

# Packages in Java

- Packages are used to *organize Java classes into logical groups*, much like folders on your computer.
- A package in Java is a namespace that *organizes a set of related classes* and interfaces.
- It is used to avoid name conflicts and to control access to classes, interfaces, and methods.

# How to Create and Use Packages

## Step 1: Create the Package and Class

```
1 // Save this as Circle.java in the directory com/example/shapes/
2 package com.example.shapes;
3
4 public class Circle {
5     private double radius;
6
7     public Circle(double radius) {
8         this.radius = radius;
9     }
10
11     public double area() {
12         return Math.PI * radius * radius;
13     }
14 }
```

Create a simple package named  
com.example.shapes


The package contains a class  
called Circle.

# How to Create and Use Packages

## Step 2: Use the Package and Class

```
1 // Save this as MainApp.java in the default package (no package declaration)
2 import com.example.shapes.Circle;
3
4 public class MainApp {
5     public static void main(String[] args) {
6         Circle circle = new Circle(5.0);
7         System.out.println("Area of the circle: " + circle.area());
8     }
9 }
```

Another class that uses the Circle class.

A blue line originates from the text 'Another class that uses the Circle class.' and branches into two arrows. One arrow points to the 'import com.example.shapes.Circle;' statement on line 2. The other arrow points to the 'new Circle(5.0)' instantiation on line 6.

# Understanding Interfaces

- An interface is like *a contract that a class can implement*. It *defines a set of methods that the implementing class must provide*.
- Interfaces *help achieve abstraction*, provide *a way to achieve multiple inheritance* (since a class can implement multiple interfaces), and promote loose coupling between classes.
- 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 *cannot have instance fields or constructors*. They are abstract by nature.

# How to Create and Implement Interfaces

## Step 1: Create the Interface

```
1 // Save this as Shape.java in the package com/example/shapes/  
2 package com.example.shapes;  
3  
4 public interface Shape {  
5     double area();  
6     double perimeter();  
7 }
```

# How to Create and Implement Interfaces

## Step 2: Implement the Interface in a Class

```
1 // Save this as Rectangle.java in the package com/example/shapes/  
2 package com.example.shapes;  
3  
4 public class Rectangle implements Shape {  
5     private double length;  
6     private double width;  
7  
8     public Rectangle(double length, double width) {  
9         this.length = length;  
10        this.width = width;  
11    }  
12  
13    @Override  
14    public double area() {  
15        return length * width;  
16    }  
17  
18    @Override  
19    public double perimeter() {  
20        return 2 * (length + width);  
21    }  
22 }
```

# How to Create and Implement Interfaces

## Step 3: Use the Implementation in Another Class

```
1 // Save this as MainApp.java in the default package (no package declaration)
2 import com.example.shapes.Rectangle;
3 import com.example.shapes.Shape;
4
5 public class MainApp {
6     public static void main(String[] args) {
7         Shape rectangle = new Rectangle(4.0, 5.0);
8         System.out.println("Area of the rectangle: " + rectangle.area());
9         System.out.println("Perimeter of the rectangle: " + rectangle.perimeter());
10    }
11 }
```

# Differences between abstract classes and interfaces

Feature	Abstract Class	Interface
Method Implementation	Can have both abstract and concrete methods.	Only abstract methods (default and static methods allowed from Java 8).
Multiple Inheritance	A class can extend only one abstract class.	A class can implement multiple interfaces.
Variables	Can have instance and static variables with any access modifier	All variables are implicitly public, static, and final (constants)
Constructors	Can have constructors.	Cannot have constructors.
State	Can maintain state using instance variables.	Cannot maintain state; only constants allowed.
Use Case	Used to share code among closely related classes.	Used to define a contract that unrelated classes can implement.
Example	<pre>abstract class Animal {     abstract void makeSound(); // Abstract method     void sleep() { // Concrete method         System.out.println("Sleeping...");     } }  class Dog extends Animal {     void makeSound() {         System.out.println("Bark");     } }</pre>	<pre>interface Playable {     void play(); // Abstract method }  class Football implements Playable {     public void play() {         System.out.println("Playing football");     } }</pre>