الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

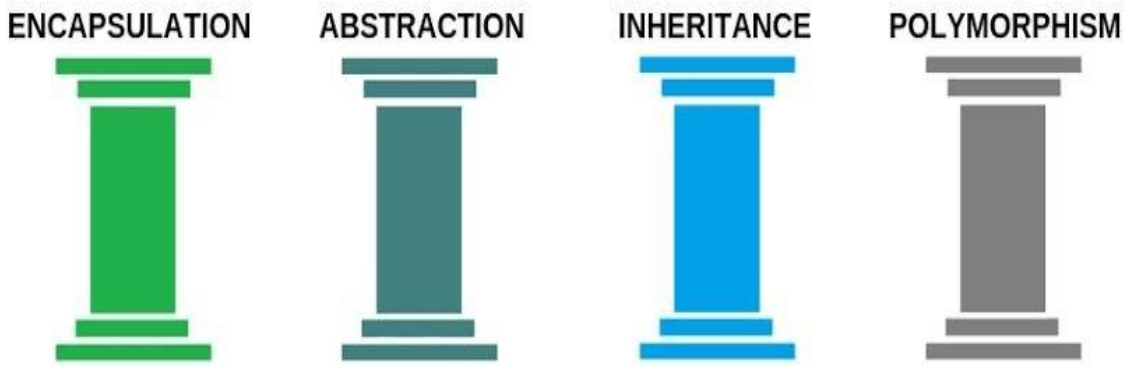
Programming 2

Week 10: abstraction

Eng. Raghad al-hossny

Object-oriented programming OOP pillars:

The four pillars of object-oriented programming (OOP) are abstraction, encapsulation, inheritance, and polymorphism. These principles help organize and manage complex code by using objects to interact with each other through a set of core ideas that promote code reuse, flexibility, and modularity.



Abstraction:

The ability to hide complex implementation details and show only the necessary features of an object. This simplifies the interaction with objects.

Abstraction in OOP:

Types of abstraction:

- **Data abstraction: data abstraction focuses on what data is used and what operations can be performed on it, while hiding how the data is implemented.**

- **Control abstraction: Control abstraction focuses on hiding complex logic and control flow inside methods, so the user does not need to know how something is done.**

```
class BankAccount {
    private double balance;
    public void deposit(double amount) {
        balance += amount;
    }
    public double getBalance() {
        return balance;
    }
}
```

```
public void processOrder() {
    validateOrder();
    calculatePrice();
    saveToDatabase();
    sendConfirmation();
}
```

Abstraction in OOP:

How to achieve abstraction?

1. **Abstract class: an abstract class is a class that can have both abstract and non-abstract methods, where abstract methods have only a signature without a body (implementation).**

```
abstract class Animal {
    abstract void makeSound();   // abstract method

    void sleep() {               // normal method
        System.out.println("Sleeping...");
    }
}
```

An abstract class lets you:
Provide shared behavior
Force subclasses to implement specific methods.

Abstraction in OOP:

How to achieve abstraction?

