Java Cheat Sheet

Java is a programming language and platform that has been widely used since its development by James Gosling in 1982. It follows the Object-oriented Programming concept and can run programs written in any programming language. Java is a high-level, object-oriented, secure, robust, platform-independent, multithreaded, and portable programming language All those words are collectively called Java Buzzwords. It is commonly used for programming web-based, window, enterprise, and mobile applications. This **Java Cheat Sheet** article has been written by experts in Java and based on the experience of students who have recently undergone Java interviews.

This **Core Java Cheat Sheet** has been designed by Java experts, based on the experience of students who have recently undergone Java interviews. Whether you are a beginner or an experienced Java developer, this Java Cheat Sheet is a valuable resource for quickly accessing essential syntax, concepts, and best practices related to Java Programming.

Learn Java Programming: Basics to Advanced Concepts

- Java Programming Terminologies
- Java Basics

1. Java Programming Terminologies

- **JVM:** executes the bytecode generated by the compiler.
- **Bytecode:** The Javac compiler of JDK compiles the Java source code into bytecode so that it can be executed by JVM.
- **JDK:** It is a complete Java development kit that includes everything including compiler, Java Runtime Environment (JRE), java debuggers, java docs, etc.
- **JRE:** allows the Java program to run, however, we cannot compile it.
- **Garbage Collector:** To delete or recollect that memory JVM has a program called Garbage Collector.
- **Finalize method:** this function is triggered by the garbage collector just before an object is deleted or destroyed.

2. Java Basics

Now, we will explore some of the fundamental concepts often utilized in the Java programming language.

Object – An object refers to an entity that possesses both behavior and state, such as a bike, chair, pen, marker, table, and car. These objects can be either tangible or intangible, including the financial system as an example of an intangible object.

There are three characteristics of an object:

- **State:** The data (value) of an object is represented by its state.
- **Behaviour:** The functionality of an object, such as deposit, withdrawal, and so on, is represented by the term behaviour.
- **Identity:** A unique ID is often used to represent an object&#x2019s identification. The value of the ID is hidden from the outside user. The JVM uses it internally to uniquely identify each object.

Class – A class is a collection of objects with similar attributes. It&#x2019s a blueprint or template from which objects are made. It&#x2019s a logical thing. It can&#x2019t be physical. In Java, a class definition can have the following elements:

- **Modifiers:** A class can be private or public, or it can also have a default access level
- **class keyword:** To construct a class, we use the class keyword.
- **class name:** The name of the class should usually start with a capital letter.

- **Superclass (optional):** If the class has any superclass, we use the extends keyword and we mention the name of the superclass after the class name.
- **Interface (optional):** If the class implements an interface, we use the implements keyword followed by the name of the interface after the class name.

Constructors: In Java, a constructor is a block of code similar to a method. Whenever a new class instance is created, the constructor is called. The memory allocation for the object only happens when the constructor is invoked.

There are two types of constructors in Java. They are as follows:-

Default Constructor – A default constructor is a type of constructor that does not require any parameters. When we do not declare a constructor for a class, the compiler automatically generates a default constructor for the class with no arguments.

Parameterised Constructor – A parameterized constructor is a type of constructor that requires parameters. It is used to assign custom values to a class's fields during initialization.

Keyword – In Java, *Reserved words are* also known as keywords. These are particular terms that hold specific meanings. Java has 61 Reserved Keywords that are predefined and cannot be used as variable, object, or class names. Here's a list of the keywords used in Java:-

Keyword	Use Case
abstract	Used to declare an abstract class or abstract method
assert	Used to check assertions during debugging
boolean	Represents a boolean value (true or false)
break	Exits from a loop or a switch statement
byte	Represents a signed 8-bit integer

Keyword	Use Case
case	Used in a switch statement to define a case
catch	Catches exceptions thrown in a try block
char	Represents a 16-bit Unicode character
class	Declares a class
const*	Not used in Java, reserved for future use
continue	Skips the rest of the loop and starts the next iteration
default	Used in a switch statement as a default case
do	Starts a do-while loop
double	Represents a 64-bit double-precision floating-point number
else	Used in an if-else statement
enum	Declares an enumeration type
exports	Used in module declarations to specify exported packages
extends	Indicates a class is derived from another class
final	Declares a variable, method, or class as final (unchangeable)
finally	Defines a block of code to be executed after try-catch
float	Represents a 32-bit single-precision floating-point number
for	Starts a for loop

Keyword	Use Case
goto*	Not used in Java, reserved for future use
if	Used in an if statement
implements	Indicates a class is implementing an interface
import	Imports classes, packages, or individual members
instanceof	Tests if an object is an instance of a specific class
int	Represents a 32-bit integer
interface	Declares an interface
long	Represents a 64-bit integer
module*	Defines a module, reserved for future use
native	Indicates a method is implemented in platform-specific code
new	Creates a new object
open	Used in module declarations to specify open packages
opens	Used in module declarations to specify opened packages
private	Defines a private access modifier
protected	Defines a protected access modifier
provides	Used in module declarations to specify service providers
public	Defines a public access modifier

Keyword	Use Case
requires	Used in module declarations to specify required modules
return	Exits a method and returns a value
short	Represents a 16-bit integer
static	Declares a static variable or method
strictfp	Ensures consistent floating-point calculations
super	Refers to the parent class
switch	Selects one of many code blocks to be executed
synchronized	Defines a synchronized block or method
this	Refers to the current instance of the class
throw	Throws an exception
throws	Declares exceptions that a method may throw
to	Used in switch expressions to specify case values
transient	Indicates a member variable should not be serialized
while	Starts a while loop
transitive	Used in module declarations to specify transitive dependencies
try	Defines a block of code to be tested for exceptions
uses	Used in module declarations to specify service uses

