



Java Cheat Sheet

Last Updated : 20 Sep, 2024

Java is a programming language and platform that has been widely used since its development by James Gosling in 1991. It follows the Object-oriented Programming concept and can run programs written on any OS platform. 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, Windows, enterprise, and

Courses @90% Refund HTML Cheat Sheet CSS Cheat Sheet JS Cheat Sheet Bootstrap Cheat Sheet jQuery Cheat Sheet

Experts in Java and is 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 for competitive programming is a valuable resource for quickly accessing essential syntax, concepts, and best practices related to [Java Programming](#).

Java Cheat Sheet for Competitive Programming: Basics to Advanced Concepts

- [Java Programming Terminologies](#)

- [Java Basics](#)
- [Java Program to Print “Hello World”](#)
- [Datatypes in Java](#)
- [Java Comments](#)
- [Java Variables](#)
- [Access Modifiers in Java](#)
- [Operators in Java](#)
- [Identifiers in Java](#)
- [Control Flow in Java](#)
- [Methods in Java](#)
- [Java Input-output \(I/O operations\)](#)
- [Java Polymorphism](#)
- [Java Inheritance](#)
- [Java Maths Class](#)
- [Typecasting In Java](#)
- [Arrays in Java](#)
- [Strings in Java](#)
- [Java Regex](#)
- [Java Exception Handling](#)
- [Java Commands](#)
- [Java Generics](#)
- [Java Multithreading](#)
- [java Collections](#)

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's 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's a blueprint or template from which objects are made. It's a logical thing. It can't 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.

Ad removed.

[Show details](#)

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
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

Keyword	Use Case
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
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

Keyword	Use Case
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
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

Keyword	Use Case
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
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”



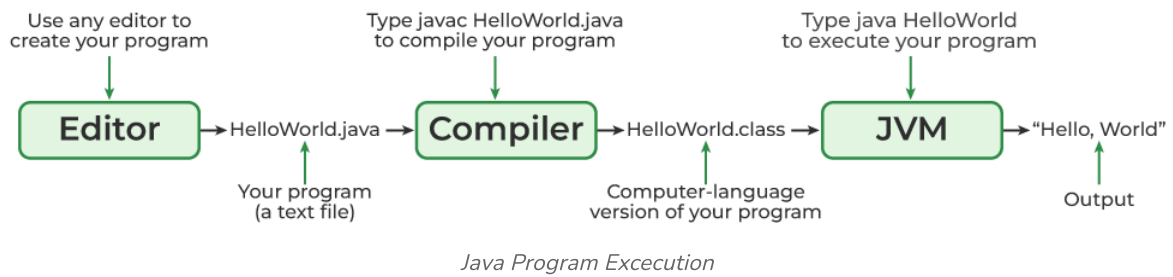
```

1  // Java Program to Print
2  // Hello World
3  class GFG {
4  public static void main (String[] args) {
5  System.out.println("Hello World!");
6  }
7  }
```

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	?, '\u0041', '\101', '\\', '\n', '\b'	characters representation of ASCII values 0 to 255
short	16 bits	(none)	-32,768 to 32,767
int	32 bits	-2,-1,0,1,2	-2,147,483,648 to

Type	Size	Example Literals	Range of values
			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.

```

1  /*package whatever //do not write package name here */
2
3  import java.io.*;
4
5  class GFG {
6      public static void main(String[] args)
7      {
8          System.out.println("GFG!");
9          // System.out.println("This line is not
           executed");
10     }
```

```
}
```

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.

```
1  /*package whatever //do not write package name here */
2
3  import java.io.*;
4
5  class GFG {
6      public static void main(String[] args)
7      {
8          System.out.println("GFG!");
9          /* System.out.println("This line is Ignored ny
10             comiler"); System.out.prinln("Including this");
11             */
12      }
13  }
```

