CSC 125
Object Oriented Programming

Ch09_Objects and Classes

Dr. Fadi Alzhouri

What is an Object-oriented programming (OOP)?

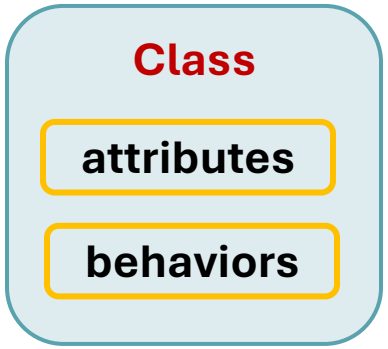
- A programming paradigm based on the concept of "objects", which can contain data and code.
- It is programming that favors using objects rather than mathematical concepts.
- An object represents a thing that can be clearly identified in the real world. For example, a car, bus, a motorcycle, a student, etc.
- OOP enables you to develop large-scale software effectively.
- It's a technology for developing reusable software.

OOP features

- Key features:
  - **Encapsulation**: Bundling data (attributes) and methods (behavior) that operate on the data into a single unit (class).
  - **Inheritance**: Mechanism to create a new class from an existing class, promoting code reuse.
  - **Polymorphism**: Ability to process objects differently based on their data type or class.
  - **Abstraction**: Hiding complex implementation details and showing only the essential features.

