Day 1

1.What is Java?

History:

* Java is both a programming language and a platform.
* It is high-level, robust, object-oriented, and secure.
* Developed by Sun Microsystems (now part of Oracle) in 1995.
* James Gosling is known as the father of Java.
* Initially named **Oak**, but later changed to **Java** as "Oak" was already a registered trademark.

Features:

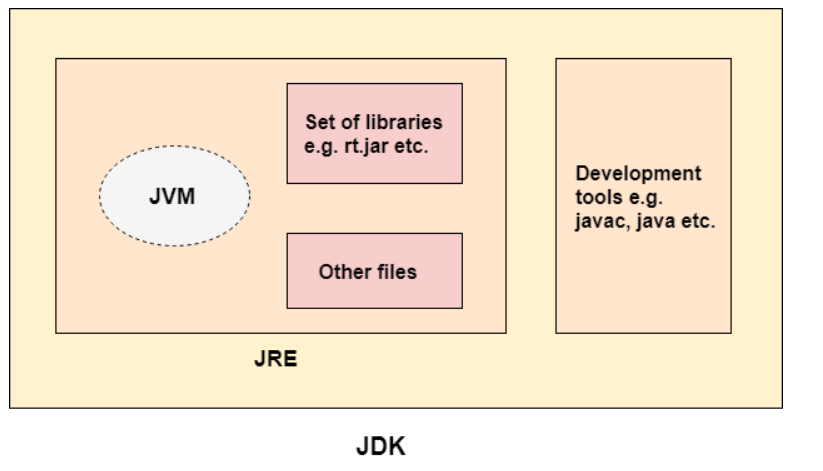
* **Simple** – Java is simple and easy to run, understand and learn, syntax is in simplest form, removes complex features (like pointers & operator overloading), and has automatic garbage collection.
* **Object-oriented** – Everything in java is object based, state and behaviours are organized.
* **Platform Independence**- Java doesn’t need any specific platform for run. There are two – software platforms and hardware platforms (java provide software platforms).
* **Secure** – Java doesn’t contain any exclusive pointers. Java runs inside a virtual machine so it will provide security
* **Robust** – Java provide strong and powerful memory management, also provide automated garbage collection process. Handles exceptions and runtime errors effectively.
* **Portable** – Java bytecode can run on any platform without requiring reimplementation.
* **High Performance** – Uses Just-In-Time (JIT) compilation for optimized execution, faster because byte code is close to native code.
* **Multi-threading** – Supports **concurrent execution** for better performance, main advantage- it shares a common memory.

2. JDK, JRE, and JVM

JDK (Java Development Kit) is a software development kit that includes JRE (Java Runtime Environment), compilers, and debugging tools needed to develop, compile, and run Java applications.

JRE (Java Runtime Environment) is a software package that provides the libraries and JVM (Java Virtual Machine) needed to run Java applications but does not include development tools like a compiler or debugger.

JVM (Java Virtual Machine) is an abstract machine that provides a runtime environment to execute Java bytecode. It is called a virtual machine because it doesn’t physically exist. JVM enables platform independence but is itself platform-dependent, as its configuration varies across operating systems. It can also run programs written in other languages compiled to Java bytecode. JVM has three notions: Specification, Implementation, and Instance.



Object and Class

* Object – It is an entity. And java objects have State (Datatypes) and Behaviour (Methods).
* Class – Blue print o objects or collection of objects.

Eg : class Sample (Sample is my class name and the first letter should be capital)

Syntax of main() Method

Public static void main(String args[])

Public: access specifier

Static: keyword, class related main method

Void: Return type

Main: Method name

args[]: array of string type

class Sample

{

Public static void main(String args[])

{

System.out.println(“hello world”)