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:

```
 1 // Java Program to demonstrate  
 2 // Variables  
3 import java.io.*;  
4  
5 class GFG {  
6     // instance variable  
7     public int revise;  
8  
9     // static variable  
10    public static int count = 0;
```

```

12     public static void help(int i)
13     {
14         // here int i is local variable
15         // changes will not affect original value
16         GFG.count++;
17         System.out.println(i);
18     }
19
20     public static void main(String[] args)
21     {
22         // local variable
23         int i = 10;
24
25         System.out.println("Before Calling function
count:"
26                             + GFG.count);
27
28         help(i);
29         System.out.println("After Calling function
count:"
30                             + GFG.count);
31
32         GFG temp = new GFG();
33         temp.revise = i + count;
34
35         System.out.println("Instance variable value:"
36                             + temp.revise);
37     }
38 }

```

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 – 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 Non-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
++, -	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]). `?geeks,` for example, is not a valid Java identifier.
- Case matters when it comes to Java Identifiers. For example `?it'` and `?IT'` 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 40 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 40 letters.

10. Control Flow in Java

1. If, else-if, else

```
1  // Java Program to implement
2  // if - else if - else
3  import java.io.*;
4
5  // Driver Class
6  class GFG {
7      // main function
8      public static void main(String[] args)
9      {
10         int a = 1, b = 2;
11
12         if (a < b)
13             System.out.println(b);
14         else if (a > b)
15             System.out.println(a);
16         else
17             System.out.println(a + "==" + b);
18     }
```



```
}
```

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.

```
1 // Java program to illustrate nested-if statement
2 import java.util.*;
3
4 class NestedIfDemo {
5     public static void main(String args[])
6     {
7         int i = 10;
8
9         if (i == 10 || i < 15) {
10             // First if statement
11             if (i < 15)
12                 System.out.println("i is smaller than
13 15");
14
15             // Nested - if statement
16             // Will only be executed if statement above
17             // it is true
18             if (i < 12)
19                 System.out.println(
20                     "i is smaller than 12 too");
21         }
22         else {
23             System.out.println("i is greater than 15");
24         }
25     }
}
```

Output

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

2. Switch Statement



```
1  // Java program to Demonstrate Switch Case
2
3  // Driver class
4  public class GFG {
5
6      // main driver method
7      public static void main(String[] args)
8      {
9          int day = 5;
10         String dayString;
11
12         // Switch statement with int data type
13         switch (day) {
14
15             // Case
16             case 1:
17                 dayString = "Monday";
18                 break;
19
20             // Case
21             case 2:
22                 dayString = "Tuesday";
23                 break;
24
25             // Case
26             case 3:
27                 dayString = "Wednesday";
28                 break;
29
30             // Case
31             case 4:
32                 dayString = "Thursday";
33                 break;
34
35             // Case
36             case 5:
37                 dayString = "Friday";
38                 break;
39
40             // Case
41             case 6:
42                 dayString = "Saturday";
```

```

        break;
44
45        // Case
46        case 7:
47            dayString = "Sunday";
48            break;
49
50        // Default case
51        default:
52            dayString = "Invalid day";
53        }
54
55        System.out.println(dayString);
56    }
57 }

```

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

Java



```

1    // Java Program to implement
2    // For loop
3    import java.io.*;
4
5    // Driver Class
6    class GFG {
7        // Main function
8        public static void main(String[] args)
9        {
10           int n = 5;
11           for (int i = 1; i <= n; i++) {

```

```

        System.out.println(i);
13    }
14    }
15    }

```

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:

```

1  // Java Program to demonstrate the
2  // Use of Methods
3  import java.io.*;
4
5  // Driver Class
6  class GFG {
7      public static int sum(int i, int j) { return i + j;
8      }
9
10     // main method
11     public static void main(String[] args)
12     {
13         int n = 3, m = 3;
14
15         for (int i = 0; i < n; i++) {
16             for (int j = 0; j < m; j++) {
17                 System.out.print(sum(i, j) + " ");
18             }
19         }
20     }
21 }

```

```
                System.out.println();
19            }
20        }
21    }
```

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 for 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

```
1 // Java Program to take Input
2 // from command line arguments
3 class GFG {
4     public static void main(String args[])
5     {
6
7         for (int i = 0; i < args.length; i++)
8             System.out.println(args[i]);
9     }
10 }
```

```
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:

```
1 // Java Program for taking user
2 // input using BufferedReader Class
3 import java.io.*;
4
5 class GFG {
6
```