Keyword	Use Case
void	Defines a method that does not return a value
volatile	Indicates a variable may be modified by multiple threads
with	Used in switch expressions to specify pattern matching
_	Reserved for future use

3. Java Program to Print “Hello World”

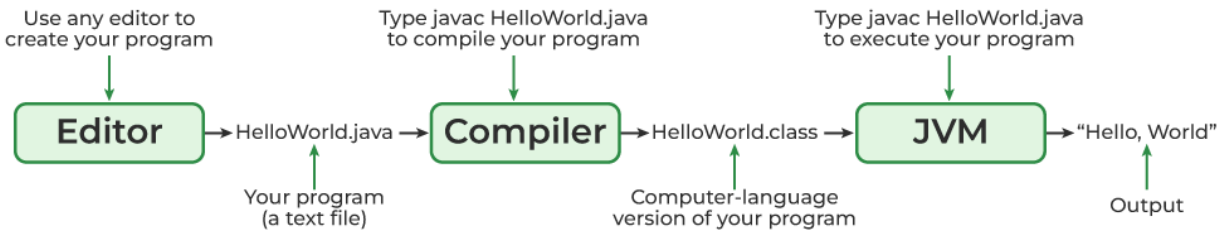
Java

```
// Java Program to Print
// Hello World
class GFG {
public static void main (String[] args) {
System.out.println("Hello World!");
}
}
```

Output

```
Hello World!
```

Editing, Compiling, and Executing



4. Data Types in Java

Data Types in Java are the different values and sizes that can be stored in the variable according to the requirements.

Java Datatype are further two types:-

1. Primitive Data Type in Java

In Java, primitive data types serve as the foundation for manipulating data. They are the most basic types of data that the Java programming language uses. Java has several primitive data types, including:

Type	Size	Example Literals	Range of values
boolean	1 bit	true, false	true, false
byte	8 bits	(none)	-128 to 127
char	16 bits	&#x2018a&#x2019, &#x2018\u0041&#x2019, &#x2018\101&#x2019, &#x2018\\&#x2019, &#x2018\&#x2019,	characters representation of ASCII values 0 to 255

Type	Size	Example Literals	Range of values
		&#x2018\n&#x2019, &#x2018β&#x2019	
short	16 bits	(none)	-32,768 to 32,767
int	32 bits	-2,-1,0,1,2	-2,147,483,648 to 2,147,483,647
long	64 bits	-2L,-1L,0L,1L,2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	32 bits	1.23e100f , -1.23e-100f , .3f ,3.14F	upto 7 decimal digits
double	64 bits	1.23456e300d , -123456e-300d , 1e1d	upto 16 decimal digits

2. Non-primitive Data Type

Non-primitive datatypes are created from primitive datatypes. Examples of non-primitive datatypes include arrays, stacks, and queues.

5. Java Comments

There are three types of comments in Java

1. Single Line comment

To comment on a single line of code, you can use a double forward slash "//" followed by your message. This syntax is commonly used for coding.

Java

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {
    public static void main(String[] args)
    {
        System.out.println("GFG!");
        // System.out.println("Thi line is not executed");
    }
}
```

Output

```
GFG!
```

2. Multi-line comment

If you need to comment on multiple lines of code, you can utilize the syntax of a double forward slash "/*". Simply enter your message between the two symbols, and complete the comment with "*/". This is a widely used syntax in coding.

Java

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {
    public static void main(String[] args)
    {
        System.out.println("GFG!");
        /* System.out.println("This line is Ignored ny
        comiler"); System.out.prinln("Including this");
        */
```

```
    }
}
```

Output

```
GFG!
```

3. JavaDoc Type Comment

When working on a project or software package, it is often helpful to use this method as it can assist in creating a documentation page for reference. This page can provide information on available methods, their parameters, and more, making it a useful tool for learning about the project.

Syntax:

```
/** Comment starts
*This is
*sample comment */
```

6. Java Variables

Variables are the containers that save the data values. Each variable is assigned according to the data type it is assigned.

Syntax:

```
data_type var_name;
```

Types of Variables

There are 3 types of Data types in Java as mentioned below:

1. Local Variables

A variable defined within a block or method or constructor is called a local variable.

2. Instance Variables

Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

3. Static Variables

Static variables are also known as class variables. The static variables are declared using the static keyword within a class outside of any method, constructor, or block.

Below is the Example of the above topic:

Java

```
// Java Program to demonstrate
// Variables
import java.io.*;

class GFG {
    // instance variable
    public int revise;

    // static variable
    public static int count = 0;

    public static void help(int i)
    {
        // here int i is local variable
        // changes will not affect original value
        GFG.count++;
        System.out.println(i);
    }

    public static void main(String[] args)
    {
        // local variable
        int i = 10;

        System.out.println("Before Calling function count:"
                           + GFG.count);

        help(i);
        System.out.println("After Calling function count:"
                           + GFG.count);
```

```
        GFG temp = new GFG();
        temp.revise = i + count;

        System.out.println("Instance variable value:"
                           + temp.revise);
    }
}
```

Output

```
Before Calling function count:0
10
After Calling function count:1
Instance variable value:11
```

7. Access Modifiers in Java

Access modifiers help to restrict the scope of a class, constructor, variable, method, or data member. It provides security, accessibility, etc to the user depending upon the access modifier used with the element

Types of Access Modifiers in Java

There are four types of access modifiers available in Java:

- **Default &#x2013 No keyword required**
- **Private**
- **Protected**
- **Public**

	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package N0on-Subclass	No	No	No	Yes

8. Operators in Java

Comparison Operators are the Operators which return a boolean value as the result means the answer will be either true or false.