}

}

**6.Errors**

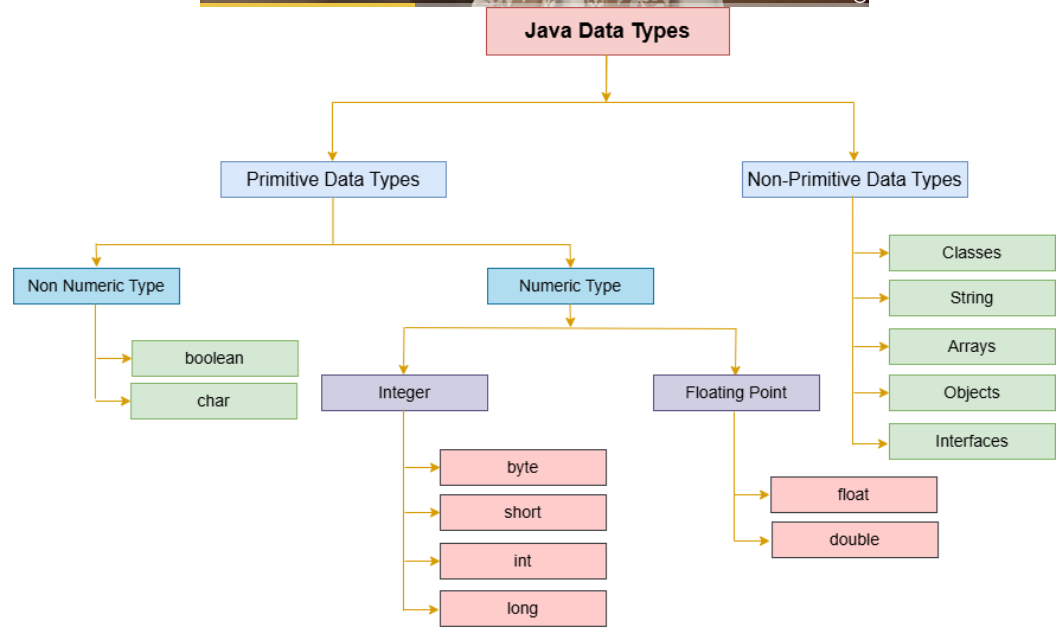
There are 2 types of errors: **Compile time error and Run time error.**

Compile time error means it deals with syntax mistakes .

Run time error means it deals with logical mistakes, eg : division by zero (Arithmetic error).

**7.Data Types**

There are two types of data types: **Primitive and Non primitive.**



Primitive Data Type: Numeric and Non Numeric

Numeric Data Type: 1. Integer – byte, short, int and long. (variable declaration and variable initialize, default value is 0)

2. Floating points – double (large decimal value) and float (small decimal value 0.0f).

Non-numeric: 1. Character (char c=’A’;)

2. Boolean (default value of Boolean is false)

Non Primitive Data Type: Strings, arrays and user defined classes

(String is an inbuild function, String s=”abc”;)

**8. Variables –** Variable is a name of reserved area allocated in memory. (To hold the value in a container)

There are 3 types

1. Local – declare inside a method, only access inside this method, local variable must be initialized
2. Static – declare inside the class and outside the method with the use of static keyword, the memory in the static variable is created only at once, and memory allocation occurs when the class is loading
3. Instance - declare inside the class and outside the method, also called non static variable, full access, and separate memory location for each and every instance is created

Class Sample

{

Static int a=10; // static var

Float b=34.3f; //instance var

Public static void main (String args[])

{

String name=”anu” ; //local var

System.out.println(“local var :”+name);

System.out.println(“static var :”+a);

System.out.println(“instance var :”+b); // it will not work because it is inside main method and it is non static method .

**9. Command Line Arguments**

The java command line argument is an argument that is passed at the run time of java program.

* String s = args[index];
* int a = Integer.parseInt(args[index]);
* float f = Float.parseFloat(args[index]);
* double d = Double.parseDouble(args[index]);
* char c=args[index].charAt(index);

**10. Operators**

Unary operators

* Unary operators (++,--)

Binary Operator

* Arithmetic operators
* Relational operators
* Logical operators
* Bitwise operators
* Assignment operators

Ternary operators – Ternary or conditional operator

**11. Conditional Statements - 3 types**

1. **Decision making statements**

**If statement**

* The if statement is the simplest form of a conditional.
* It executes a block of code only if the specified condition is true.

**Simple if statement**

If-else statement

* This extends the if statement.
* If the condition is true, the if block is executed; otherwise, the else block runs.

If-else-if ladder

* Used when there are multiple conditions to check.
* The first condition that evaluates to true will have its block executed.

Switch statements

* + - The switch statement is used when there are many possible values for a single variable.
    - It provides a cleaner alternative to multiple if-else conditions.

1. **Loop statements**

For loop

* + Used when the number of iterations is known.
  + It includes initialization, condition, and increment/decrement in one line.

while Loop

* Used when the number of iterations is not known in advance.
* Checks the condition before executing the loop body.

do-while Loop

* Similar to the while loop, but the condition is checked after executing the loop body.
* Ensures the loop body runs at least once.