1. Abstract class

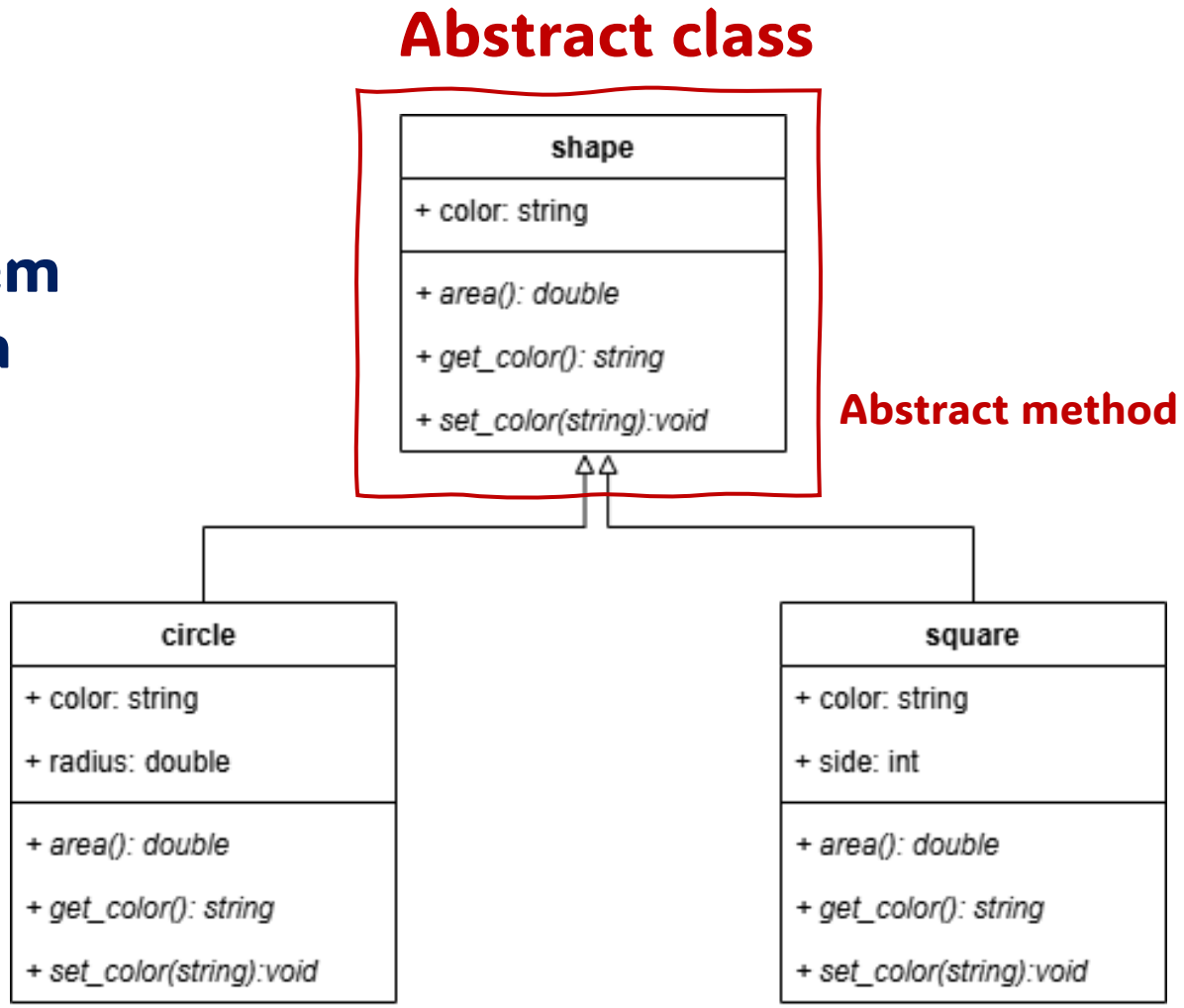
Example: design and implement a system that calculates the area of a circle and a square.

Abstraction in OOP:

How to achieve abstraction?

1. Abstract class

Example: design and implement a system that calculates the area of a circle and a square.



Abstraction in OOP:

How to achieve abstraction?

1. Abstract class

Example: design and implement a system that calculates the area of a circle and a square.

```java
public abstract class Shap {

    public String color;

    public abstract  double  area() ;
}
```

```java
public class Circle extends Shap {

    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double area() {
        return Math.PI * radius * radius;
    }

}
```

```java
public class Square extends Shap {

    private double side;

    Square(double side) {
        this.side = side;
    }

    double area() {
        return side * side;
    }

}
```

Abstraction in OOP:

How to achieve abstraction?

1. Abstract class

Example: design and implement a system that calculates the area of a circle and a square.

```
package shapes;


public class Shapes {


    public static void main(String[] args) {

        Square s1 = new Square(2);
        s1.color = "red";
        System.out.println(s1.color);
        System.out.println(s1.area());

    }

}
```