```

        // Main Method
8      public static void main(String[] args)
9          throws IOException
10     {
11         // Creating BufferedReader Object
12         // InputStreamReader converts bytes to
13         // stream of character
14         BufferedReader bfn = new BufferedReader(
15             new InputStreamReader(System.in));
16
17         // String reading internally
18         String str = bfn.readLine();
19
20         // Integer reading internally
21         int it = Integer.parseInt(bfn.readLine());
22
23         // Printing String
24         System.out.println("Entered String : " + str);
25
26         // Printing Integer
27         System.out.println("Entered Integer : " + it);
28     }
29 }

```

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:

```

1  // Java Program to take input using
2  // Scanner Class in Java
3  import java.io.*;
4  import java.util.Scanner;
5

```

```

class GFG {
7     public static void main(String[] args)
8     {
9         // creating the instance of class Scanner
10        Scanner obj = new Scanner(System.in);
11        String name;
12        int rollno;
13        float marks;
14
15        // taking string input
16        System.out.println("Enter your name");
17        name = obj.nextLine();
18
19        // taking float input
20        System.out.println("Enter your marks");
21        marks = obj.nextFloat();
22
23        // printing the output
24        System.out.println("Name :" + name);
25        System.out.println("Marks :" + marks);
26    }
27 }

```

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:


```

1  // Java Program to demonstrate
2  // Polymorphism
3  import java.io.*;
4
5  // Class
6  class ABC {
7      // Sum Method with int returning value
8      public int sum(int x, int y) { return x + y; }
9
10     // Sum Method with float returning value
11     public double sum(double x, double y) { return x +
12     y; }
13 }
14 // Driver Class
15 class GFG {
16     // main function
17     public static void main(String[] args)
18     {
19         ABC temp = new ABC();
20
21         System.out.println(temp.sum(1, 2));
22         System.out.println(temp.sum(3.14, 4.23));
23     }
24 }

```

Output

```

3
7.3700000000000001

```

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.

```

1  // Java Program to demonstrate
2  // Inheritance (concise)
3  import java.io.*;
4

```

```

        // Base or Super Class
6    class Employee {
7        int salary = 70000;
8    }
9
10   // Inherited or Sub Class
11   class Engineer extends Employee {
12       int benefits = 15000;
13   }
14
15   // Driver Class
16   class Gfg {
17       public static void main(String args[])
18       {
19           Engineer E1 = new Engineer();
20           System.out.println("Salary : " + E1.salary
21                               + "\nBenefits : " +
22                               E1.benefits);
23       }
24   }

```

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

Functions	Returns or Calculates
<code>max(int a,int b)</code>	maximum of a and b
<code>sin(int θ)</code>	sine of θ
<code>cos(int θ)</code>	cosine of θ
<code>tan(int θ)</code>	tangent of θ
<code>toRadians(int θ)</code>	Convert angle in degrees(θ) to radians
<code>toDegrees(int θ)</code>	Convert angle int radians(θ) to degrees
<code>exp(int a)</code>	exponential value e^a
<code>log(int a)</code>	natural log $\rightarrow \log_e a$
<code>pow(int a, int b)</code>	power a^b
<code>round(double a)</code>	round to the nearest integer
<code>random()</code>	returns the random value in [0,1)
<code>sqrt(int a)</code>	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.

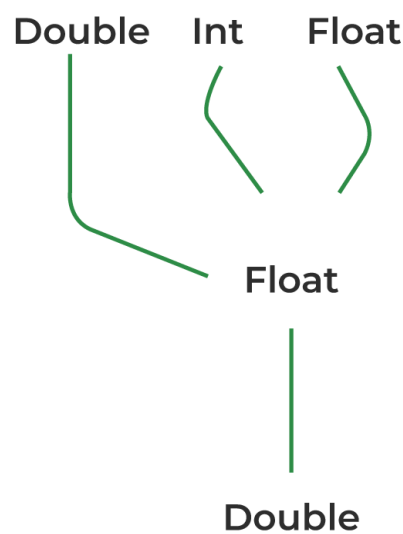
16. Typecasting In Java

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