1. **Jump statements**

**break**: Exits the loop prematurely.

**continue**: Skips the current iteration and moves to the next.

**return**: Exits the method entirely.

**15. Comments**

1. **Single-line Comment (//)** – Used for short explanation
2. **Multi-line Comment (/\* ... \*/)** – Used for longer or multi-line explanations.
3. **Documentation Comment (**/\*\* ... \*/**) -**Used to generate **JavaDocs** for classes, methods, and fields.

**16. Arrays**

An array in Java is a collection of elements of the same data type stored in contiguous memory locations. It is used to store multiple values in a single variable, instead of declaring separate variables for each value.

* Arrays are **fixed in size** (defined when created).
* They can hold **primitive types** (int, char, etc.) or **objects**.
* Each element is accessed using an **index**, starting from **0**.

**adv: code optimization and random access**

**dis adv: size limited.**

**two types of array**

**1. Single dimensional array**

**2. Multi dimensional array**

**1.Single dimensional array**

syntax

* Declaration

dataType[] arrayName;

dataType []arrayName;

dataType arrayName[];

* Instantiation

dataType arrayName[] = new dataType[size];

* Initialization

dataType arrayName[] = {value 1, value 2, ...... value n};

**2.Multi dimensional array**

syntax

* Declaration

dataType[][] arrayName;

dataType [][]arrayName;

dataType arrayName[][];

* Instatiation

dataType arrayName[][] = new dataType[rows][columns];

* Initialization

dataType[][] arrayName = {{value1, value2, value3}, {value4, value5, value6}};

**17 . Methods**

A method is a block of code that performs a specific task are executed when it is called from the main method.

adv: code reusability, easy modification

Two types of methods

**1. Static method**

**2. Instance method**

**static method** - parameterized static methods and non-parameterized static method, return type

use static keyword

main adv: we can call it without creating an object

it is invoked by using the className.

**className.methodName();**

main method syntax (para-static method)

access specifier-non access specifier-returntype-methodName(parameter list)

{

//stmt

}

main method syntax (nonpara-static method)

access specifier-non access specifier-returntype-methodName()

{

//stmt

}

**Return type**

It is used for returning value, when the execution of the block is completed.

Every method is declared with a return type such as int, float, double, string etc.

The void return type doesn't require return statement.

We can return only one value from a method.

return return-value;

syntax

returnType methodName(parameters) {

// method body

return value;

}

ex : public static int add(int a, int b) {

return a + b;

}

(int is a return type value)

**Instance method** - parameterized instance methods and non-parameterized instance method

return type

syntax

access specifier - returntype - methodName()

(here instance method is call by creating an objects)

object declaration

**className objectname = new className();**

**objectname.methodName(); //method call**

**Method overloading**

If a class has multiple methods with the same name but different parameters (type, number, or order).

diff ways to overload a method

1.By changing the number of arguments(parameter).

2.By changing the datatype.

How to create object

syntax

className objrefvar = new className();

(objrefvar - variable declaration , new - keyword that used to create an object or initialize an object in a memory)

**18. constructor**

A constructor is a special method used to initialize objects. It is called automatically when an object is created.

-It calls a default constructor, if there is no constructor available in the class.

**className objrefvar = new className()**;//call

Rules for creating constructors

1. Name must be same as className

2. No return type.

syntax

**access specifier constructorName()**

**{**

**//stmt**

**}**

**Types of constructor**

1.default constructor (non-parameterized constructor)

2.parameterized constructor

Syntax of Default constructor

access modifier className()

{

}

Syntax of Parameterized constructor

access modifier className(arg/parameter list)

{

}

**19. Packages**

A package in Java is a namespace that groups related classes and interfaces together. It helps in organizing code, avoiding name conflicts, and improving reusability and maintainability.

Types of Packages in Java

* **Built-in Packages**: Java provides several pre-defined packages, such as:

java.lang (default package, contains core classes like String, Math, System)

java.util (contains utility classes like ArrayList, HashMap, Collections)

java.io (provides classes for input and output operations)

java.net (supports networking operations)

* **User-defined Packages**: Developers can create their own packages to organize code.

**20. This Keyword**

This keyword is a reference variable that refers to the current class objects

used to refer instance variable of current class: this.instance variablename;.

used to refer current class methods this.methodName();.

used to refer current class constructors: this();

**21. String**

String is a sequence of characters. it is used for storing collection of characters. It is not a primitive data type.

inbuild object in java.