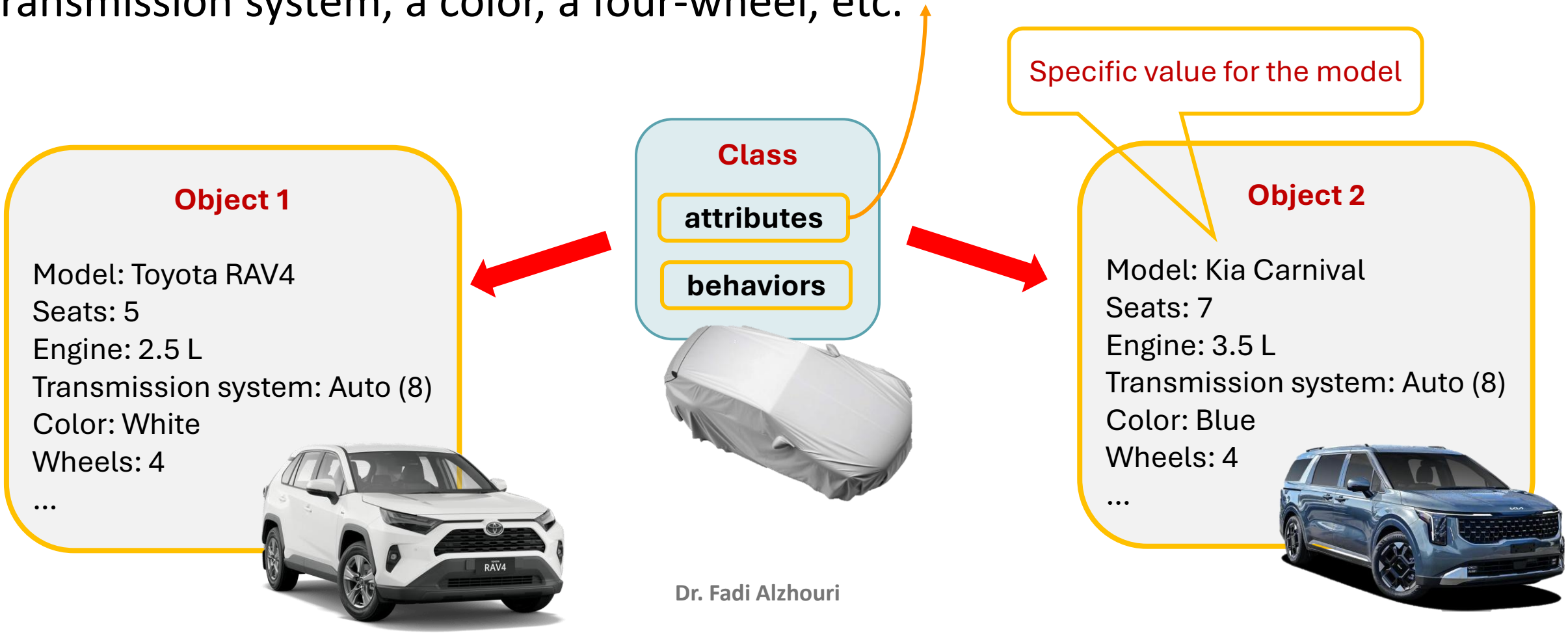
Classes & Objects

- Class: A blueprint or template defining attributes and methods (behavior) for creating objects.
- A class is a high level of abstraction that represents an umbrella of common objects (things) in the real world.
- A class is an abstract representation of the objects that have the same attributes and the same behaviors.
- A class is a construct that defines objects of the exact nature.
- Object: An instance of a class that contains actual values (specific data).



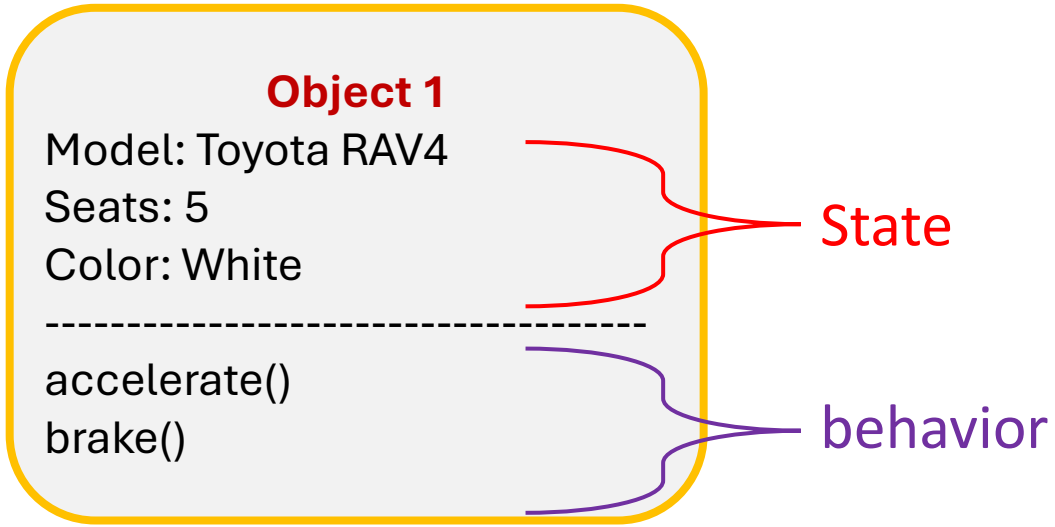
Classes & Objects (cont.)

- Example: if a car is a class, it has a model, number of seats, an engine, a transmission system, a color, a four-wheel, etc.