```
1 // Java Program to Implement
2 // Type Casting of the Datatype
3 import java.io.*;
4
5 // Main class
6 public class GFG {
7     // Main driver method
8     public static void main(String[] args)
9     {
10         int a = 3;
11
12         // Casting to Large datatype
13         double db = (double)a;
14
15         // Print and display the casted value
```

```
        System.out.println(db);
17    }
18 }
```

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:

```
 1 // Java Program to implement
2 // arrays
 3 class GFG {
4     public static void main(String[] args)
5     {
6         // declares an Array of integers.
7         int[] arr = new int[5];
8
9         // Allocating memory to the
10        // elements
11        arr[0] = 10;
12        arr[1] = 20;
13        arr[2] = 30;
14        arr[3] = 40;
15        arr[4] = 50;
16
17        // accessing the elements of the specified
18        array
19        for (int i = 0; i < arr.length; i++)
20            System.out.println("arr[" + i + "]" : " +
21            arr[i]);
```

```
21    }  
    }
```

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:

```
 1 // Java Program to implement  
 2 // 2-D dimensional array  
3  
4 // driver class  
5 public class multiDimensional {  
6     // main function  
7     public static void main(String args[])  
8     {  
9         // declaring and initializing 2D array  
10        int arr[][]  
11        = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }  
12    };  
13  
14        // printing 2D array  
15        for (int i = 0; i < 3; i++) {  
16            for (int j = 0; j < 3; j++)  
17                System.out.print(arr[i][j] + " ");  
18            System.out.println();  
19        }  
20    }  
21 }
```

```
        }  
    20    }  
    21 }
```

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.
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.

Method	Description
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.

Method	Description
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.

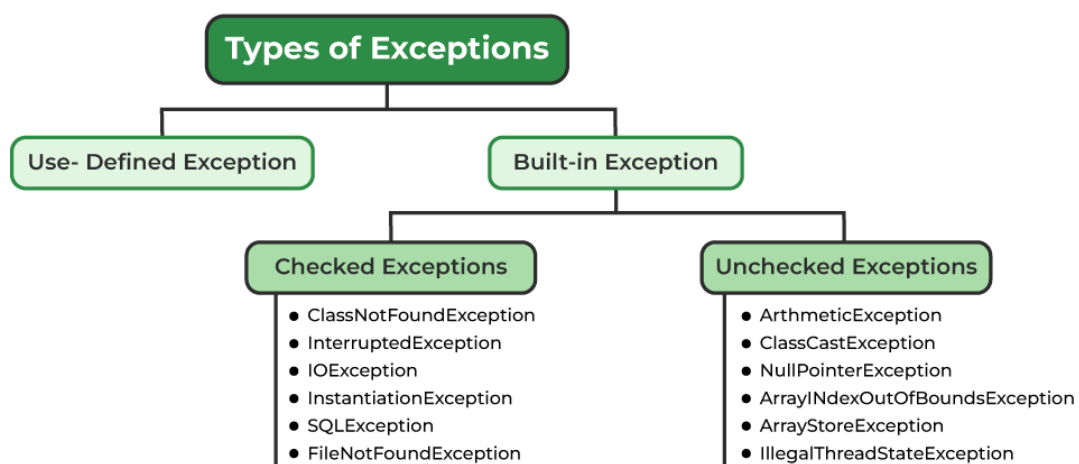
Error vs Exception: What is the difference?

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.
- The finally block can be used to close resources, such as files and database 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:


```

12      // Driver method
13      public static void main(String[] args)
14      {
15          // Calling generic method with Integer argument
16          genericDisplay(11);
17
18          // Calling generic method with String argument
19          genericDisplay("GeeksForGeeks");
20
21          // Calling generic method with double argument
22          genericDisplay(1.0);
23      }
24  }

```

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.