Strings are immutable (cannot be changed once created).

Two ways to create string object

1. By new keyword

2. By String literal

* **By String literal**

String s = "java";

String s1 = "java";

(the usage of string literals thus makes java more memory efficient)

* **By new keyword**

String a = new String("java");

String a1 = new String("java");

{JVM will create a new String object in the normal heap area

even if the same String object is present.}

**Methods of String class**

1. length()

→ Returns the number of characters in the string.

Example: "Java".length() → 4

2. charAt()

→ Returns the character at the specified index.

Example: "Java".charAt(1) → 'a'

3. valueOf()

converts various data types (int, float, double, boolean, char, etc.) into their String representation.

Example: int a = 10;

String.valueOf(a)

4. equals()

it compares content and case of the string . Compares two strings for equality (case-sensitive).

Example: "Java".equals("java") → false

5. equalsIgnoreCase()

Compares strings ignoring case.it only returns the Boolean functions

Example: "Java".equalsIgnoreCase("java") → true

6. isEmpty()

Checks if the string is empty.

Example: "".isEmpty() → true

7. concat()

The concat() method in Java is used to join two strings.

8.toUpperCase()

→ Converts all characters to uppercase.

Example: "java".toUpperCase() → "JAVA"

9.toLowerCase()

→ Converts all characters to lowercase.

Example: "JAVA".toLowerCase() → "java"

10.trim()

→ Removes leading and trailing spaces.

Example: " Hello ".trim() → "Hello"

11.replace(char oldChar, char newChar)

→ Replaces all occurrences of a character.

Example: "java".replace('a', 'o') → "jovo"

12.indexOf(char)

→ Returns the index of the first occurrence of the character.

Example: "hello".indexOf('l') → 2

**creating a string object - java.lang.String class**

we can create strings in java by using these three classes - String class, String Buffer class and String builder class.

* **String class**

Once string object is created its data or state can't be changed but a new string object is created.

**StringBuffer class and stringBuilder class**

Both this class re used to create mutable(modifiable) string.

StringBuffer is synchronized , ie. thread safe.

StringBuffer buffer = new StringBuffer("hello");

StringBuilder is non-synchronized , ie. not thread safe.

StringBuilder builder = new StringBuilder("hello");

* **StringBuffer class**

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed**.**

**Important Methods of StringBuffer Class**

append(String s) → Appends the specified string (overloaded for different data types).

insert(int offset, String s) → Inserts the specified string at the given position.

replace(int startIndex, int endIndex, String str) → Replaces part of the string between startIndex and endIndex.

delete(int startIndex, int endIndex) → Deletes part of the string between startIndex and endIndex.

reverse() → Reverses the string.

capacity() → Returns the current buffer capacity.

ensureCapacity(int minimumCapacity) → Ensures the buffer has at least the given minimum capacity.

charAt(int index) → Returns the character at the specified index.

length() → Returns the length (total number of characters) in the StringBuffer.

substring(int beginIndex) → Returns the substring from beginIndex.

substring(int beginIndex, int endIndex) → Returns the substring between beginIndex and endIndex.

* **String Builder class**

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

**Methods of StringBuilder Class**

append(String s) → Appends the specified string (overloaded for different data types).

insert(int offset, String s) → Inserts the specified string at the given position.

replace(int startIndex, int endIndex, String str) → Replaces part of the string between startIndex and endIndex.

delete(int startIndex, int endIndex) → Deletes part of the string between startIndex and endIndex.

reverse() → Reverses the string.

charAt(int index) → Returns the character at the specified index.

length() → Returns the length (total number of characters) in the StringBuilder.

**22. Inheritance**

Inheritance in Java allows a class (subclass) to inherit properties and behaviors from another class (superclass). This promotes code reuse and establishes a parent-child (IS-A) relationship between classes.

Use of inheritance

For method overriding.

For code reusability.

Syntax

class Subclass-name **extends** Superclass-name

{

*//methods and fields*

}

**Types of inheritance:**

1. Single Inheritance – A subclass inherits from a single superclass.
2. Multilevel Inheritance – A subclass inherits from another subclass, forming a chain.
3. Hierarchical Inheritance – Multiple subclasses inherit from a single superclass.
4. Multiple Inheritance (via Interfaces) – A class implements multiple interfaces to achieve multiple inheritance.
5. Hybrid Inheritance – A combination of two or more types of inheritance (achieved using interfaces in Java).