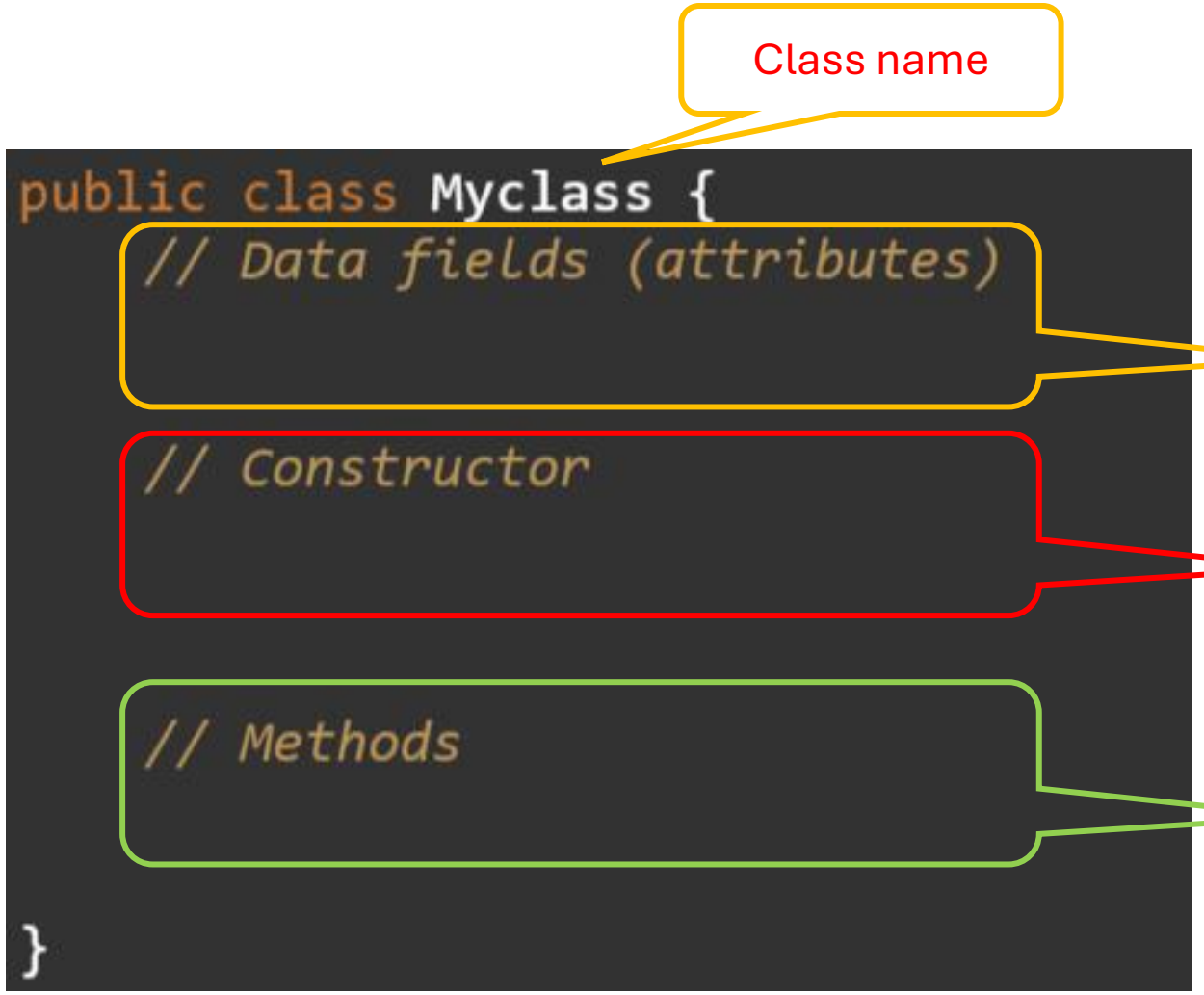


Classes & Objects (cont.)

- A class uses variables to define data fields (attributes) and methods to define behaviors.
- An object has a unique identity, state, and behaviors.
- The data fields (aka. properties or attributes) with their current values represent the state of the object.
- The behavior of an object is defined by a set of functions (methods).



Define a class for objects



Properties that characterize the object (state)

A function (method) used to create objects and initialize objects' variables

To ask objects to perform an action or behave in a certain way.

Define a class for objects (cont.)

- The syntax for class declaration:

```
[AccessControlModifier] class ClassName {
   // Class body contains variables and methods
   ......
}
```

- Example:

```
public class Circle {
   double radius;              // variables
   String color;

   double getRadius() { ..... }  // methods
   double getArea() { ..... }
}
```

No explicit constructor

Constructing Objects using Constructors

- A constructor is a special method used to create and initialize objects.
- It has the same method name as the class name
- Do not have a return type, not even void.
- Constructors are invoked using the `new` operator when an object is created.
- Constructors type:
  1. Default constructor
  2. Parametrized constructor

Default Constructor

- No parameters.
- Automatically provided by Java if no constructors are defined (implicit definition).
- You may use the following or not:

```
[AccessControlModifier] ClassName() {
    // Initialization code
}
```

- Example:

```
public Circle() {
    // Initialization code
}
```

Parameterized Constructor

- Takes parameters to initialize object attributes.
- Parameterized constructor syntax:

```
[AccessControlModifier] ClassName(Parameter list) {
    // Initialization code
}
```

- Example:

```
 /** Construct a circle with a specified radius */
Circle(double newRadius) {
        radius = newRadius;
}
```

Constructor Overloading

- Method (constructor) overloading means that the same method name can have different implementations (versions).

