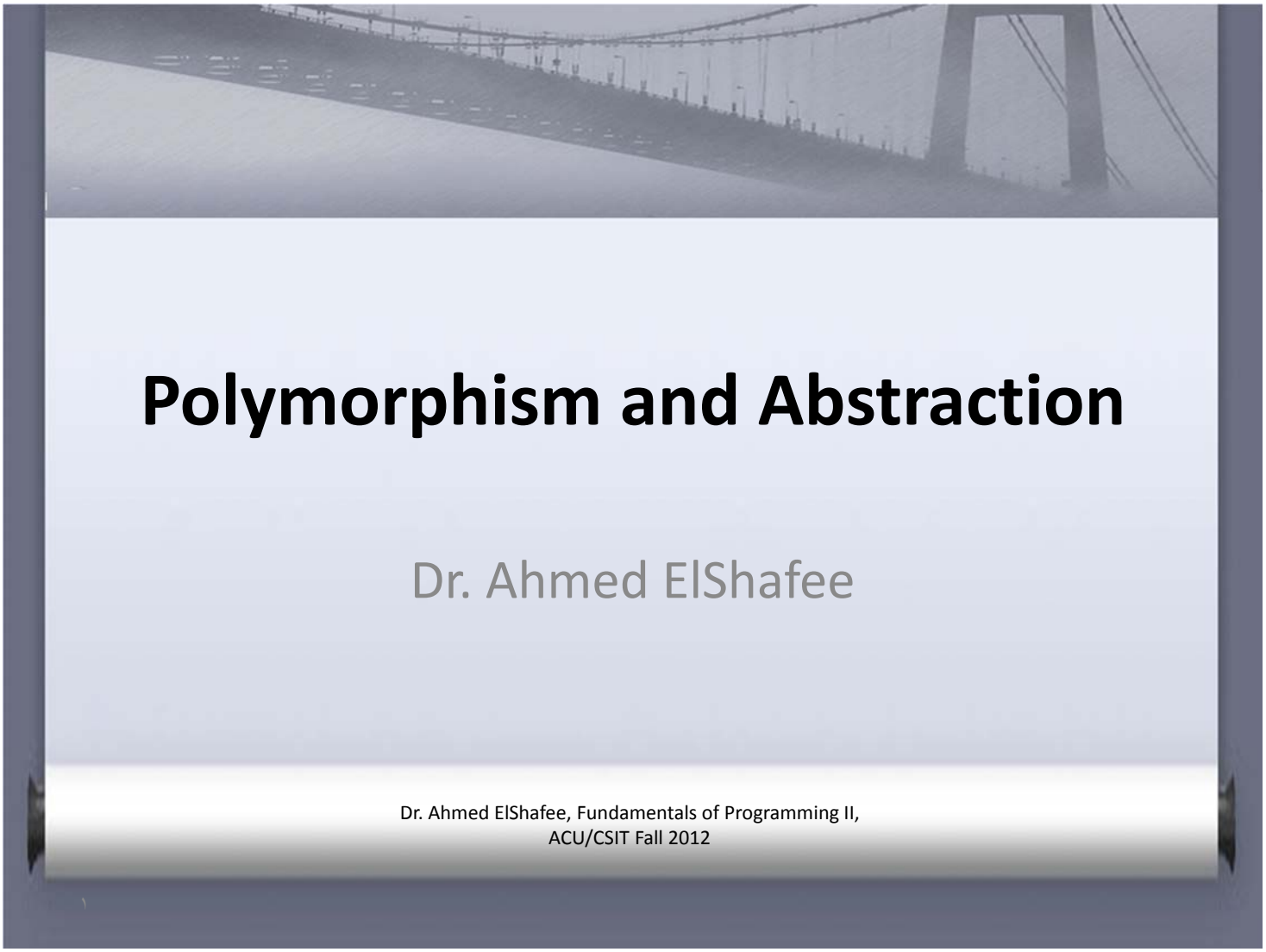

Polymorphism and Abstraction

Dr. Ahmed ElShafee

Agenda

1. *
2. *
3. *

Before beginning

- Polymorphism is a long word for a very simple concept.
- Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface.
- The beauty of polymorphism is that the code working with the different classes does not need to know which class it is using since they're all used the same way.
- A real world analogy for polymorphism is a button.
- you simply apply click it. What a button "does," however, depends on what it is connected to and the context in which it is used — but the result does not affect how it is used.

- If your boss tells you to press a button, you already have all the information needed to perform the task.