**Advantages of Inheritance:**

* Code Reusability: Avoids redundant code by reusing superclass members.
* Hierarchical Organization: Creates structured and maintainable class hierarchies.
* Polymorphism: Allows method overriding for dynamic behavior.
* Easier Maintenance: Changes in the superclass reflect in subclasses automatically.

**23. Access modifiers**

Access modifiers control the visibility and accessibility of classes, methods, and variables. Java has four types:

1. **Public** – Accessible from anywhere in the program.
2. **Private** – Accessible only within the same class.
3. **Protected** – Accessible within the same package and by subclasses.
4. **Default (No Modifier)** – Accessible only within the same package.

**25. Super Keyword**

The super keyword in Java is a reference variable that is used to refer to immediate parent class objects. Program:

Uses :

1. super can be used to refer to the immediate parent class instance variable.
2. super can be used to invoke the immediate parent class method.
3. super() can be used to invoke the immediate parent class constructor.

**This keyword / Super keyword**

| **Feature** | **this Keyword** | **super Keyword** |
| --- | --- | --- |
| **Definition** | Refers to the current class instance | Refers to the parent class instance |
| **Usage** | Used to access current class methods, variables, and constructors | Used to access parent class methods, variables, and constructors |
| **Method Call** | Calls current class method: this.methodName(); | Calls parent class method: super.methodName(); |
| **Constructor Call** | Calls another constructor of the same class: this(); | Calls parent class constructor: super(); |
| **Variable Access** | Used when the local variable and instance variable have the same name: this.variableName; | Used to access parent class variables if overridden: super.variableName; |
| **Inheritance** | Does not work across inheritance | Works across inheritance to refer to the parent class |

**26. Aggregation(HAS- A)**

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship**.**

Aggregation is a relationship between two classes where one class contains a reference to another class. It represents a "has-a" relationship rather than an "is-a" relationship.

Use: Code Reusability

**27. Polymorphism**

Polymorphism in Java is the ability of an object to take on many forms. It allows methods to perform a single action in different ways.

Advantages of Polymorphism:

1. Code Reusability: Methods in subclasses can override superclass methods, promoting reusable and consistent interfaces.
2. Flexibility and Extensibility: Subclasses can have their own method implementations without altering the existing code.
3. Dynamic Method Invocation: The method to be called is determined at runtime, based on the object type.
4. Interface Implementation: Multiple classes can implement the same interface, allowing objects to be treated interchangeably.
5. Method Overloading: Multiple methods with the same name but different parameters can coexist, improving code readability.
6. Reduced Code Complexity: Supports modular and hierarchical class structures, making complex systems easier to manage.

**Method Overloading**

Method overloading in Java allows a class to have multiple methods with the same name but different parameter lists (different number or types of parameters). It improves code readability and usability.

Real-Life Example:  
Think of a calculator that can add two, three, or even four numbers. Instead of creating separate methods like addTwoNumbers(), addThreeNumbers(), etc., we use method overloading to define a single add() method with different parameter counts.

Example:

class Calculator {

void add(int a, int b) {

System.out.println(a + b);

}

void add(int a, int b, int c) {

System.out.println(a + b + c);

}

}

Here, both methods are named add(), but they differ in the number of parameters.

**Method Overriding**

Method overriding in Java occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. The method in the child class must have the same name, return type, and parameters as the one in the parent class. It enables runtime polymorphism.

**Real-Life Example:**  
Imagine a **vehicle** system where every vehicle has a method to **start**. A **Car** and a **Bike** will both have a start() method, but the way they start is different. Overriding allows each subclass to provide its own version of start().

**Example:**

class Vehicle {

void start() {

System.out.println("Vehicle starts");

}

}

class Car extends Vehicle {

void start() {

System.out.println("Car starts with a key");

}

}

class Bike extends Vehicle {

void start() {

System.out.println("Bike starts with a kick");

}

}

Top of Form

Bottom of Form

28. **Encapsulation**

Encapsulation in Java is a process of wrapping code and data together into a single unit. A capsule is the perfect example of encapsulation because a capsule is a mixture of several medicines.