shapes.Shapes > main >

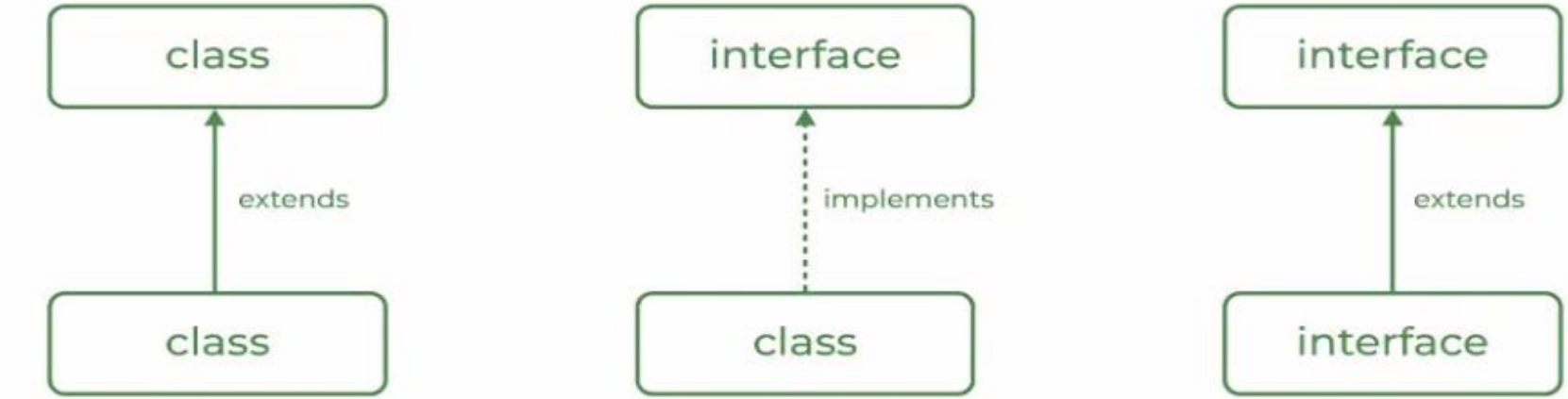
ut - shapes (run) × Terminal - **localhost**

```
run:
red
4.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Abstraction in OOP:

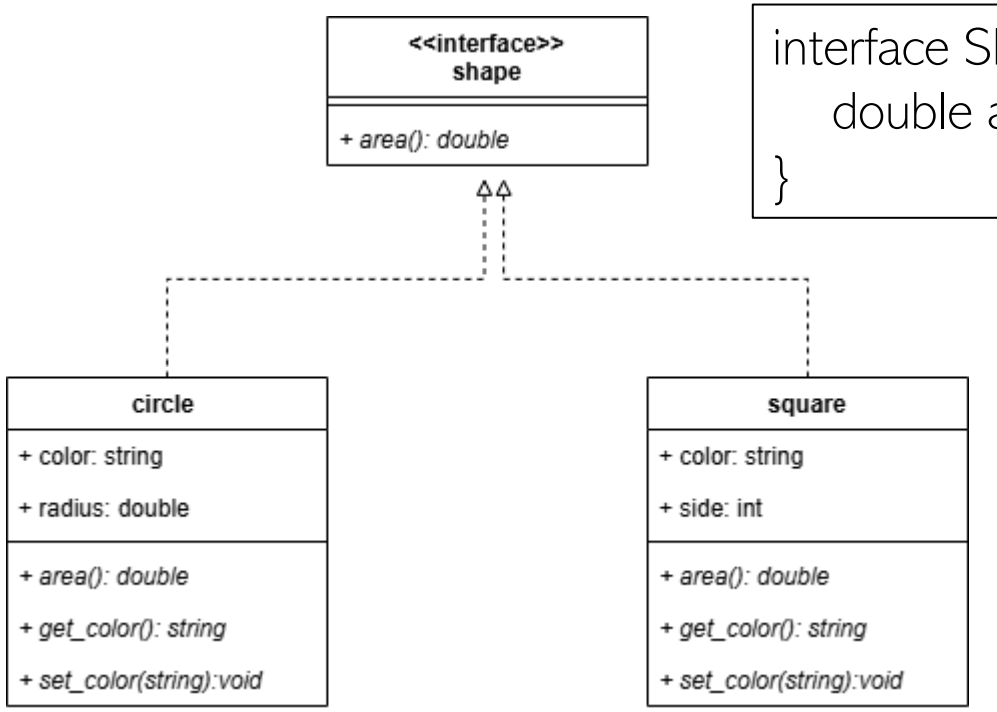
How to achieve abstraction?

2. interfaces: an interface is a contract that a class agrees to follow.