```

1  class MultithreadingDemo extends Thread {
2      public void run()

```

```

4      try {
5          // Displaying the thread that is running
6          System.out.println(
7              "Thread " +
8              Thread.currentThread().getId()
9              + " is running");
10     }
11     catch (Exception e) {
12         // Throwing an exception
13         System.out.println("Exception is caught");
14     }
15 }
16
17 // Main Class
18 public class Multithread {
19     public static void main(String[] args)
20     {
21         int n = 8; // Number of threads
22         for (int i = 0; i < n; i++) {
23             MultithreadingDemo object
24                 = new MultithreadingDemo();
25             object.start();
26         }
27     }
28 }

```

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

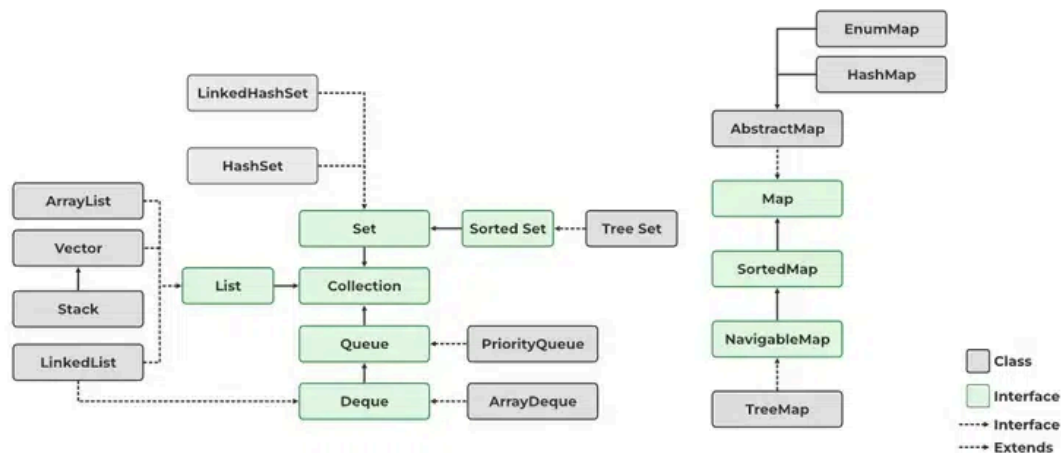
```

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<> ();

Interfaces	Syntax
	<pre>Set<T> lhs = new LinkedHashSet<> (); Set<T> ts = new TreeSet<> ();</pre>
Sorted Set Interface	<pre>SortedSet<T> ts = new TreeSet<> ();</pre>
Map Interface	<pre>Map<T> hm = new HashMap<> (); Map<T> tm = new TreeMap<> ();</pre>

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't 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

Conclusion

This Java Cheat Sheet serves as a quick reference guide for both beginners and experienced developers working with Java. By summarizing essential syntax, key concepts, and common commands, it aims to enhance your productivity and ensure you have the critical information at your fingertips. Whether you're preparing for an interview, working on a project, or simply refreshing your knowledge, this cheat sheet is designed to make your Java development process more efficient and effective.

Java Cheat Sheet – FAQs

1. Why Use Java?

Java is simple to learn programming language because doesn't contain concepts like: Pointers and operator overloading and it is secure and portable.

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

C++	JAVA
<i>C++ is platform dependent.</i>	<i>Java is platform-independent.</i>
<i>C++ uses a compiler only.</i>	<i>Java uses a compiler and interpreter both.</i>
<i>C++ support pointers and operator overloading.</i>	<i>Java doesn't support pointers and operator overloading concepts.</i>
<i>C++ does not support the multithreading concept.</i>	<i>Java supports the multithreading concept.</i>

3. What is the most important feature of Java?

The most important features of Java are Platform Independent and Object Oriented. That's 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.

Comment

More info

Next Article

C++ STL Cheat Sheet

Similar Reads

Geeksforgeeks Cheatsheets - All Coding Cheat Sheets Collections

Cheatsheets are short documents that contain all the most essential information about a specific technology in short, such as its syntax, commands, functions, or its features. Sheets are designed to help...

4 min read

Subnet Mask Cheat Sheet

A Subnet Mask is a numerical value that describes a computer or device's how to divide an IP address into two parts: the network portion and the host portion. The network element identifies the network to which...

9 min read

Git Cheat Sheet

Git Cheat Sheet is a comprehensive quick guide for learning Git concepts, from very basic to advanced levels. By this Git Cheat Sheet, our aim is to provide a handy reference tool for both beginners and...