**Benefits of Encapsulation:**

* **Data hiding**
* **Better control** over fields
* **Increased security**
* **Easier to maintain**

**29. Final Keyword**

The **final keyword in Java** is used to restrict the user. It is also known as a non-access modifier. We can use the final keyword with:

1. Variable
2. Method
3. Class

1) Java final variable

When a variable is declared as final, it is known as a final variable. Its value cannot be changed once initialized. It behaves like a constant.

**Syntax:**

1. **final** datatype VARIABLE\_NAME=VALUE;

2) Java final Method

A method declared as final is known as a final method. Subclasses cannot override the final method.

**Syntax:**

1. **final** **void** paint() {

3) Java final Class

A class declared with the final keyword is known as a final class. Note that the final class cannot be inherited.

**Syntax:**

1. **final** **class** Square {
2. *//statement*
3. }

**30. Abstraction**

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Abstract class

An abstract class in Java acts as a partially implemented class that itself cannot be instantiated. It exists only for subclassing purposes, and provides a template for its subcategories to follow. Abstract classes can have implementations with abstract methods. Abstract methods are declared to have no body, leaving their implementation to subclasses.

Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have constructors and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

Syntax

public **abstract** **class** Shape {

**public** **abstract** **double** area();

**public** **void** display() {

        System.out.println("This is a shape.");

    }

}

Interface

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a* mechanism to achieve [abstraction](https://www.tpointtech.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not a method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.tpointtech.com/inheritance-in-java).

Implements is used to extend

In other words, we can say that interfaces can have abstract methods and variables. It cannot have a method body.

* Java Interface also represents the IS-A relationship.
* It cannot be instantiated just like the abstract class.
* Since Java 8, we can have default and static methods in an interface.
* Since Java 9, we can have private methods in an interface.

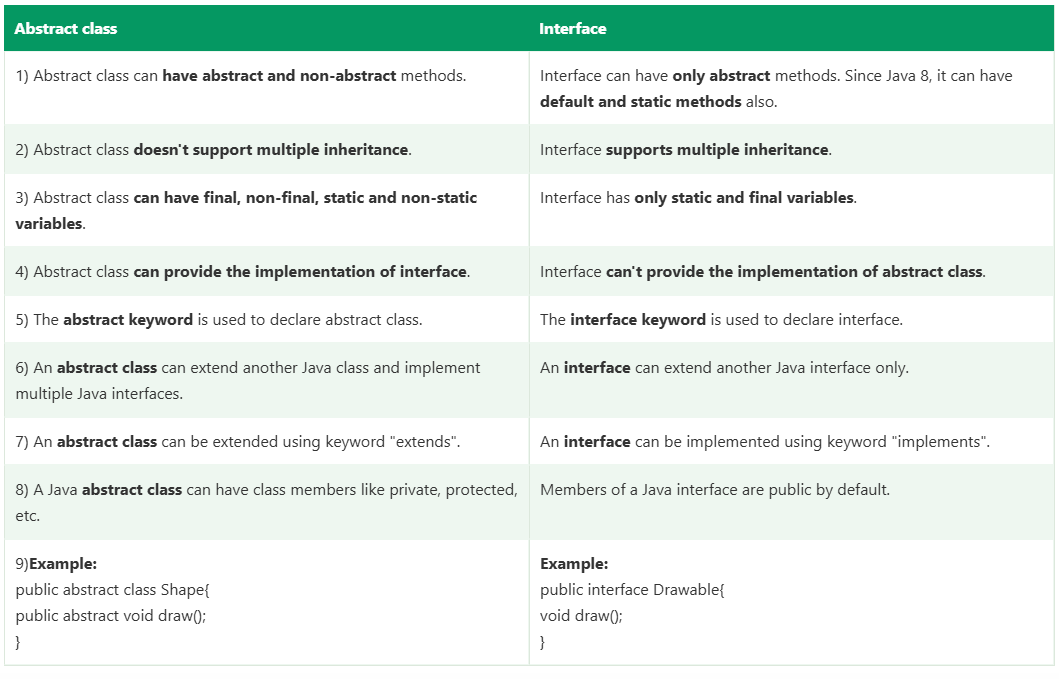
Syntax

1. interface <interface\_name>{
3. *// declare constant fields*
4. *// declare methods that abstract*
5. *// by default.*
6. }

**Declaring Interface**

1. **interface** Animal {
2. **void** eat();
3. **void** sleep();
4. }

Abstract v/s Interface



**31. Exception Handling**

What is Exception in Java?