It contains method declarations only (abstract methods).
The purpose of an interface is to define what a class can do, not how it does it.

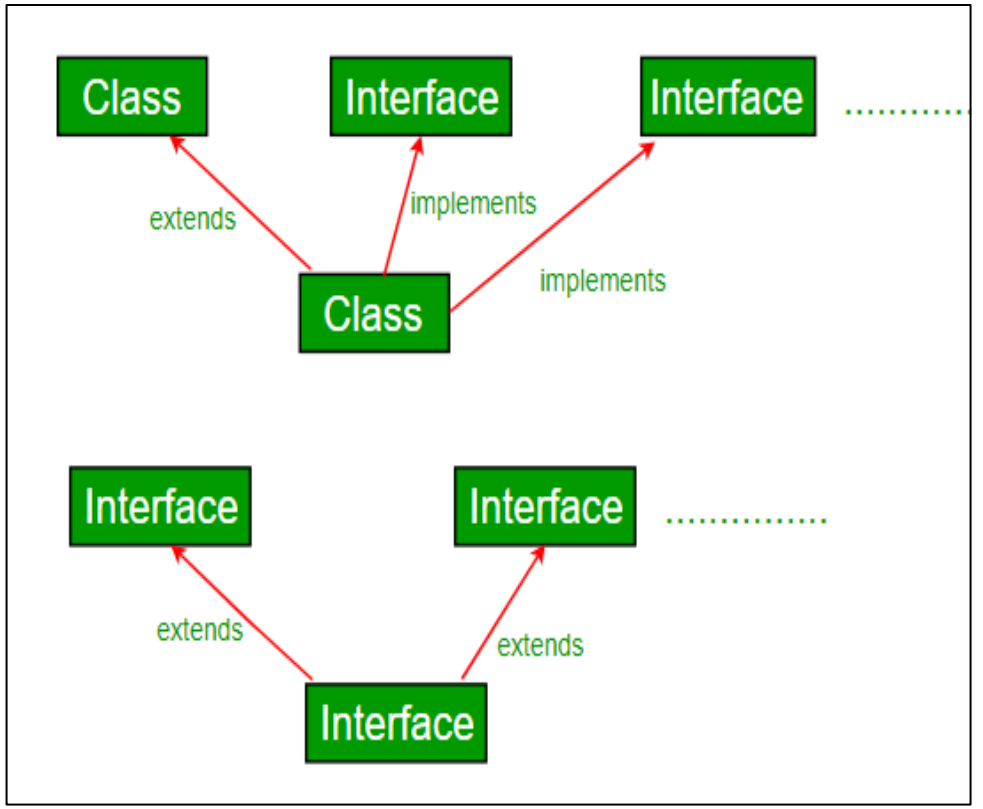


Abstraction in OOP:

2. interfaces: allow us to achieve the multiply inheritance.

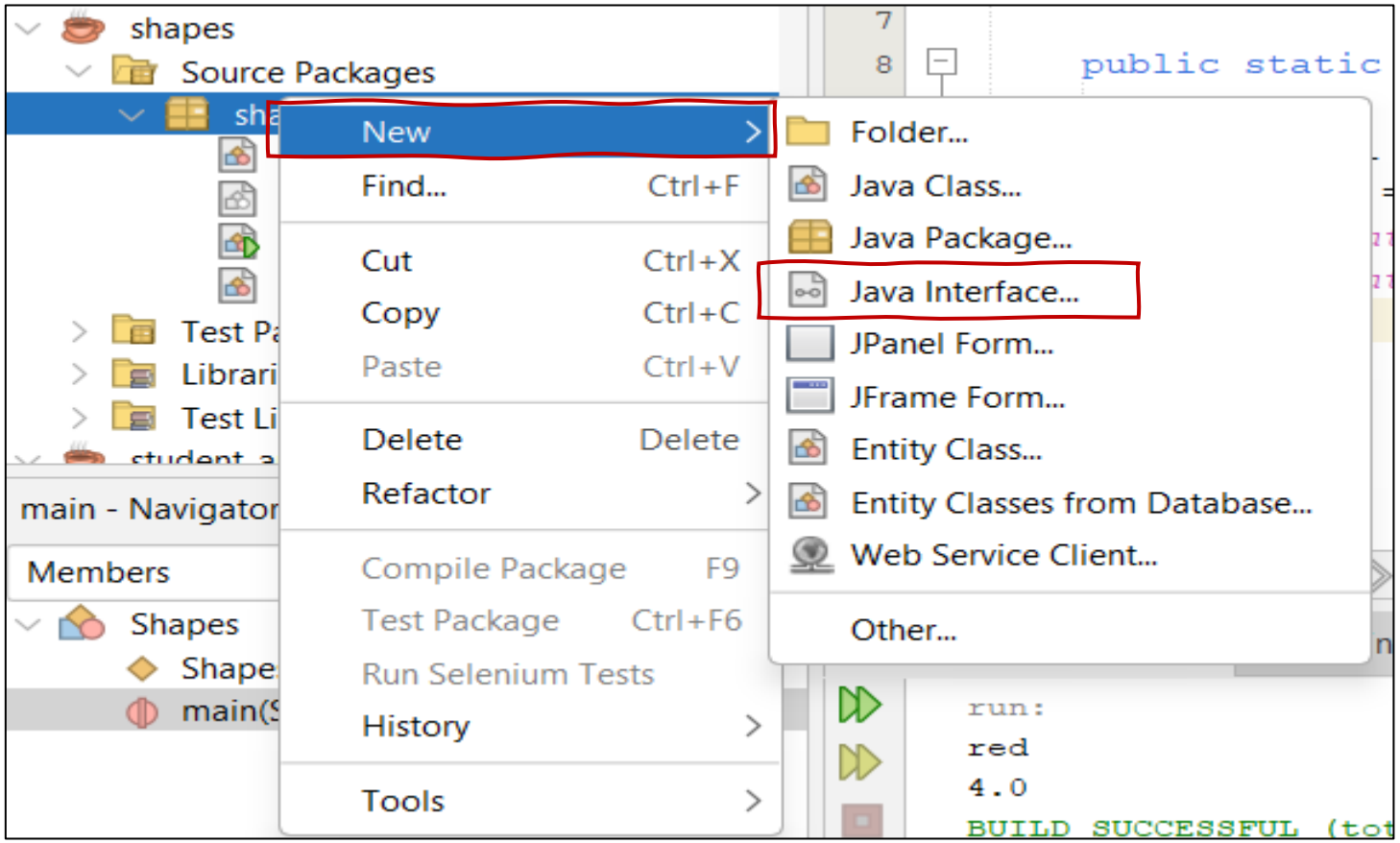