Operators	Meaning	True	False
==	Equal	1==1	1==2
!=	Not Equal	1!=2	1!=1
<	Less Than	1<2	2<1
<=	Less Than Equal to	1<=1	2<=1
>	Greater Than	3>2	2>3
>=	Greater Than Equal to	3>=3	2>=3

Precedence and Associativity of Operator in Java

Here is the table representing the precedence order of Java operators from high to low as we move from top to bottom.

Operators	Associativity	Type
++, &#x2013	Right to Left	Unary postfix
++, –, +, -, !	Right to Left	Unary prefix
/, *, %	Left to Right	Multiplicative
+, –	Left to Right	Additive
<, <=, >, >=	Left to Right	Relational
==, !==	Left to Right	Equality
&	Left to Right	Boolean Logical AND
^	Left to Right	Boolean Logical Exclusive OR
|	Left to Right	Boolean Logical Inclusive OR
&&	Left to Right	Conditional AND
||	Left to Right	Conditional OR
?:	Right to Left	Conditional
=, +=, -=, *=, /=, %=	Right to Left	Assignment

9. Identifiers in Java

The name that we give to the class, variable, and methods are formally called identifiers in Java. and for defining Identifier there are some rules of it we need to take care of that while defining Identifiers.

Rules of defining Java identifiers:-

- Valid Java Identifiers have strict guidelines to avoid compile-time errors. Similar rules apply to C and C++
- Only letters (both upper and lowercase), numbers, dollar signs, and underscores are allowed as identifiers in Java. Special characters like “@” are not permitted.
- Numbers should not be used to begin identifiers ([0-9]). &#x201c123geeks,&#x201d for example, is not a valid Java identifier.
- Case matters when it comes to Java Identifiers. For example &#x2018bit&#x2019 and &#x2018BIT&#x2019 would be considered as different identifiers in Java.
- The length of the identifier is not limited, however, it is recommended that it be kept to a maximum of 4&#x201315 letters.
- Using reserved words as identifiers is not allowed in Java. This includes the term “while,” as it is one of the 53 reserved terms.
- The length of the identifier is not limited, however, it is recommended that it be kept to a maximum of 4&#x201315 letters.

10. Control Flow in Java

1. If, else-if, else

Java

```java
// Java Program to implement
// if - else if - else
import java.io.*;

// Driver Class
class GFG {
    // main function
    public static void main(String[] args)
    {
        int a = 1, b = 2;

        if (a < b)
            System.out.println(b);
        else if (a > b)
            System.out.println(a);
        else
            System.out.println(a + "==" + b);
```

```
    }
}
```

Output

```
2
```

2. Nested if – else

A nested if is an if statement that is the target of another if or else. Nested if statements mean an if statement inside an if statement.

Java

```java
// Java program to illustrate nested-if statement
import java.util.*;

class NestedIfDemo {
    public static void main(String args[])
    {
        int i = 10;

        if (i == 10 || i < 15) {
            // First if statement
            if (i < 15)
                System.out.println("i is smaller than 15");

            // Nested - if statement
            // Will only be executed if statement above
            // it is true
            if (i < 12)
                System.out.println(
                    "i is smaller than 12 too");
        }
        else {
            System.out.println("i is greater than 15");
        }
    }
}
```

Output

```
i is smaller than 15
i is smaller than 12 too
```

2. Switch Statement

Java

```java
// Java program to Demonstrate Switch Case

// Driver class
public class GFG {

    // main driver method
    public static void main(String[] args)
    {
        int day = 5;
        String dayString;

        // Switch statement with int data type
        switch (day) {

        // Case
        case 1:
            dayString = "Monday";
            break;

        // Case
        case 2:
            dayString = "Tuesday";
            break;

        // Case
        case 3:
            dayString = "Wednesday";
            break;

        // Case
        case 4:
            dayString = "Thursday";
            break;

        // Case
        case 5:
            dayString = "Friday";
            break;
```

```
        // Case
        case 6:
            dayString = "Saturday";
            break;

        // Case
        case 7:
            dayString = "Sunday";
            break;

        // Default case
        default:
            dayString = "Invalid day";
        }

        System.out.println(dayString);
    }
}
```

Output

```
Friday
```

Loops in Java

Loops are used for performing the same task multiple times. There are certain loops available in Java as mentioned below:

1. For Loop
2. While Loop
3. do-while

Java

```
// Java Program to implement
// For loop
import java.io.*;

// Driver Class
class GFG {
    // Main function
    public static void main(String[] args)
```

```
    {
        int n = 5;
        for (int i = 1; i <= n; i++) {
            System.out.println(i);
        }
    }
}
```

Java

```
// Java Program to demonstrate
// the working of while loop
import java.io.*;

// Driver Class
class GFG {
    // main function
    public static void main(String[] args)
    {
        int i = 16;

        // Working of while loop
        while (i != 0) {
            System.out.println(i);

            if (i > 4)
                i -= 4;
            else
                i -= 1;
        }
    }
}
```

Java

```
// Java Program to demonstrate
// do-while loop

// Driver Class
class GFG {
    // main function
    public static void main(String[] args)
    {
        int i = 1;
        do {
            System.out.println(i);
```

```
            i++;
        } while (i <= 5);
    }
}
```

Output

```
1
2
3
4
5
```

11. Methods in Java

Java Methods are collections of statements that perform some specific task and return the result.

