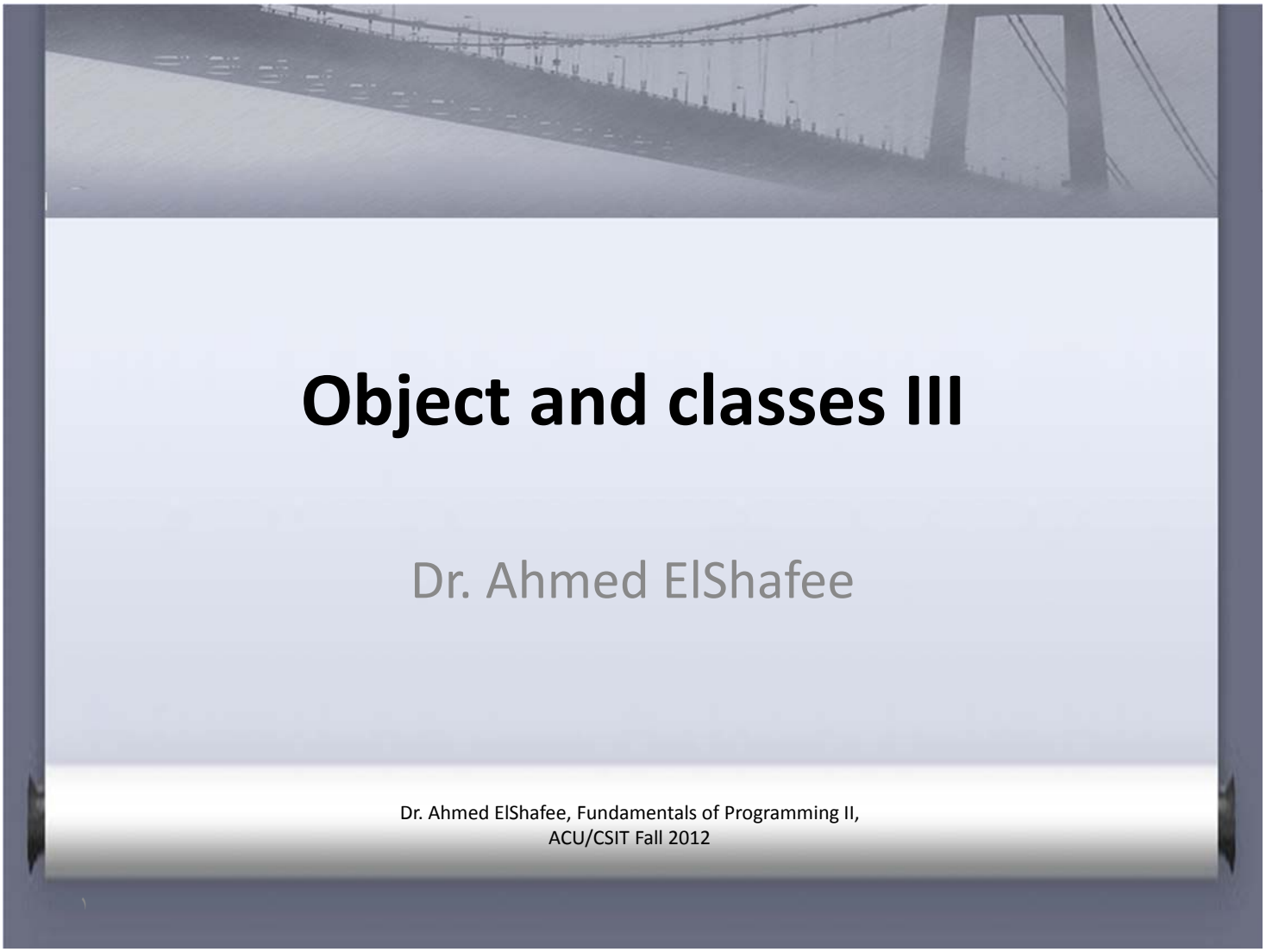


Object and classes III

Dr. Ahmed ElShafee

Agenda

1. *
2. *
3. *

Public and Private Fields

- In a class, you can control how fields are accessed outside the class.
- You can create some fields that can be accessed outside the class and you can create other fields that other classes cannot access.
- To create a field that can be accessed by only members of the same class, precede its data type with the **private** keyword.

```
public class House{
String propertyNumber;
private String kitchenCharacteristics;
byte Stories;
int bedrooms;
private boolean bathroomNeedCleaning;
double Value;
}
```


- To create a field that can be accessed by members of the same class and members of other classes, precede its data type with the **public** keyword.

```
public class House{
public String propertyNumber;
private String kitchenCharacteristics;
byte Stories;
public int bedrooms;
private boolean bathroomNeedCleaning;
double Value;
}
```

- by default, the members of a Java class are public.
- This means that if you do not use the public or the private keyword, a field is automatically made public.