In Java, an exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. These exceptions can occur for various reasons, such as invalid user input, file not found, or division by zero. When an exception occurs, it is typically represented by an object of a subclass of the java.lang.Exception class.

What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Adv: **to maintain the normal flow of the application**

Types of Java Exceptions

In Java, exceptions are categorized into two main types: checked exceptions and unchecked exceptions. Additionally, there is a third category known as errors.

1.Checked Exception

2.Unchecked Exception

**Checked Exceptions**

* Handled at **compile-time**.
* Must be declared with throws or handled with try-catch.
* Examples:
  + IOException – file read/write errors
  + SQLException – database access errors
  + ParseException – format/parse errors
  + CheckedException
  + NumberFormat
  + IndexOutOfBound
  + ClassNotFoundException – class not found in classpath

**Unchecked Exceptions (Runtime)**

* Handled at **runtime**, not checked by the compiler.
* Optional to catch or declare.
* Occur due to **logic/programming errors**.
* Examples:
  + NullPointerException – null object access
  + ArrayIndexOutOfBoundsException – invalid array index
  + ArithmeticException – divide by zero
  + IllegalArgumentException – bad method arguments

**A screenshot of a computer

AI-generated content may be incorrect.**

**Java Custom Exception**

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

**31. Debugging**

Debugging is the process of identifying and fixing errors in a program. It involves:

1. Using a Debugger – Tools like Eclipse, IntelliJ, or JDB help track program execution.
2. Breakpoints – Pause execution at specific lines to inspect variables.
3. Step Execution – Run code line by line to find issues.
4. Print Statements – Use System.out.println() to check variable values.
5. Exception Handling – Use try-catch blocks to handle errors gracefully.

**32. Collections in Java**

Collections are data structures that hold groups of objects. Java provides a powerful **Collection Framework** to manage data efficiently.

Interfaces used in collection

All interfaces have implemented classes

1. List – classes : Linked list and Array list
2. Set – Implemented classes: Hash set and Tree set
3. Queue
4. Deque

Every interface has its own implemented classes.

Difference between list and set

List

Ordered , allow duplicate

Set

Unordered, no duplicates ( shuffled)

Hashset, Treeset

Two types of collection

1. Generic collections
2. Non generic collection

**Generic Collections**

* A Generic Collection defines the type of objects it can hold using **type parameters (like <String>, <Integer> etc.)**.
* **Introduced in Java 5** to enable **type safety** and eliminate **type casting**.
* **Compile-time type checking** ensures fewer runtime errors.
* Improves **code readability** and **maintainability**.

**Example Concept:**  
A List<String> can only store String objects. Trying to add other types will result in a **compile-time error**.

List<String> s=new ArrayList<String>();

Set<String> s =new HashTree<String>();

List<Integer> s=new ArrayList<Integer>();

**Non-Generic Collections**

* Non-Generic Collections do **not define a type** and can store **any kind of objects**.
* Based on the **Object** class, so all data types can be stored.
* **Type casting is needed** when retrieving data.
* Higher risk of **runtime exceptions** like ClassCastException.

**Example Concept:**  
A non-generic List can store both String and Integer, but you must cast the object while retrieving it, which can cause errors.

List s=new ArrayList();

List Methods

1. Add() - add an elements to the list.
2. Get() – Returns the elements at the specified position.
3. Set() – Replace the elements at the specified position.
4. indexOf()- Returns the index of the 1st occurrence of the elements for repeated elements.
5. LastIndexOf() - Returns the last index of the elements for repeated elements.
6. Remove() – Removes the elements at the specified position.
7. Contains()- checks if the list contains the specific elements
8. isEmpty()- Checks if the list is Empty.
9. Size()- Returns the no of elements in the list.

Methods of Set

1. Add() – Add specified element to the set
2. addAll() – Adds all elements from another collection to the set.
3. remove() – Removes the specified element.
4. removeAll() – Remove all elements from the set that are contained in another collection.
5. contains() – Checks if a specific element exists in the set.
6. containsAll() – set contains all the elements in the specified collection.
7. isEmpty() – Checks if the set is empty.
8. size() – Returns the number of elements in the set.
9. clear() – Removes all elements from the set.
10. iterator() – Returns an iterator to loop through the set.
    * hasNext() – check next elements contains or not.
    * Next() – to print element
    * Remove() – remove last index elements in the set.