Syntax of Method

```
<access_modifier> <return_type> <method_name>( list_of_parameters)
{
    //body
}
```

Below is the implementation of the above method:

Java

```
// Java Program to demonstrate the
// Use of Methods
import java.io.*;

// Driver Class
class GFG {
    public static int sum(int i, int j) { return i + j; }

    // main method
    public static void main(String[] args)
    {
        int n = 3, m = 3;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
```

```
                System.out.print(sum(i, j) + " ");
            }
            System.out.println();
        }
    }
}
```

Output

```
0 1 2
1 2 3
2 3 4
```

12. Java Input-output (I/O operations)

We can only print using System.out but can use different print varieties in it:

1. print

The cursor remains on the same line

```
System.out.print(" GFG " + x);
```

2. println

Cursor Moves to a new line

```
System.out.println(" GFG " + x);
```

3. printf

The cursor remains on the same line

```
System.out.printf(" GFG %d",x);
```

Formatted Print

```
System.out.printf("%7.5f", Math.PI);
```

Explanation of the above statement:

7 is the Field width and .5 is the precision fo the floating number printed

. So, answer will be ***3.14159***

Taking Input in Java

1. Parsing Command-line arguments

We can take input using the Command line simp

Java

```
// Java Program to take Input
// from command line arguments
class GFG {
    public static void main(String args[])
    {

        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

```
javac GFG.java
java GFG This is just to check 2
```

Output:

```
This
is
just
to
Check
2
```

2. Buffer Reader Class

InputStreamReader() is a function that converts the input stream of bytes into a stream of characters so that it can be read as BufferedReader expects a stream

of characters.

Below is the implementation of the above method:

Java

```
// Java Program for taking user
// input using BufferedReader Class
import java.io.*;

class GFG {

    // Main Method
    public static void main(String[] args)
        throws IOException
    {
        // Creating BufferedReader Object
        // InputStreamReader converts bytes to
        // stream of character
        BufferedReader bfn = new BufferedReader(
            new InputStreamReader(System.in));

        // String reading internally
        String str = bfn.readLine();

        // Integer reading internally
        int it = Integer.parseInt(bfn.readLine());

        // Printing String
        System.out.println("Entered String : " + str);

        // Printing Integer
        System.out.println("Entered Integer : " + it);
    }
}
```

Output:

```
ABC
11
Entered String : ABC
Entered Integer : 11
```

3. Scanner Class

Syntax:

```
Scanner scn = new Scanner(System.in);
```

Below is the implementation of the above method:

Java

```java
// Java Program to take input using
// Scanner Class in Java
import java.io.*;
import java.util.Scanner;

class GFG {
    public static void main(String[] args)
    {
        // creating the instance of class Scanner
        Scanner obj = new Scanner(System.in);
        String name;
        int rollno;
        float marks;

        // taking string input
        System.out.println("Enter your name");
        name = obj.nextLine();

        // taking float input
        System.out.println("Enter your marks");
        marks = obj.nextFloat();

        // printing the output
        System.out.println("Name :" + name);
        System.out.println("Marks :" + marks);
    }
}
```

Output:

```
ABC
65
Name :ABC
Marks :65
```

13. Java Polymorphism

Polymorphism: It is the ability to differentiate between entities with the same name efficiently.

Compile-time polymorphism: Static polymorphism, also called compile-time polymorphism, is achieved through function overloading in Java. Static polymorphism, also called compile-time polymorphism, is achieved through function overloading in Java.

Method Overloading: If there are multiple functions that have the same name but different parameters, it is known as overloading. Alterations in the number or type of arguments can result in the overloading of functions.

Example:

Java

```java
// Java Program to demonstrate
// Polymorphism
import java.io.*;

// Class
class ABC {
    // Sum Method with int returning value
    public int sum(int x, int y) { return x + y; }

    // Sum Method with float returning value
    public double sum(double x, double y) { return x + y; }
}

// Driver Class
class GFG {
    // main function
    public static void main(String[] args)
    {
        ABC temp = new ABC();

        System.out.println(temp.sum(1, 2));
        System.out.println(temp.sum(3.14, 4.23));
    }
}
```

Output

```
3
7.370000000000001
```

14. Java Inheritance

Inheritance: It is the mechanism in Java by which one class is allowed to inherit the features (fields and methods) of another class.

Java

```java
// Java Program to demonstrate
// Inheritance (concise)
import java.io.*;

// Base or Super Class
class Employee {
    int salary = 70000;
}

// Inherited or Sub Class
class Engineer extends Employee {
    int benefits = 15000;
}

// Driver Class
class Gfg {
    public static void main(String args[])
    {
        Engineer E1 = new Engineer();
        System.out.println("Salary : " + E1.salary
                           + "\nBenefits : " + E1.benefits);
    }
}
```

Output

```
Salary : 70000
Benefits : 15000
```

15. Java Maths Class

In Java, there exists a Math Library that can simplify intricate calculations

Library:

```
import java.lang.Math;
```

Use:

```
Math.function_name <parameters>
```

Functions	Returns or Calculates
min(int a,int b)	minimum of a and b
max(int a,int b)	maximum of a and b
sin(int θ)	sine of θ
cos(int θ)	cosine of θ
tan(int θ)	tangent of θ
toRadians(int θ)	Convert angle in degrees(θ) to radians
toDegrees(int θ)	Convert angle int radians(θ) to degrees
exp(int a)	exponential value e^a
log(int a)	natural log -> $\log_e a$
pow(int a, int b)	power a^b
round(double a)	round to the nearest integer

Functions	Returns or Calculates
random()	returns the random value in [0,1)
sqrt(int a)	the square root of a
E	value of e (constant value)
PI	Value of π(constant value)

Note: *All the functions mentioned above can be used with either data-types not bounded any single data-types.*

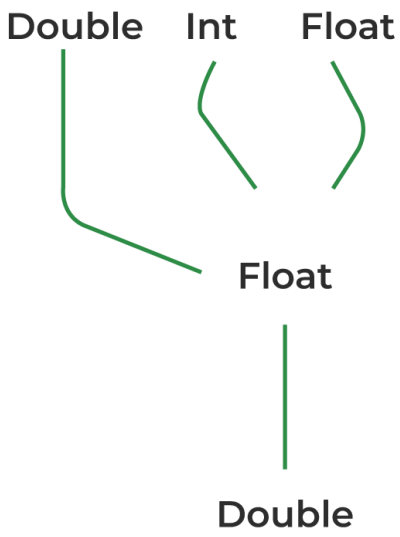
16. Typecasting In Java

1. Type Casting in Java

Type Casting is a method in which we convert from one data type to another.

2. Type Conversion in Java

Type Conversion in Java is a technique where we can convert from double, int, and float to double.



Below is the implementation of the above methods:

Java

```
// Java Program to Implement
// Type Casting of the Datatype
import java.io.*;

// Main class
public class GFG {
    // Main driver method
    public static void main(String[] args)
    {
        int a = 3;

        // Casting to Large datatype
        double db = (double)a;

        // Print and display the casted value
        System.out.println(db);
    }
}
```

Java

```
// Java Program to illustrate
// Type Conversion
import java.io.*;

// Driver Class
class GFG {
    // main function
```

```
    public static void main(String[] args)
    {
        long a = 1;
        byte b = 1;
        double c = 1.0;

        // Type Conversion
        double final_datatype = (a + b + c);
        // Printing the sum of all three initialized values
        System.out.print(final_datatype);
    }
}
```

Output

```
3.0
```

17. Arrays in Java

Arrays are the type of data structure that can store data of similar data types. Arrays are allocated in contiguous memory allocation with a fixed size.

Syntax:

```
// Both Methods are correct
int arr[]=new int[20];
int[] arr=new int[20];
```

Below is the implementation of the above method:

Java

```
// Java Program to implement
// arrays
class GFG {
    public static void main(String[] args)
    {
        // declares an Array of integers.
        int[] arr = new int[5];

        // Allocating memory to the
        // elements
        arr[0] = 10;
        arr[1] = 20;
```

```
        arr[2] = 30;
        arr[3] = 40;
        arr[4] = 50;

        // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
            System.out.println("arr[" + i + "] :" + arr[i]);
    }
}
```

Output

```
arr[0] :10
arr[1] :20
arr[2] :30
arr[3] :40
arr[4] :50
```

Multidimensional Arrays in Java

Arrays are not bounded to be single-dimensional but can store elements in multidimensional.

Syntax:

```
int[][] arr= new int[3][3];
int arr[][]=new int[3][3];
```

Below is the implementation of the above method:

Java

```
// Java Program to implement
// 2-D dimensional array

// driver class
public class multiDimensional {
    // main function
    public static void main(String args[])
    {
        // declaring and initializing 2D array
        int arr[][]
```

```
            = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

        // printing 2D array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                System.out.print(arr[i][j] + " ");

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

Output

```
1 2 3
4 5 6
7 8 9
```

18. Strings in Java

Strings are the type of objects that can store the character of values. A string acts the same as an array of characters in Java.

Syntax:

```
String abc=" ";
```

CharSequence Interface in Java

1. StringBuffer

```
StringBuffer s = new StringBuffer("GeeksforGeeks");
```

2. StringBuilder

```
StringBuilder str = new StringBuilder();
str.append("GFG");
```

3. String Tokenizer

```
public StringJoiner(CharSequence delimiter)
```

19. Java Regex

Regular Expressions, or Regex for short, is a Java API that allows users to define String patterns for searching, manipulating, and modifying strings. It is commonly used for setting limits on various areas of strings such as email validation and passwords. The java.util.regex package contains regular expressions and consists of three classes and one interface. The following table lists the three classes included in the java.util.regex package:

Class	Description
util.regex.Pattern	It is used to define patterns.
util.regex.Matcher	It is used to conduct match operations on text using patterns.
PatternSyntaxException	In a regular expression pattern, it's used to indicate a syntax problem.

Pattern class: This class does not have any public constructors. Instead, it consists of a group of regular expressions that can be utilized to define different types of patterns. To do this, simply execute the compile() method with a regular expression as the first input. After execution, the method will return a pattern.

The table below provides a list of the methods in this class along with their descriptions.

Method	Description
compile(String regex)	Compiles a regular expression into a pattern.
compile(String regex, int flags)	Turns a regular expression into a pattern using the flags provided.

Method	Description
flags()	Gets the match flags for this pattern.
matcher(CharSequence input)	Builds a matcher that compares the given input to the pattern.
matches(String regex, CharSequence input)	Matches a regular expression and tries to match it against the given input.
pattern()	Gets the regular expression that was used to create this pattern.
quote(String s)	Generates a literal pattern String from the given String.
split(CharSequence input)	Splits the given input sequence around patterns that match this pattern.
split(CharSequence input, int limit)	Divides the given input sequence into sections based on matches to this pattern. The number of times the pattern is applied is controlled by the limit parameter.
toString()	Returns the pattern's string representation.

Matcher Class: To evaluate previously described patterns through match operations on an input string in Java, the 'object' is utilized. No public constructors are defined here. One can execute a matcher() on any pattern object to achieve this. The table below displays the methods present in this class along with their descriptions:To evaluate previously described patterns through match operations on an input string in Java, the 'object' is utilized. No public constructors are defined here. One can execute a matcher() on any pattern object to achieve this.

The table below displays the methods present in this class along with their descriptions:

Method	Description
find()	find() is mostly used to look for multiple occurrences of regular expressions in a text.
find(int start)	It is used to find occurrences of regular expressions in the text starting from the provided index.
start()	start() is used to retrieve the start index of a match found with the find() method.
end()	It's used to acquire the end index of a match discovered with the find() method.
groupCount()	groupCount() is a function that returns the total number of matched subsequences.
matches()	It's used to see if the pattern matches the regular expression.

20. Java Exception Handling

Exception:– An Exception is an unexpected error that occurs during the run-time of a program.

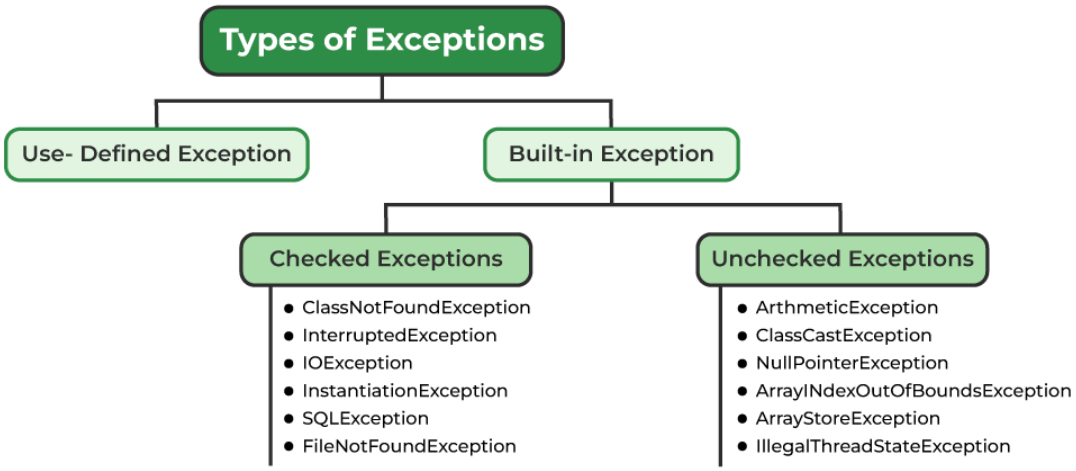
Error vs Exception: What is the difference?

When a program encounters an error, it means there is a significant issue that a well-designed program should not try to fix. On the other hand, an exception indicates a particular situation that a well-designed program should try to handle.

Types of Exceptions:-

Checked Exception:- This mainly includes IO Exceptions and Compile time Exceptions

Unchecked Exceptions:– This covers both Runtime Exceptions and Null Pointer Exceptions.



Handle Exceptions

try block: The try block comprises a set of statements that may throw an exception.

Catch Block: When there is an uncertain condition in the try block, the catch block comes in handy to handle it. It is mandatory to have a catch block right after the try block to handle any exceptions that may be thrown by the try block.

Finally Block: When programming in Java, the finally block is a section of code that will always run, regardless of whether or not an exception has been caught. If there is a catch block following a try block, it will be executed before the finally block. However, if there is no catch block, the finally block will be executed immediately after the try block.

Example:-

```
try {
    // block of code to monitor for errors
    // the code you think can raise an exception
} catch (ExceptionType1 exOb) {
    // exception handler for ExceptionType1
} catch (ExceptionType2 exOb) {
    // exception handler for ExceptionType2
}
// optional
finally {  // block of code to be executed after try block ends
```

```
}
```

final vs finally vs finalize

Below is a table that outlines the distinctions between the terms final, finally, and finalize:

final

- A final keyword can be used with classes, methods, and variables.
- A final class cannot be inherited.
- A final method cannot be overridden.
- A final variable cannot be reassigned.

finally

- A finally block is always executed, regardless of whether an exception is thrown or not.
- The finally block is executed after the try block and catch block, but before the program control returns to the calling method.

connections.

finalize

- The finalize() method is called by the garbage collector when an object is no longer needed.
- The finalize() method can be used to perform cleanup operations, such as releasing resources or writing data to a file.
- The finalize() method is not guaranteed to be called, so it should not be used to perform critical operations.

throw keyword:- When using Java, the throw keyword is utilized to throw an exception from a block of code or a method. It is possible to throw either a checked or unchecked exception. The throw keyword is frequently used for throwing custom exceptions.

throws keyword:- If a try/catch block is not present, the throws keyword can be used to manage exceptions. This keyword specifies the specific exceptions that a method should throw if an exception happens.

21. Java Commands

Here are the most commonly used Java commands:

- `java -version`: Displays the version of the Java Runtime Environment (JRE) that is installed on your computer.
- `java -help`: Displays a list of all of the Java commands.
- `java -cp`: Specifies the classpath for the Java program that you are running.
- `java -jar`: Runs a Java Archive (JAR) file.
- `java -D`: Sets a system property.
- `java -X`: Specifies a non-standard option for the Java Virtual Machine (JVM).