```
interface Shape {
    double area();
}
```



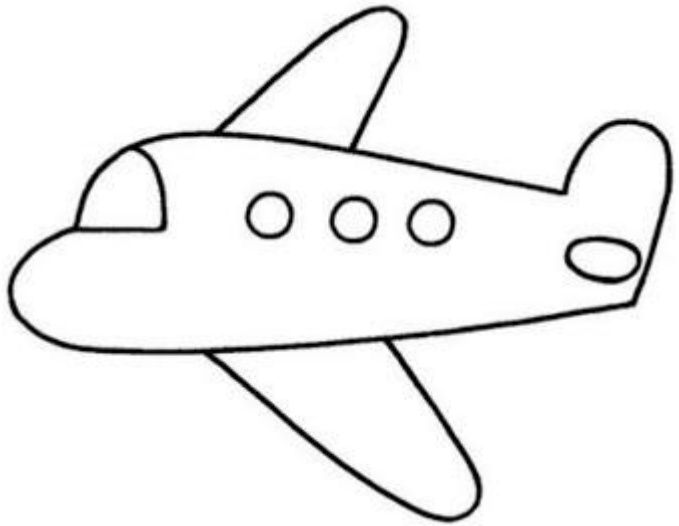
Abstraction in OOP:

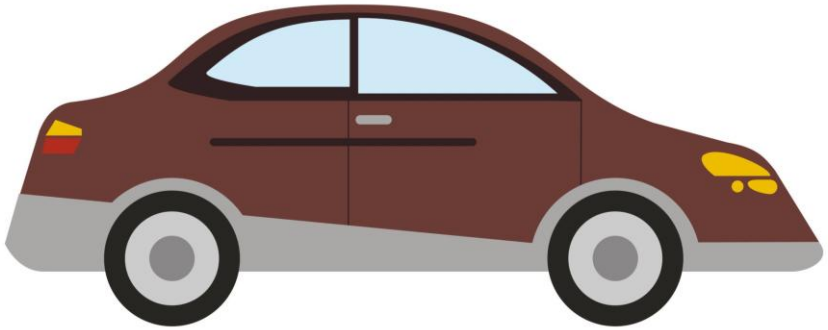
2. interfaces: how to define an interface?



Example: implement this system:

The Vehicle Management System is designed to model different types of vehicles in a generic and extensible way.
The system is based on the concept of abstraction, where a general vehicle is represented without specifying exact movement behavior.





Example: implement this system:

The Vehicle Management System is designed to model different types of vehicles in a generic and extensible way.
The system is based on the concept of abstraction, where a general vehicle is represented without specifying exact movement behavior.