Public and Private Methods

- you can control the access level of a method.
- To do this, you can precede the return value or void of a method with **private** or **public**.
- When a method has been marked as private, it can be accessed by methods of the same class but other classes cannot call that method.
- If a method was created as public, other methods of the same class can access it and instances of other classes can call it.
- the methods of a class are public by default.
- Therefore, if you want to hide a member of a class, you must explicitly mark it as private.

Accessing private fields/methods

- We saw that you could create (or make) a field as private to prevent its access from the outside of its class. Here is an example:

```
public class Square {
private double side;
}
```

- After making a field private, other classes can neither specify its value not retrieve
- To solve this problem, you create a property for the field. There are two types of properties: a reader and a writer.

Read property

- A property is referred to as *read* if its role is only to make available the value of the field it represents.
- To create a read property , create a method with the following characteristics:

  a. Because the method is used to communicate with the outside world, it must be created as **public**

  b. The method must return the same type of value as the field it represents

  c. The name starts with *get* and ends with the name of the field with the first letter in upper case

  d. The method does not take any argument

  e. The method returns the field it represents

```
public class Square{
private double side;
// This is a read property
public double getSide() {
return side;
}
}
```

- A read property is also referred to as read-only property because the clients of the
- class can only retrieve the value of the property but they cannot change it.
- Once a read property has been created, other classes can access it

lecture0601

- Check lab manual

lecture0602

- Check lab manual

A Property Writer

- instead of, retrieving the value of a field of a class, you may want external classes to be able to change the value of that member.
- Continuing with our policy of hiding a field as private, you can create another type of property.
- a property is referred to as write if it can change (or write) the value of its corresponding field.
- To build a write property, create a method using the following characteristics:

a. Because the method is used to communicate with the outside world, it must be created as public

b. Because the method is used by outside classes to assign a value to the field, the method is (or should be) created as **void**

c. The name starts with *set* and ends with the name of the field with the first letter in upper case

d. The method takes one argument that is the same type as the field it refers to

e. In the body of the method, assign the argument to the field it represents


```
public class Square {
private double side;
// This is a write property
public void setSide(double value) {
side = value;
}
}
```

- As you see, clients of a class can change the corresponding field of a member variable through the property writer.
- Based on this relationship, it is not unusual for a client of a class to make an attempt to "mess" with a field.
- For example, an external object can assign an invalid value to a member variable.

lecture0603

- Check lab manual

lecture0604

- Check lab manual

Inheritance

- The primary characteristic of the objects on the above pictures is that they are balls used in different sports.
- they share is that they are round.
- one made for one sport cannot (or should not) be used in another sport

- The common characteristics of these objects can be listed in a group like a Java class.
- The class would appear as:

```
class Ball {
String TypeOfSport;
double Size;
}
```

- If you were asked to create a class to represent these balls, you may be tempted to implement a general class that defines each ball
- A bad idea because, despite their round resemblance, there are many internal differences among these balls.
- Java provide an alternate solution to this type of situation

- Inheritance consists of creating a class whose primary definition or behavior is based on another class.
- In other words, inheritance starts by having a class that can provide behavior that other classes can improve on.

```
public class Circle {
private double radius;
public Circle() {
this.radius = 0.00;
}
}
```


lecture0605

- Check lab manual

- Now, suppose you want to create a class for a sphere. You could start from scratch as we have done so far.
- On the other hand, since a sphere is primarily a 3-dimensional circle, and if you have a class for a circle already, you can simply create your sphere class that uses the already implemented behavior of a circle class.
- Creating a class that is based on another class is also referred to as deriving a class from another.
- The first class serves as parent or base.
- The class that is based on another class is also referred to as child or derived.