Here are some other useful Java commands:

- `javac`: Compiles a Java source file into bytecode.
- `javap`: Disassembles a Java class file.
- `jdb`: A Java debugger.
- `jconsole`: A Java monitoring and management tool.
- `jvisualvm`: A Java profiling and diagnostic tool.

22. Java Generics

Generics means parameterized types. The idea is to allow type (Integer, String, &#x2026 etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Types of Java Generics

Generic Method: Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function, however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care

of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.

```
// To create an instance of generic class
BaseType <Type> obj = new BaseType <Type>()
```

Generic Functions: We can also write generic functions that can be called with different types of arguments based on the type of arguments passed to the generic method. The compiler handles each method.

Java

```
// Java program to show working of user defined
// Generic functions

class Test {
    // A Generic method example
    static<T> void genericDisplay(T element)
    {
        System.out.println(element.getClass().getName()
                           + " = " + element);
    }

    // Driver method
    public static void main(String[] args)
    {
        // Calling generic method with Integer argument
        genericDisplay(11);

        // Calling generic method with String argument
        genericDisplay("GeeksForGeeks");

        // Calling generic method with double argument
        genericDisplay(1.0);
    }
}
```

Output

```
java.lang.Integer = 11
java.lang.String = GeeksForGeeks
java.lang.Double = 1.0
```

23. Java Multithreading

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

- Extending the Thread class
- Implementing the Runnable Interface

Example: – Thread creation by extending the Thread class

To create a new thread in Java, we can extend the java.lang.Thread class and override its run() method. This is where the thread's execution starts. Then, we can create an instance of our class and call the start() method to begin the execution of the thread. The start() method will then call the run() method of the Thread object.

Java

```
class MultithreadingDemo extends Thread {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}

// Main Class
public class Multithread {
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i = 0; i < n; i++) {
```

```
            MultithreadingDemo object
                = new MultithreadingDemo();
            object.start();
        }
    }
}
```

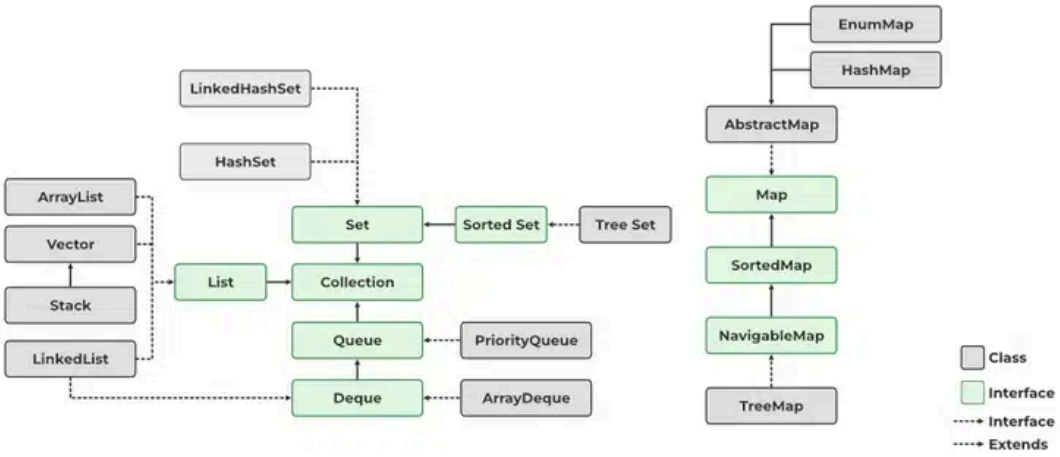
Output

```
Thread 15 is running
Thread 17 is running
Thread 14 is running
Thread 12 is running
Thread 13 is running
Thread 18 is running
Thread 11 is running
Thread 16 is running
```

24. Java Collections

The collection is a framework in Java that provides an architecture to store and manipulate objects in Java. It contains the interface as mentioned below:

- List Interface
- Queue Interface
- Deque Interface
- Set Interface
- SortedSet Interface
- Map Interface
- Collection Interface
- Iterable Interface



Syntax of the Interfaces:

Interfaces	Syntax
Iterable Interface	Iterator iterator();
List Interface	List <T> al = new ArrayList<> (); List <T> ll = new LinkedList<> (); List <T> v = new Vector<> ();
Queue Interface	Queue <T> pq = new PriorityQueue<> (); Queue <T> ad = new ArrayDeque<> ();
Deque Interface	Deque<T> ad = new ArrayDeque<> ();
Set Interface	Set<T> hs = new HashSet<> (); Set<T> lhs = new LinkedHashSet<> (); Set<T> ts = new TreeSet<> ();
Sorted Set Interface	SortedSet<T> ts = new TreeSet<> ();

Interfaces	Syntax
Map Interface	Map<T> hm = new HashMap<> (); Map<T> tm = new TreeMap<> ();

Why Use Java?

Java is simple for programmers to learn. Java is frequently chosen as the first programming language to learn. Furthermore, the sector continues to benefit from Java's prominence. The majority of websites and apps in the government, healthcare, defence, and educational sectors continue to use Java technology. Java is thus worthwhile to learn and use. If you choose a career in Java, you can follow a number of different career routes. Almost anything that Java can accomplish.

Why is Java so Popular?

