📘**Unit 3 – INTERFACES AND API**

🧠INTERFACES – Overview

😾INTERFACES

👎Introduction

- An **interface** in Java is a reference type, similar to a class, that can contain **only constants, method signatures, default methods, static methods, and nested types**.
- Interfaces help achieve **multiple inheritance** and define a **contract** for what a class can do, without dictating how.

👎Defining Interfaces

```
interface MyInterface {
    void show(); // abstract method
}
```

- All methods in interfaces are **implicitly public and abstract** unless default/static.

👎Implementing Interfaces

- A class uses the `implements` keyword to define the behavior of the interface methods.

```
class Demo implements MyInterface {
    public void show() {
        System.out.println("Implemented show()");
    }
}
```

👎Extending Interfaces

- Interfaces can extend other interfaces using `extends`, allowing hierarchical abstraction.

```
interface A {
    void methodA();
}
interface B extends A {
    void methodB();
}
```

👎Accessing Interface Variables

- All variables in interfaces are implicitly `public`, `static`, and `final`.

```
interface Constants {
    int VALUE = 10;
}
System.out.println(Constants.VALUE);
```

👇PACKAGES

👎Introduction

- A **package** is a namespace that organizes classes and interfaces.
- Helps avoid naming conflicts and controls access.

👎Types of Packages

1. **Built-in Packages** – Provided by Java (e.g., `java.util`, `java.io`, `java.lang`)
2. **User-defined Packages** – Created by developers

👎Using System Packages

```
import java.util.Scanner;
```

- Import specific class or use `*` to import all classes from a package.

👎Naming Conventions

- Package names are written in **lowercase**.
- Often follow reverse domain name convention (e.g., `com.openai.project`)

👎Creating and Using Packages

```
// File: Demo.java
package mypack;
class Demo {
    void display() {
        System.out.println("Demo class");
    }
}
```

- Compile: `javac -d . Demo.java`
- Access: `import mypack.Demo;`

👎Hiding Classes

- Use **default (no access modifier)** to limit visibility to within the same package.

👹ARRAYS, STRINGS, AND VECTORS

😢Arrays

- Collection of **fixed-size**, homogeneous data.

```
int[] arr = new int[5];
arr[0] = 10;
```

- Access using index (0-based)

😢Strings

- Represented by `String` class – **immutable**
- Declaration:

```
String s1 = "Hello";
String s2 = new String("World");
```

- Common methods:
- `.length()`, `.charAt()`, `.substring()`, `.equals()`, `.compareTo()`

😢Vectors

- Part of `java.util` package
- Growable array, **thread-safe**

```
import java.util.Vector;
Vector<Integer> v = new Vector<>();
v.add(10);
v.add(20);
```

- Methods: `add()`, `get()`, `remove()`, `size()`

😝STRING HANDLING

😥String Immutability

- Once created, a `String` object **cannot be changed**.

😥StringBuffer and StringBuilder

- **StringBuffer** – mutable and thread-safe
- **StringBuilder** – mutable but not thread-safe (faster)

```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
```

😥Common Operations

- Concatenation: `+`, `.concat()`
- Comparison: `.equals()`, `.compareTo()`
- Searching: `.indexOf()`, `.contains()`
- Modification: `.replace()`, `.toUpperCase()`, `.trim()`

😄WRAPPER CLASSES

👎Introduction

- Provides object representation for primitive data types.
- Enables primitive types to be used in **collections** and with **generics**.

Primitive	Wrapper
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

👎Auto-boxing

- Automatic conversion from primitive → wrapper

```
int a = 5;
Integer obj = a; // auto-boxing
```

👎Unboxing

- Wrapper → primitive

```
Integer obj = 10;
int b = obj; // unboxing
```

👎Useful Methods

- `Integer.parseInt("123")` – String to int
- `Character.isDigit('5')` – check character type
- `Boolean.parseBoolean("true")`

🔧Understanding these topics equips you to handle real-world Java programming with efficiency and confidence. ⏳