

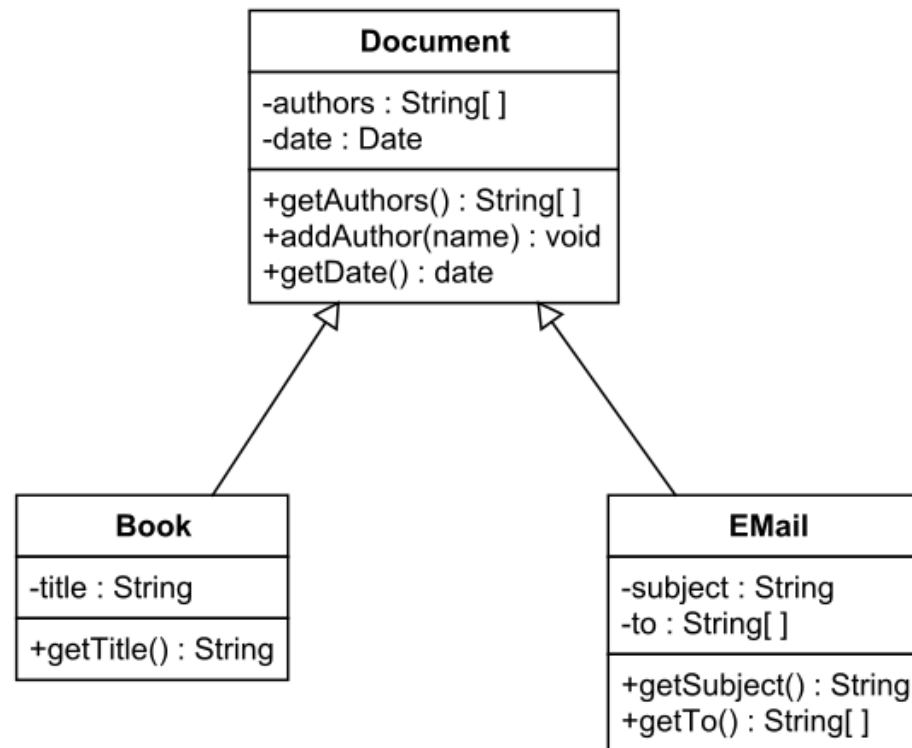


CHAPTER 4

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.
- Classes that inherit from other classes, are called “sub” classes, “child” classes or “derived” classes.

INHERITANCE RELATIONSHIP



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.

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

}
```

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**.

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

An Example

```
public class TestEmployee
{
    public static void main(String[] args)
    {
        SalesEmployee sm = new SalesEmployee("ahmad", 500, 100, 0.10);
        Employee em = new Employee("enas", 600);
        //.. some code here
    }
}
```

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)

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 can 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( );

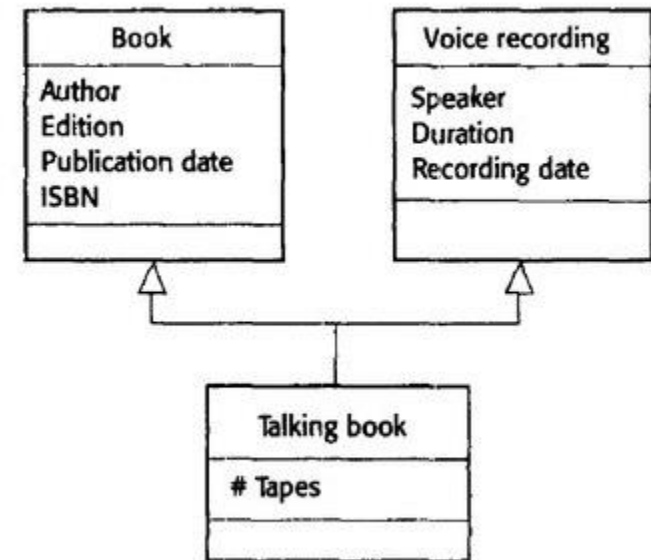
    // ... some code here
}
```

INTERFACE

- An **interface** is a class whose methods are all **public** and **abstract**.
- An interface does not have attributes nor a constructor.
- An interface may have variables which are declared as **static** and **final**.

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

An Example

```
public interface Animal
{
    public void eat( );
    public int calc_age( );
    // ... some code here
}
```


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();

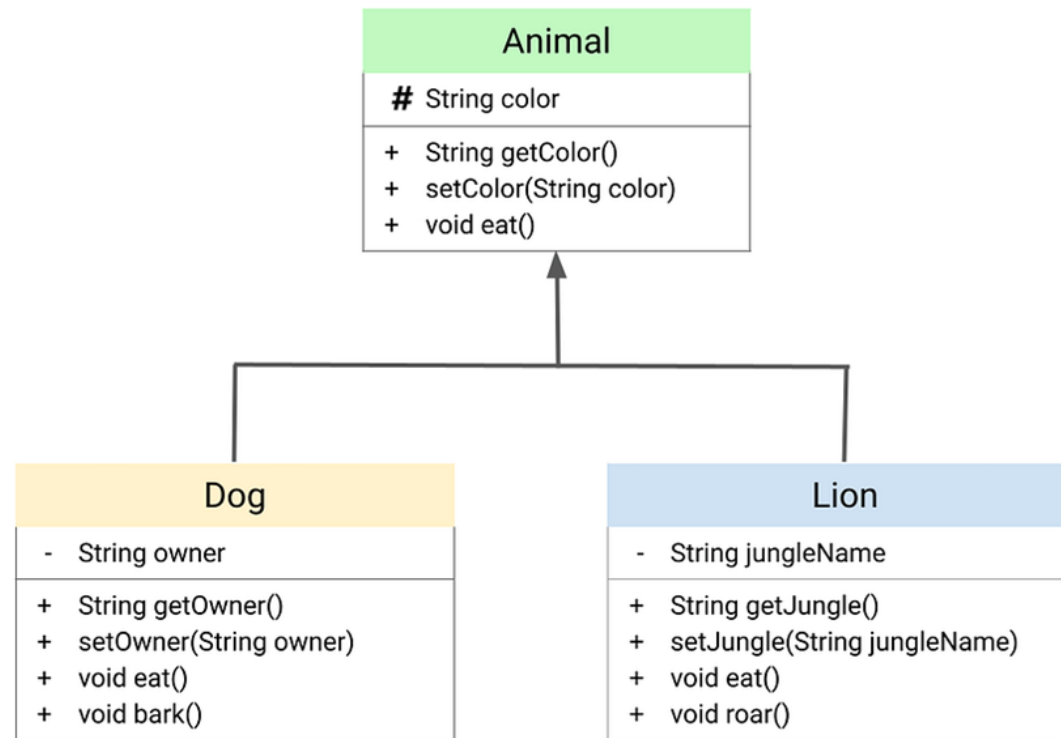
Animal a2 = new Cat();

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



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

- See lectures polymorphism examples