Because Java includes a unique tool called the Java Virtual Machine that ensures the code functions consistently everywhere, it is feasible for Java to execute without any issues on many types of computers and machines. This makes Java a popular programming language. A rule in Java also known as WORA states that code only has to be written once and may execute anywhere. Java's high level of security, which shields the code from hackers, viruses, and other dangers, is another factor in its popularity. Java also enables the creation of many jobs that may run concurrently within a programme, making it quicker and more effective. Java provides a clever method for generating code that primarily concentrates on.

Features of Java

- Java is an **Object Oriented Programming language**.
- Java is **Platform Independent** because it comes with its own platform to run applications.
- **Simple** to learn programming language because doesn&#x2019t contain Pointers and operator overloading.
- Java is **secure** because the Java program runs on Java virtual machine(JVM) so, if there is any problem occurred in the program it will remain inside the

JVM only, it will not affect the operating system.

- Java is **Architecture neutral** because it is independent of the platform so we do not have to design the program for different devices differently.
- Java is **portable** because, after the compilation of any Java program, it generates a bytecode that can run on any machine which has JVM.
- Java is **Robust** means Java comes with automatic garbage collection management and strong exception handling.
- Java supports **multithreading**, It is possible in Java that we can divide a program into two or many subprograms and run these programs/threads at the same time.

Applications of Java Programming language

- Mobile Applications
- Desktop GUI Applications
- Artificial intelligence
- Scientific Applications
- Cloud Applications
- Embedded Systems
- Gaming Applications

Java Cheat Sheet – FAQs

1. Why Use Java?

Java is simple to learn programming language because doesn&#x2019t contain concepts like: Pointers and operator overloading and it is secure and portable.

2. What is the difference between C++ and Java?

C++	*JAVA*

C++ is platform dependent.	*Java is platform-independent.*
C++ uses a compiler only.	*Java uses a compiler and interpreter both.*
C++ support pointers and operator overloading.	*Java doesn&#x2019t support pointers and operator overloading concepts.*
C++ does not support the multithreading concept.	*Java supports the multithreading concept.*

3. What is the most important feature of Java?

The most important features of Java are Platform Independent and Object Oriented. That&#x2019s why Java is the most popular among high-level programming languages.

4. What are the main uses of Java?

Since Java is distributed and system independent, it can be used anywhere like:

1. *Web Development*
2. *Software Development*
3. *Mobile Application Development*
4. *Distributed Applications*
5. *Web servers*
6. *Enterprise Application*

5. How is Java useful in real life?

Java is useful in developing real-world web or mobile applications, and also useful to build servers. There are a lot of things you can do in Java, there are multiple libraries, and using that you can do anything. The application built in Java can be distributed over the internet or on any network.

6. What is the scope of Java?

The developer community of Java is so vast and is the strength of the Java language. This technology is growing at a fast pace and job opportunities are also increasing for Java developers having good knowledge of Java technologies. As Java is scalable, you can find Java in mobile applications, software applications, servers, and web applications.

Last Updated : 05 Mar, 2024

23

Next

Java Cheat Sheet

Share your thoughts in the comments

Add Your Comment

Similar Reads

jQuery Cheat Sheet – A Basic Guide to jQuery

Tkinter Cheat Sheet

CSS Cheat Sheet - A Basic Guide to CSS

Linux Commands Cheat Sheet

Git Cheat Sheet

ggplot2 Cheat Sheet

Subnet Mask Cheat Sheet

C++ STL Cheat Sheet

C Cheat Sheet

React Cheat Sheet

GeeksforGeeks

Article Tags : GFG Sheets , Java

Practice Tags : Java

GeeksforGeeks
Sanchhaya Education Private Limited
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

Company

About Us

Legal

Careers

In Media

Contact Us

Advertise with us

GFG Corporate Solution

Placement Training Program

Explore

Job-A-Thon Hiring Challenge

Hack-A-Thon

GfG Weekly Contest

Offline Classes (Delhi/NCR)

DSA in JAVA/C++

Master System Design

Master CP

GeeksforGeeks Videos

Geeks Community

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

DSA

Data Structures

Algorithms

DSA for Beginners

Basic DSA Problems

DSA Roadmap

DSA Interview Questions

Competitive Programming

Data Science & ML

Data Science With Python

Data Science For Beginner

Web Technologies

HTML

CSS

Machine Learning Tutorial

ML Maths

Data Visualisation Tutorial

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

JavaScript

TypeScript

ReactJS

NextJS

NodeJs

Bootstrap

Tailwind CSS

Python Tutorial

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

Web Scraping

OpenCV Tutorial

Python Interview Question

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

DevOps Roadmap

System Design

High Level Design

Low Level Design

UML Diagrams

Interview Guide

Design Patterns

OOAD

System Design Bootcamp

Interview Questions

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

Commerce

Accountancy

Business Studies

Economics

Management

HR Management

Finance

Income Tax

UPSC Study Material

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economy Notes

Ethics Notes

Previous Year Papers

Preparation Corner

Company-Wise Recruitment Process

Resume Templates

Aptitude Preparation

Puzzles

Company-Wise Preparation

Companies

Colleges

Competitive Exams

JEE Advanced

UGC NET

SSC CGL

SBI PO

SBI Clerk

IBPS PO

IBPS Clerk

More Tutorials

Software Development

Software Testing

Product Management

Project Management

Linux

Excel

All Cheat Sheets

Free Online Tools

Typing Test

Image Editor

Code Formatters

Code Converters

Currency Converter

Random Number Generator

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Share your Experiences

Internships

Random Password Generator