

getAbsolutePath method

It returns the absolute pathname of the file

```
import java.io.*;

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

        // try-catch block to handle exceptions
        try {

            // Create a file object
            File f = new File("Example.txt");

            // Get the absolute path of file f
            String absolute = f.getAbsolutePath();

            // Display the file path of the file object
            // and also the file path of absolute file
            System.out.println("Original  path: " + f.getPath());
            System.out.println("Absolute  path: "+ absolute);
        }
        catch (Exception e) {
```

length Method

It returns the size of the file in bytes

```
import java.io.*;

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

        // Get the file
        File f = new File("D:\\Example.txt");

        // Get the length of the file
        System.out.println("length: " + f.length());
    }
}
```

list Method

It returns an array of the files in the directory

```
import java.io.*;

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

        // try-catch block to handle exceptions
        try {

            // Create a file object
            File f = new File("f:\\Examples");

            // Get all the names of the files present
            // in the given directory
            String[] files = f.list();

            System.out.println("Files are:");

            // Display the names of the files
            for (int i = 0; i < files.length; i++) {
                System.out.println(files[i]);
```

mkdir method

It is used to create a new directory

```
import java.io.*;

public class FileOperations {

    public static void main(String args[])
    {
        // create an abstract pathname (File object)
        File f = new File("D:\\program");

        // check if the directory can be created
        // using the abstract path name
        if (f.mkdir()) {

            // display that the directory is created
            // as the function returned true
            System.out.println("Directory is created");
```

```
        }
        else {
            // display that the directory cannot be created
            // as the function returned false
            System.out.println("Directory cannot be created");
```

close method

It is used to close the file

```
import java.io.File;
import java.io.FileInputStream;

public class FileOperations {

    public static void main(String[] args)
    {

        // Creating file object and specifying path
        File file = new File("file.txt");

        try {
            FileInputStream input= new FileInputStream(file);
            int character;
            // read character by character by default
            // read() function return int between
            // 0 and 255.

            while ((character = input.read()) != -1) {
                System.out.print((char)character);
            }
```

To write something in the file

```
import java.io.FileWriter; // Import the FileWriter class
import java.io.IOException; // Import the IOException class to hand

public class WriteToFile
{
  public static void main(String[] args) {
  try
  {
     FileWriter myWriter = new FileWriter("filename.txt");
     myWriter.write("Laal Phool Neela Phool, Harry Bhaiya Beautiful
     myWriter.close();
     System.out.println("Successfully wrote to the file.");
```

```
    }
    catch (IOException e)
    {
       System.out.println("An error occurred.");
       e.printStackTrace();
    }
}
}
```

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

try-catch block

try statement allow you to define a block of code to be tested for errors. catch block is used to handle the exception.

```
try {
// Statements
}
catch(Exception e) {
// Statements
}
```

Example

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

    try {
      int divideByZero = 5 / 0;
      System.out.println("Rest of code in try block");
    }

    catch (ArithmeticException e) {
      System.out.println("ArithmeticException => " + e.getMessage())
    }
  }
}
```

finally block

finally code is executed whether an exception is handled or not.

```
try {
//Statements
}
catch (ExceptionType1 e1) {
// catch block
}
finally {
// finally block always executes
}
```

Example

```
class Main {
  public static void main(String[] args) {
    try {
      int divideByZero = 5 / 0;
    }

    finally {
      System.out.println("Finally block is always executed");
    }
  }
}
```

Download this Cheatsheet

Comments (42)

20131a4222_gm 2024-08-07

Harry Bro unable to download the sheet.

REPLY

rambhopal118_gm 2024-07-30

Harry bhai chatsheet Download nahi ho rahi h

REPLY

ds1874037_gm 2024-07-27

Error.....No downloaded

REPLY

princerajsinha863_gm 2024-07-23

Download nahi ho raha hai

REPLY

raushankumar6203961_gm 2024-07-21

Harry bhai Java bhi cheatsheet nhi download ho raha hai.

REPLY