- To create a class based on another, you use the following formula:

```
modifier class NewChild extends BaseClass {
// Body of the new class
}
```

```
class Sphere extends Circle {
// The class is ready
}
```

```
public static void main(String[] args) {
Sphere ball = new Sphere();
        ball.setRadius(25.55);
        System.out.println("\nSphere Characteristics");
        System.out.printf("Side: %f\n", ball.getRadius());
        System.out.printf("Diameter: %f\n",
                ball.calculateDiameter());
        System.out.printf("Circumference: %f\n",
                ball.calculateCircumference());
        System.out.printf("Area: %f", ball.calculateArea());
}
```


lecture0606

- Check lab manual

- When a class is based on another class, all public members of the parent class are made available to the derived class.
- While other methods and classes can also use the public members of a class,
- That is, the child class does not have to "qualify" the public members of the parent class when these public members are used in the body of the derived class.


```
public class Sphere extends Circle {
   public void ShowCharacteristics() {
System.out.println("\nSphere Characteristics");
       System.out.printf("Side: %f\n", getRadius());
       System.out.printf("Diameter: %f\n",
              calculateDiameter());
       System.out.printf("Circumference: %f\n",
              calculateCircumference());
       System.out.printf("Area: %f", calculateArea());    }}
```

```
public static void main(String[] args) {
Circle round = new Circle();
round.setRadius(25.55);
round.ShowCharacteristics();
Sphere ball = new Sphere();
ball.setRadius(25.55);
ball.ShowCharacteristics();
}
```

lecture0607

- Check lab manual

lecture0608

- Check lab manual

Overriding Derived Methods

- You may have noticed in the previous example that the derived class produced the same results as the base class.
- inheritance is used to solve various object-oriented programming (OOP) problems.
- One of them consists of customizing, adapting, or improving the behavior of a method of the parent class.
- For example, although both the circle and the sphere have an area, their areas are not the same.
- Another example, to display cylinder properties we need to display its height too not just radius.
- Providing a new version of a method that exists in the parent is referred to as overriding it.

- To override a method, start it with the expression **@Override**. Then create the new version of the method. It must use the same return type, the same name, and the same arguments,

```
class Circle {
private double radius;
public double calculateArea() {
return radius * radius * 3.14159;
}
}
```

```
class Sphere extends Circle {
@Override
public double calculateArea() {
return 4 * getRadius() * getRadius() * 3.14159;
}
}
```

```
public static void main(String[] args) {
Sphere ball = new Sphere();
ball.setRadius(25.55);
System.out.println("\nSphere Characteristics");
System.out.printf("Side: %f\n", ball.getRadius());
System.out.printf("Diameter: %f\n",
ball.calculateDiameter());
System.out.printf("Circumference: %f\n",
ball.calculateCircumference());
System.out.printf("Area: %f", ball.calculateArea());
}
```


lecture0609

- Check lecture notes

- Besides customizing member variables and methods of a parent class, you can add new members as you wish
- In our example, while a circle is a flat shape, a sphere has a volume.
- In this case, you may need to calculate the volume of a sphere as a new method or property of the derived class.

```
class Sphere extends Circle {
@Override
public double calculateArea() {
return 4 * getRadius() * getRadius() * 3.14159;
}
public double calculateVolume() {
return 4 * 3.14159 * getRadius() *
getRadius() * getRadius() / 3;
}}
```


lecture0610

- Check lab manual

Protected Members

- If you create a class member and mark it as **protected**, the other classes of the same application (package) and the classes derived from that parent class can access those protected members.

```
public class Circle {
    protected double radius;
    void setRaduis(double r)
            {raduis=r;}
…
}
```



```
public class Cylinder extends Circle {
    private double height;
public double calculateLateralArea() {
return 2 * 3.14159 * radius * height;
}
Void setHeight(double h)
  {Height=h;}
….
}
```

- Public variables, are variables that are visible to all classes.
- Private variables, are variables that are visible only to the class to which they belong.
- Protected variables, are variables that are visible only to the class to which they belong, and any subclasses.


```
private static Cylinder create() {
        double r, h;
        Cylinder cup = new Cylinder();
        Scanner scnr = new Scanner(System.in);
        System.out.print("Enter the Radius: ");
        r = scnr.nextDouble();
        System.out.print("Enter the Height: ");
        h = scnr.nextDouble();
        cup.setRadius(r);
        cup.setHeight(h);
        return cup;
    }
```

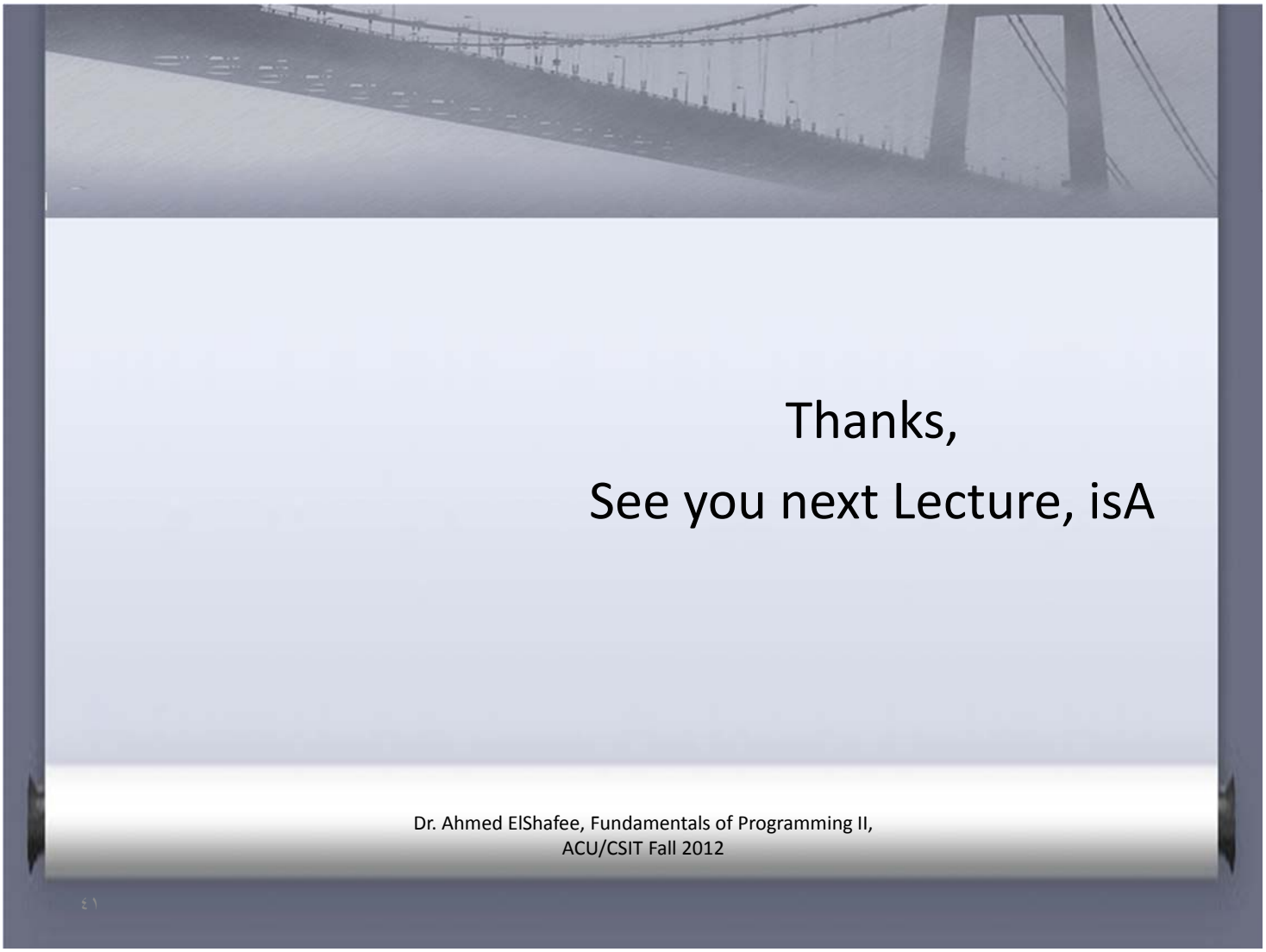
```
public static void main(String[] args) {
        Cylinder tube = create();
        System.out.printf("\nLateralArea = %f", calculateLateralArea());
    }
```

lecture0611

- Check lab manual

lecture0612

- Check lab manual



Thanks,

See you next Lecture, isA