- The different implementations must be distinguishable by the parameter list.
  1. The number of parameters
  2. The type of parameters
  3. And their order.

- Provides flexibility in object creation.

Example 1: Implicit (default) constructor

```java
public class Circle {
    double radius = 1.0; // Default radius

    // Method to calculate the area of the circle
    public double area() {
        return Math.PI * radius * radius;
    }

    // Main method for testing
    public static void main(String[] args) {
        // Using the implicit default constructor
        Circle c1 = new Circle();
        System.out.println("Area: " + c1.area());
    }
}
```

```
Area: 3.141592653589793
```

Example 1: Implicit (default) Constructor

```java
public class Circle {
    double radius = 1.0; // Default radius

    // Method to calculate the area of the circle
    public double area() {
        return Math.PI * radius * radius;
    }

    // Main method for testing
    public static void main(String[] args) {
        // Using the implicit default constructor
        Circle c1 = new Circle();
        System.out.println("Area: " + c1.area());
    }
}
```

Attributes (data fields)

Methods: only one method

No explicit constructor, the compiler will use the default constructor.

Example 2: Explicit (Parametrized) Constructor

```java
public class Circle {
    double radius = 1.0; // Default radius

    // Parameterized constructor
    public Circle(double r) {
        radius = r; // Sets radius to the specified value
    }

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

    // Main method for testing
    public static void main(String[] args) {
        // Using parametrized constructor
        Circle c2 = new Circle(3.2);
        System.out.println("Area: " + c2.area());
    }
}
```

Attributes (data fields)

Explicit (parameterized) constructor

Area method

```
Area: 32.169908772759484
```

Example 3: Overlapped Constructors

```
/**********************************************************
Classes & Objects
Author: Dr. Fadi Alzhouri
Example 3: Overlapped Constructors
**********************************************************/
public class Circle {
    double radius = 1.0;
    String color = "black";

    public Circle() {
    }

    public Circle(double r) {
        radius = r; // Set a new radius
    }

    public Circle(double r, String col) {
        radius = r; // Set a new radius
        color = col; // Set a new color
    }
```

The default constructor

Constructor with 1 parameter

Constructor with 2 parameters

The rest of this code is in the next slide

Example 3: Overlapped Constructors (cont.)

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

    // Main method for testing
    public static void main(String[] args) {
        Circle c1 = new Circle( );
        Circle c2 = new Circle(3.2);
        Circle c3 = new Circle(4,"red");
    }
}
```

Three ways to instantiate and initialize objects

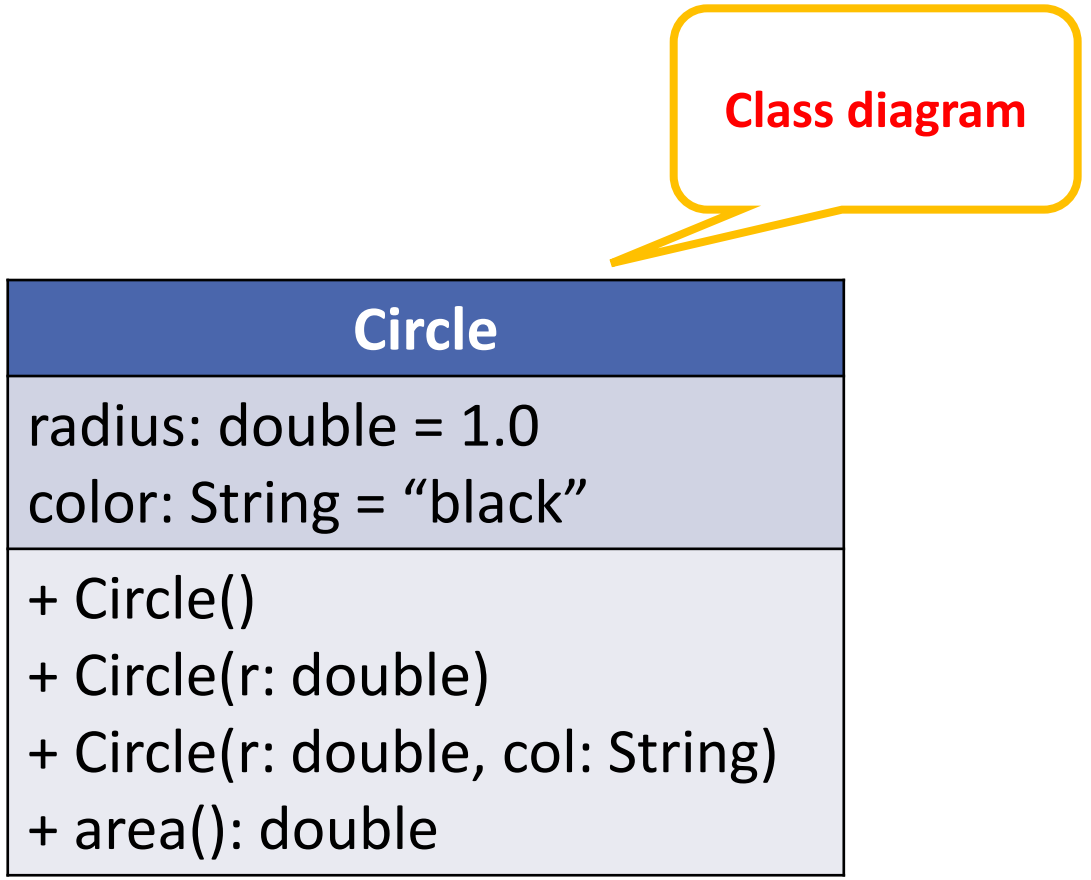
Unified Modeling Language (UML)

- Unified Modeling Language (UML) is a standardized modeling language used to visualize the design of a system.
- UML helps in specifying, visualizing, and documenting software system artifacts.
- UML includes various types of diagrams such as:
  1. Class diagrams
  2. Sequence diagrams
  3. Use case diagrams

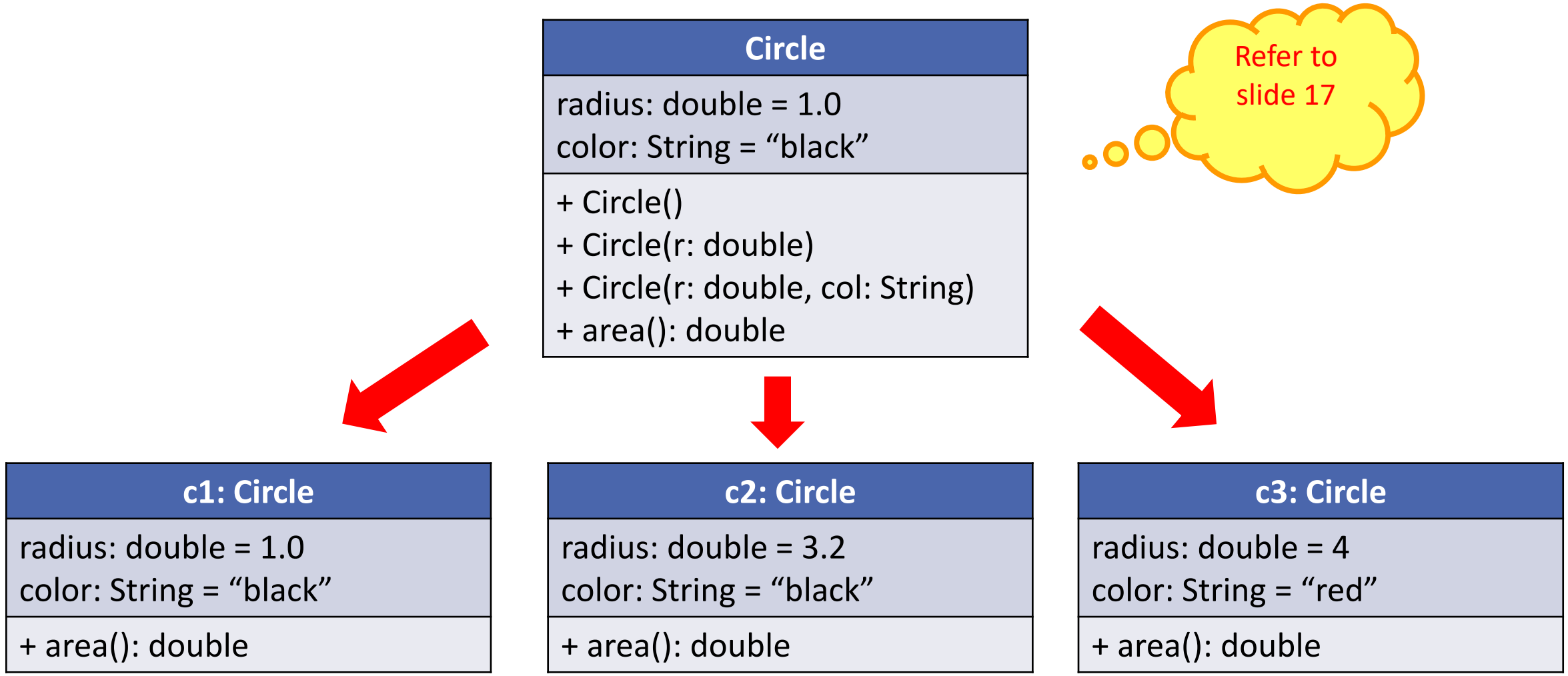
- Class Diagrams: Used to represent the structure of a system by showing the system's classes and their relationships.

Class Diagram

- Class Name: Circle
- Attributes:
  - radius: double
  - color: String
- Methods:
  - Circle()
  - Circle(r: double)
  - Circle(r: double, col: String)
  - area(): double



Class diagram with instances (objects)



Exercise 1

1. Create two objects of the Circle class defined alongside based on the following specifications:
   - Instance c4:
     - Radius: 5.0
     - Color: "Green"
   - Instance c5:
     - Radius: 1.0
     - Color: "Yellow“
2. Draw the UML diagrams to show the above objects

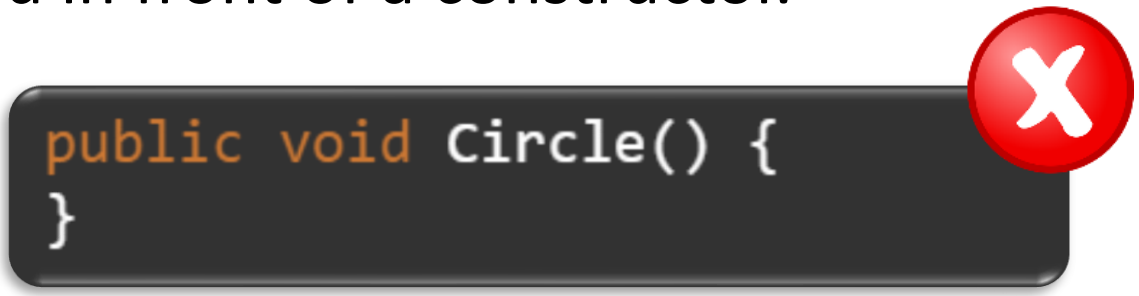
Circle
radius: double = 1.0 color: String = “black”
+ Circle() + Circle(r: double) + Circle(r: double, col: String) + area(): double

Exercise 2

Assume you've got a new job at a new car rental company. The manager asks you to develop a reservation system. As a first step in this project and based on the concepts you've learned in object-oriented programming using Java, consider the classes needed to design and implement this system.

Review

- Don’t put the `void` keyword in front of a constructor.



- A class may be defined without constructors. In this case, a public no-arg (default) constructor with an empty body is implicitly defined in the class. This constructor, called a default constructor, is provided automatically only if no constructors are explicitly defined in the class

References

- **Introduction to Java Programming, Brief Version, Global Edition, 11th edition,** Published by Pearson (June 21, 2018) © 2018