- In the programming world, polymorphism is used to make applications more modular and extensible. Instead of messy conditional statements describing different courses of action, you create interchangeable objects that you select based on your needs. That is the basic goal of polymorphism

Introduction

- In a program, you can create a class whose role is only meant to provide fundamental characteristics for other classes.
- This type of class cannot be used to declare a variable.
- Such a class is referred to as abstract.
- Therefore, an abstract class can be created only to serve as a parent class for other classes.

The Fields of an Abstract Class

- To create an abstract class, type the **abstract** keyword to the left of its name.
- Here is an example

```
abstract class DigitalCamera {
}
```

- Because an abstract class is primarily a class, you can add fields to it. Here is an example:

```
abstract class DigitalCamera {
String make;
String model;
double megapixels;
double size;
}
```

- After creating an abstract class, you cannot (yet) declare a variable from it.
- You must first derive a class from it. If the abstract contains only fields like our DigitalCamera class, you do not have to do anything in the derived class.

```
abstract class DigitalCamera {
String make;
String model;
double megapixels;
double size;
}
class PointAndShoot extends DigitalCamera {
}
```


- To use the class or its parent, you can declare a variable using it.
- Then use its fields as you see fit.
- For example, you can assign values to the fields.
- Here are examples:

```
public class Exercise {
public static void main(String[] args) {
PointAndShoot camera = new PointAndShoot();
camera.make = "Canon";
camera.model = "Powershot A590 IS";
camera.megapixels = 8.0;
camera.price = 129.95;
}
}
```

- To use the class or its parent, you can declare a variable using it.
- Then use its fields as you see fit.
- For example, you can assign values to the fields.

```
public class Exercise {
public static void main(String[] args) {
PointAndShoot camera = new PointAndShoot();
camera.make = "Canon";
camera.model = "Powershot A590 IS";
camera.megapixels = 8.0;
camera.price = 129.95;
}
}
```


- In our introduction, we mentioned that you could not declare a variable using an abstract class.
- In reality you can.
- One of the rules you must follow is that, if using the **new** operator, you must use the derived class to initialize the variable

```
abstract class DigitalCamera {
}
class PointAndShoot extends DigitalCamera {
}
public class Exercise {
public static void main(String[] args) {
DigitalCamera camera = new PointAndShoot();
}
}
```

Lecture0601

Check lab manual

The Methods of an Abstract Class

- Like a normal class, an abstract class can contain one or more methods.

```
abstract class DigitalCamera {
	String make;
	String model;
	double megapixels;
	double price;
	void describe() {
		System.out.println("Digital Camera");
		System.out.printf("Make: %s\n", make);
		System.out.printf("Model: %s\n", model);
		System.out.printf("Megapixels: %.2f\n", megapixels);
		System.out.printf("Price: %.2f", price);
	}
}
```

- After adding a method to the abstract class, you can declare a variable from it and access that method using an instance of the derived class.
- Because a class derived from an abstract is primarily a normal class, you can add new members to it.
- Here is an example:

```
public class PointAndShoot extends DigitalCamera {
void create() {
        make = "Canon";
        model = "Powershot A590 IS";
        megapixels = 8.0;
        price = 129.95;
}
}
```


- When you declare a variable using the abstract class, you cannot access the members of the derived class.
- This means that the following code will produce an error:

```
public static void main(String[] args) {
        DigitalCamera camera = new PointAndShoot();
        camera.make = "Olympus";
        camera.model = "FE-170";
        camera.megapixels = 6.0;
        camera.price = 119.95;
        camera.describe();
        }
}
```

Lecture0602

- Check lab manual

Abstract and Pure Abstract Classes

- The most common reason for creating an abstract class is to have a class that would be used as a base class for future inheritance. In such a class, you do not implement any of its methods.
- The class is only used as a foundation for other classes.
- Such a class is referred to as pure abstract.

- To create a pure abstract class, add the fields and methods that its derived classes would share.
- Do not implement any method in the class.
- You only declare the members.
- To create a class that is used only as a base for other classes, type the **abstract** keyword on the left side of each member, especially each method, but do not implement any method.
- This means that each method is declared with either **void** or its return type, followed by the name of the method, its parentheses, its arguments if any, and a semi-colon.


```
abstract class DigitalCamera {
      String make;
      String model;
      double megapixels;
      double price;
      abstract void describe();
}
```

- After creating the abstract class, once again, in order to use it, you must first derive a class from it.
- This time, you must implement each method of the abstract class.

```
class PointAndShoot extends DigitalCamera {
	void describe() {
		System.out.println("Digital Camera");
		System.out.printf("Make: %s\n", make);
		System.out.printf("Model: %s\n", model);
		System.out.printf("Megapixels: %.2f\n", megapixels);
		System.out.printf("Price: %.2f", price);
	}
}
```

- If you forget to, or do not, implement an abstract method of the parent in the derived class, the compiler would produce an error.
- After deriving the class, you can use it by declaring a variable from it.

- You can declare the variable using either the abstract class or the derived class.
- Then you can access the public and protected members of the abstract class.

```
DigitalCamera camera = new PointAndShoot();
camera.make = "Olympus";
camera.model = "FE-170";
camera.megapixels = 6.0;
camera.price = 119.95;
camera.describe();
```

Lecture0603

- Check lab manual

Interfaces

- An interface is a special class whose purpose is to serve as a template that actual classes can be based on.
- An interface is primarily created like a class: it has a name, a body and can have members.
- To create an interface, instead of the **class** keyword, you use the **interface** keyword.

```
interface Dimension {
}
```

- As done when creating classes, you can specify the access level of an interface.

```
public interface Size {
}
```

- As done for a class, the members of an interface are listed in its body.
- In an interface, you cannot declare fields like those we have used in other classes, except constants.
- This means that an interface mostly contains methods and usually only methods (you can also create constants).
- The most important rule of an interface is that, if you declare a method in it, you cannot define that method.
- In other words, you can provide the signature of a method (the void or return value, the name of the method, the parentheses, and the arguments, if any, followed by a semi-colon).


```
public interface DigitalCamera {
String getMake();
String getModel();
}
```

- An interface is used to lay a foundation for other interfaces or other classes.
- For this reason, it is the prime candidate for class derivation.
- To derive a class from an interface, type the name of the class, followed by the **implements** keyword, followed by the name of the interface, and followed by the body of the new class.
- Here is an example of a class named PointAndShoot that implements an interface named DigitalCamera:

```
public class PointAndShoot implements DigitalCamera {
}
```


- When defining a class that implements an interface, you can add or not add one or more new methods

```
interface DigitalCamera {
        String getMake();
        void setMake(String make);
        String getModel();
        void setModel(String model);
        double getMegapixels();
        void setMegaPixels(double megapixels);
        double getPrice();
        void setPrice(double value);
}
class PointAndShoot implements Size, DigitalCamera {
        void describe() {
        }
}
```

- The primary rule of interfaces is t that, if you derive a class from an interface, you must implement all methods that were created in the interface.
- Once the class is ready, you can then use it as you see fit.

Lecture0604

- Check lab manual

An Abstract Class Based on an Interface

- You can create an abstract that is based on an interface.
- To do this, precede the name of the new class with the **abstract** keyword

```
interface DigitalCamera {
}
abstract class StoreItem implements DigitalCamera {
}
```


- We saw that if you create a class that implements an interface, you must define all methods declared in the interface.
- There is an exception to this rule.
- If you create an abstract class that implements an interface, since you will not implement the methods of the interface in the abstract class, you do not have to implement any method in the derived class.

```
interface DigitalCamera {
	String getMake();
	void setMake(String make);
	String getModel();
	void setModel(String model);
	double getMegapixels();
	void setMegapixels(double megapixels);
}
abstract class StoreItem implements DigitalCamera {
	String itemNumber;
	double getPrice();
	void setPrice(double value);
}
```


- When creating a class that is based on an abstract that itself is based on an interface, you must implement all methods of the interface and all methods of the abstract

Lecture0605

- Check lab manual

- Of course, in the new class, you can add new fields and/or methods if you judge them necessary.
- After creating the class, you can instantiate it and access its public or protected members.

Lecture0606

- Check lab manual

- Just as you can derive a class from an interface, you can create an interface that itself is based on another interface. To create such an interface, use the following formula:

```
modifier interface InterfaceName extends InterfaceName {
}
```

- As you can see from this formula, you use the **extends** keyword instead of **implements** when creating an interface that extends an existing interface.

```
interface Size {
}
interface DigitalCamera {
}
interface Camcorder extends Size {
}
```

- The Java language does not allow multiple inheritance, which is the ability to create a class based on more than one class
- Multiple inheritance is allowed only if the bases are interfaces. To create a multiple inheritance, separate the names of interfaces with a comma.

```
interface Size {
}
interface DigitalCamera {
}
class PointAndShoot implements Size, DigitalCamera {
}
```


Lecture0607

- Check lab manual

Thanks,

See you next Lecture, isA