CHAPTER 4

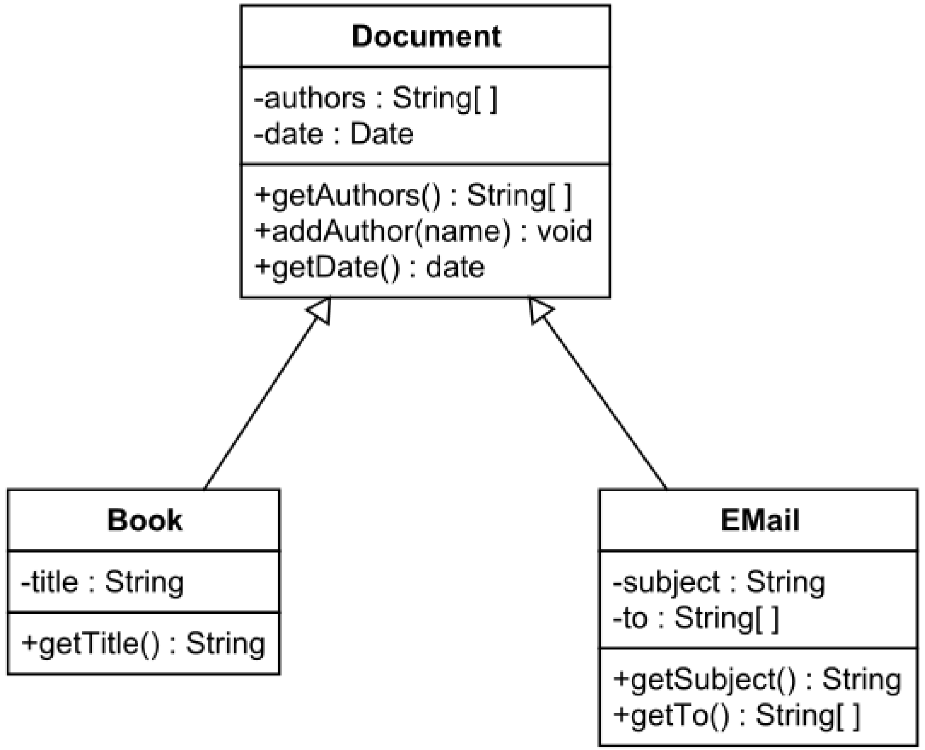
INHERITANCE, ABSTRACT CLASS, INTERFACE, POLYMORPHISM

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

INHERITANCE RELATIONSHIP

- An important feature of OO is inheritance.
- Inheritance allows classes to inherit (take) attributes and operations of other classes.
- This will simplify the UML class diagram that we build during analysis and design and will reduce code duplication.
- A class that has common attributes and operations, that are going to be inherited, is called "super" class, "parent" class or "base" class.
- Classed that inherit from other classes, are called "sub" classes, "child" classes or "derived" classes.

INHERITANCE RELATIONSHIP



Document

-authors : String[ ]
-date : Date

+getAuthors() : String[ ]
+addAuthor(name) : void
+getDate() : date

Book

-title : String

+getTitle() : String

EMail

-subject : String
-to : String[ ]

+getSubject() : String
+getTo() : String[ ]

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

HOW TO IMPLEMENT INHERITANCE ?

- In Java we use the keyword extends to indicate that a class is inheriting its characteristics from another class
- In order to use the inherited attributes; they should be declared in the parent class as either public (public accessors) or protected.
- If we only want child classes to use the attributes; protected access modifier is used.
- A child class can have its own attributes and operations.
- In Java: protected allows access within the same package and also by subclasses, even if they are in different packages.
- Protected are denoted by # in UML.

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

An Example

```
public class Employee
{
      protected String name;
      protected double salary;
      ... // some code here


}


public class SalesEmployee extends Employee
{
     private double sales, commRate;


  ... // some code here



}
```

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

CONSTRUCTORS

- If the parent class has a constructor with parameters, a child class should explicitly call the constructor of the parent class in its constructor and pass it the needed arguments using the keyword super.
- Constructors of the parent class are not inherited.

An Example

```
public class Employee
{
      protected String name;
      protected double salary;

      public Employee(String name, double salary)
      {
         this.name=name; this.salary=salary;
      }
      ... // some code here
}


public class SalesEmployee extends Employee
{
     private double sales, commRate;

     public Employee(String nam, double sly,double sls, double cmrt )
     {
         super(nam, sly);
         sales= sls;   commRate= cmrt;
     }

  ... // some code here
```

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

```
}
```

CREATING OBJECTS

- See next Example

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

An Example

```
public class TestEmployee
{
      publis static void main(String[] args)
      {
          SalesEmployee sm = new SalesEmployee(“ahmad”, 500, 100, 0.10);

          Employee em = new Employee(“enas”, 600);

          //.. some code here

      }


}
```

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

OVERRIDING

- A child class can use public operations that are defined in the parent class.
- A child class can add other operations that are specific to the child class.
- A child class can also redefine the parent class operations. This is called overriding.
- Overriding allows the child class to add its own implementation to the inherited operation, but it should keep the same signature (name and parameters)

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

An Example

```
public class Employee
{
      protected String name;
      protected double salary;

      public Employee(String name, double salary)
      {
         this.name=name; this.salary=salary;
      }
      public String toString()
      {
        return "name is: "+ name+ " "+ salary;
      }
      ... // some code here
}


public class SalesEmployee extends Employee
{
     private double sales, commRate;

     public Employee(String nam, double sly,double sls, double cmrt )
     {
         super(nam, sly);
         sales= sls;   commRate= cmrt;
     }
     @Override
     public String toString()
     {
        return super.toString()+” sales: “ + sales;
     }
     ... // some code here

}
```

ABSTRACT CLASS

- An abstract method is a method that has no implementation.
- An abstract class is a class that cannot be instantiated.
- A class that cab be instantiated is called a concrete class.
- An abstract class usually has abstract methods (one or more).
- It is possible to define an abstract class without having abstract methods in it.
- An abstract class may have attributes, constructor, etc.

An Example

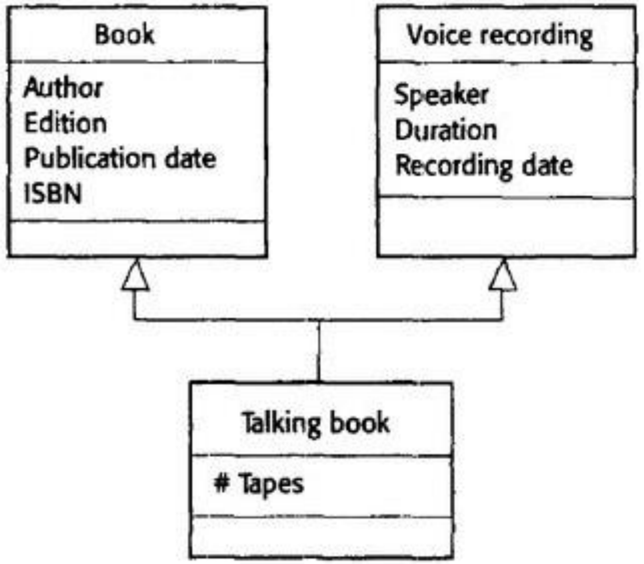
```
public abstract class Animal
{
   protected int age;
   protected String gender;

   public Animal(... ){ ...}

   public abstract void eat( );

   // ... some code here

}
```

INTERFACE

- An Interface in Java is an abstract type that defines a set of methods a class must implement.
- An interface acts as a contract that specifies what a class should do, but not how it should do it.
- It is used to achieve abstraction and multiple inheritance in Java.
- We define interfaces for capabilities (e.g., Comparable, Serializable, Drawable).

INTERFACE

- An interface solves the problem of multiple inheritance.
- It is possible for a class to extend one class (concrete or abstract) and to implement many interfaces.



Multiple inheritance example

INTERFACE

- An interface **does not have constructors.**
- All variables in an interface are **public, static, and final** (constants).
- Interfaces **cannot contain ordinary (concrete) methods**

Interfaces (Java 8 and later) add the following:

  - **Default methods** (with implementation)
  - **Static methods** (with implementation)

- Interface methods are **public** if the access modifier is not specified

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE
TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

INTERFACE

Note that:

Abstract methods must be overridden in a concrete class implementing the interface.

Default methods may be overridden (overriding these methods is not a must).

Static methods cannot be overridden.

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE
TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

An Example

```
Public interface Device {

    //  Variables (constants)
    int MAX_POWER = 100;          // public static final by default
    String TYPE = "Electronic";   // public static final

    //  Aabstract method
    void turnOn();

    //  Default method
    default void status() {
        System.out.println("Device is working");
    }

    //  Static method
    static void info() {
        System.out.println("All devices are electronic");
    }
}
```

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

An Example

```
Public class Laptop implements Device {

    // Must implement abstract method
    @Override
    public void turnOn() {
        System.out.println("Laptop is turned on");
    }

    // Optional: override default method
    @Override
    public void status() {
        System.out.println("Laptop status: ON");
    }

    void displayDetails() {
        // Access interface variables
        System.out.println("Type: " + TYPE);
        System.out.println("Max power: " + MAX_POWER);
    }
}
```

An Example

```
public class Test {
    public static void main(String[] args) {

        Laptop l = new Laptop();   // concrete class reference

        l.turnOn();        // abstract method implementation
        l.status();        // overridden default method
        l.displayDetails();

        Device.info();     // static method (must use interface name)
    }
}
```

POLYMORPHISM

- Polymorphism means having many forms.
- In inheritance, any child class object can take any form of a class in its parent hierarchy and of course itself as well.
- This means that the child class object can be assigned to any class reference in its parent hierarchy and of course itself as well.

Example :

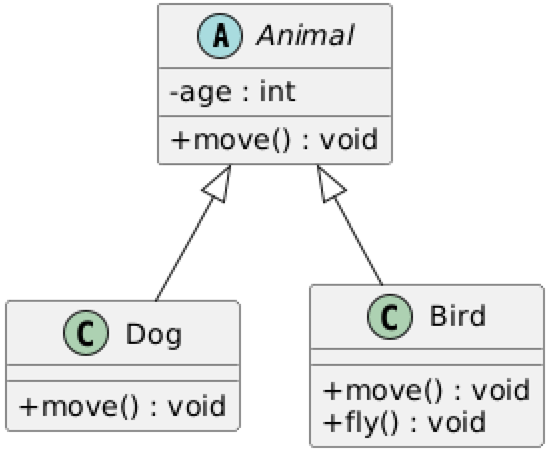
```
Animal a1= new Animal( );
Animal a2 = new Cat( );
```

Where Animal is the parent class of Cat class; it could be a **concrete** class, an **abstract** class or an **interface**.

POLYMORPHISM

- Suppose that Animal class has a method called breathe( ) and Cat class overrides breathe( ) method. This method can be called using a1 or a2 objects.
- The type of the referenced object will determine at runtime which breathe( ) to call.

POLYMORPHISM EXAMPLE



```
Public abstract class Animal {
    Private int age;
    Public abstract void move();
}


Public class Dog extends Animal {
    @Override
    void move() {
        System.out.println("Dog runs");
    }
}


Public class Bird extends Animal {
    @Override
    void move() {
        System.out.println("Bird hops");
    }

    void fly() {
        System.out.println("Bird flies");
    }
}
```

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

```
public class Main {
    public static void main(String[] args) {
        Animal a1 = new Dog();
        Animal a2 = new Bird();

        a1.move();   // Dog runs
        a2.move();   // Bird hops

        // a2.fly();  // ✖ compile-time error (Animal reference)

        // Downcasting to access Bird-specific behavior
        if (a2 instanceof Bird) {
            Bird b = (Bird) a2;
            b.fly(); // Bird flies
        }
    }
}
```

THESE SLIDES ARE DESIGNED TO BE USED IN OOP COURSE AT PHILADELPHIA UNIVERSITY BY ENAS NAFFAR. SOME SLIDES ARE TAKEN FROM PEARSON EDUCATION, AND MARTY STEP

EXERCISE

- **Using the previous example**, create an array of type Animal. Add three Dog objects and two Bird objects to the array. Then, invoke the move() method on all objects in the array, and invoke the fly() method only on objects of type Bird.