10 min read

NumPy Cheat Sheet: Beginner to Advanced (PDF)

NumPy stands for Numerical Python. It is one of the most important foundational packages for numerical computing & data analysis in Python. Most computational packages providing scientific functionality use...

15+ min read

Linux Commands Cheat Sheet

Linux, often associated with being a complex operating system primarily used by developers, may not necessarily fit that description entirely. While it can initially appear challenging for beginners, once you...

13 min read

Pandas Cheat Sheet for Data Science in Python

Pandas is a powerful and versatile library that allows you to work with data in Python. It offers a range of features and functions that make data analysis fast, easy, and efficient. Whether you are a data scientist,...

15+ min read

Java Cheat Sheet

Java is a programming language and platform that has been widely used since its development by James Gosling in 1991. It follows the Object-oriented Programming concept and can run programs written on a...

15+ min read

C++ STL Cheat Sheet

The C++ STL Cheat Sheet provides short and concise notes on Standard Template Library (STL) in C++. Designed for programmers that want to quickly go through key STL concepts, the STL cheatsheet covers...

15+ min read

Docker Cheat Sheet : Complete Guide (2024)

Docker is a very popular tool introduced to make it easier for developers to create, deploy, and run applications using containers. A container is a utility provided by Docker to package and run an applicatio...

11 min read

C++ Cheatsheet

This is a C++ programming cheat sheet. It is useful for beginners and intermediates looking to learn or revise the concepts of C++ programming. While learning a new language, it feels annoying to switch...

15+ min read

C Cheat Sheet

This C Cheat Sheet provides an overview of both basic and advanced concepts of the C language. Whether you're a beginner or an experienced programmer, this cheat sheet will help you revise and...

15+ min read

CCNA Cheatsheet

A CCNA certification proves you have the competencies needed to navigate an ever-changing IT landscape. CCNA exams cover network fundamentals, IP services, security fundamentals, automation, an...

15+ min read

Nmap Cheat Sheet

Nmap (Network Mapper) is a free and open-source network detection and security scanning utility. Many network and system administrators also find it useful for tasks such as network inventory, managing...

4 min read

Ethical Hacking Cheatsheet

Ethical hacking includes authorized attempts to gain unauthorized access to computer systems, applications, or data. Ethical hacking requires replicating the strategies and behaviors of malicious...

11 min read

Bootstrap Cheat Sheet - A Basic Guide to Bootstrap

Bootstrap is a free, open-source, potent CSS framework and toolkit used to create modern and responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for...

15+ min read

Computer Network - Cheat Sheet

A computer network is an interconnected computing device that can exchange data and share resources. These connected devices use a set of rules called communication protocols to transfer information over...

15+ min read

Angular Cheat Sheet - A Basic Guide to Angular

Angular is a client-side TypeScript-based, front-end web framework developed by the Angular Team at Google, that is mainly used to develop scalable single-page web applications(SPAs) for mobile & deskto...

15+ min read

jQuery Cheat Sheet – A Basic Guide to jQuery

What is jQuery?jQuery is an open-source, feature-rich JavaScript library, designed to simplify the HTML document traversal and manipulation, event handling, animation, and Ajax with an easy-to-use API that...

15+ min read

JavaScript Cheat Sheet - A Basic Guide to JavaScript

JavaScript is a lightweight, open, and cross-platform programming language. It is omnipresent in modern development and is used by programmers across the world to create dynamic and interactive web conten...

15+ min read

CSS Cheat Sheet - A Basic Guide to CSS

What is CSS? CSS i.e. Cascading Style Sheets is a stylesheet language used to describe the presentation of a document written in a markup language such as HTML, XML, etc. CSS enhances the look and feel of...

13 min read

HTML Cheat Sheet

HTML (HyperText Markup Language) serves as the foundational framework for web pages, structuring content like text, images, and videos. HTML forms the backbone of every web page, defining its structure,...

15 min read

Python Cheat sheet (2024)

Python is one of the most widely-used and popular programming languages, was developed by Guido van Rossum and released first in 1991. Python is a free and open-source language with a very simple and...

15+ min read



Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305



Advertise with us

Company

About Us
Legal
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

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
World GK

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

Interview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects