

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

This module examines the concepts of inheritance and polymorphism and how to apply them in object-oriented designs. The laboratory exercise then shows how to implement these concepts in Java.

Learning objectives:

- To gain skill in understanding and properly applying the concepts of inheritance and polymorphism.

Resources:

- LIOUs1 and LIOUs2 BlueJ projects. (Downloadable from UTSONline OOD Course Documents/ Self-Study Modules/ LIOU.zip).
- AddressBookLab7 BlueJ Project (Downloadable from UTSONline OOD Course Documents/ Self-Study Modules).

Contents:

Exercise 1a - Understanding the problem.....	2
Exercise 1b - Evaluating inheritance use	3
Exercise 2 - Examining inheritance in Java	13
Exercise 3 - Lab: Inheritance and Polymorphism.....	18

Self-Study Module: Inheritance and Polymorphism

Exercise 1a - Understanding the problem

The new Language Institute of a local Opera University (LIUO) wants to acquire a software system that will allow staff to manage and maintain a database with the information of the material resources for students. The requirements of the new system are:

The system shall allow the user to enter information about textbooks, CDs and videos.

The system shall allow the user to list the information of all resources and their details.

The system shall allow the user to record the following details of a textbook:

- title of the textbook
- author of the textbook
- language of the textbook
- number of pages
- whether it is available or not
- a comment

The system shall allow the user to record the following details of a CD:

- title of the album
- name of the artist (singer or band)
- language of the CD
- total playing time
- whether it is available or not
- a comment

The system shall allow the user to record the following details of a video:

- title of the video
- name of the director
- language of the video
- total playing time
- whether it is available or not
- a comment

The IT department of the LIUO has to choose between two different systems proposed by two different members of the developers' team. The proposals are detailed in Appendix A.3 in the form of a class diagram and partial Java code. Both solutions work but one uses inheritance and polymorphism and the other does not.

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

Study the information in the appendix to gain understanding of how the proposed solutions work and what design elements (classes, objects and relationships) are involved in both the class diagram and the code. Discuss with your team members any doubts about the given.

Exercise 1b - Evaluating inheritance use

Your next task is to help LIUO decide on what system to choose by evaluating the advantages and disadvantages of each system by analysing the absence and presence of inheritance. To achieve this, you have to fill out Table 1 with *valid arguments* and *examples* based on what is given about the proposed solutions and the requirements of the system.

Criterion \ Solutions	Solution 1	Solution 2
<p><i>Real world abstraction</i></p> <p>How well is the real world modelled in the system?</p> <p>Is there any organisation of classes and objects (level of abstraction)?</p> <p>Is this relevant to what is defined as a good design?</p>		
<p><i>Code duplication</i></p> <p>Is there duplication of code?</p> <p>Is it necessary?</p> <p>Can it be avoided?</p>		
<p><i>Code reuse</i></p> <p>Is any code reused?</p> <p>Can the current code be reused?</p>		

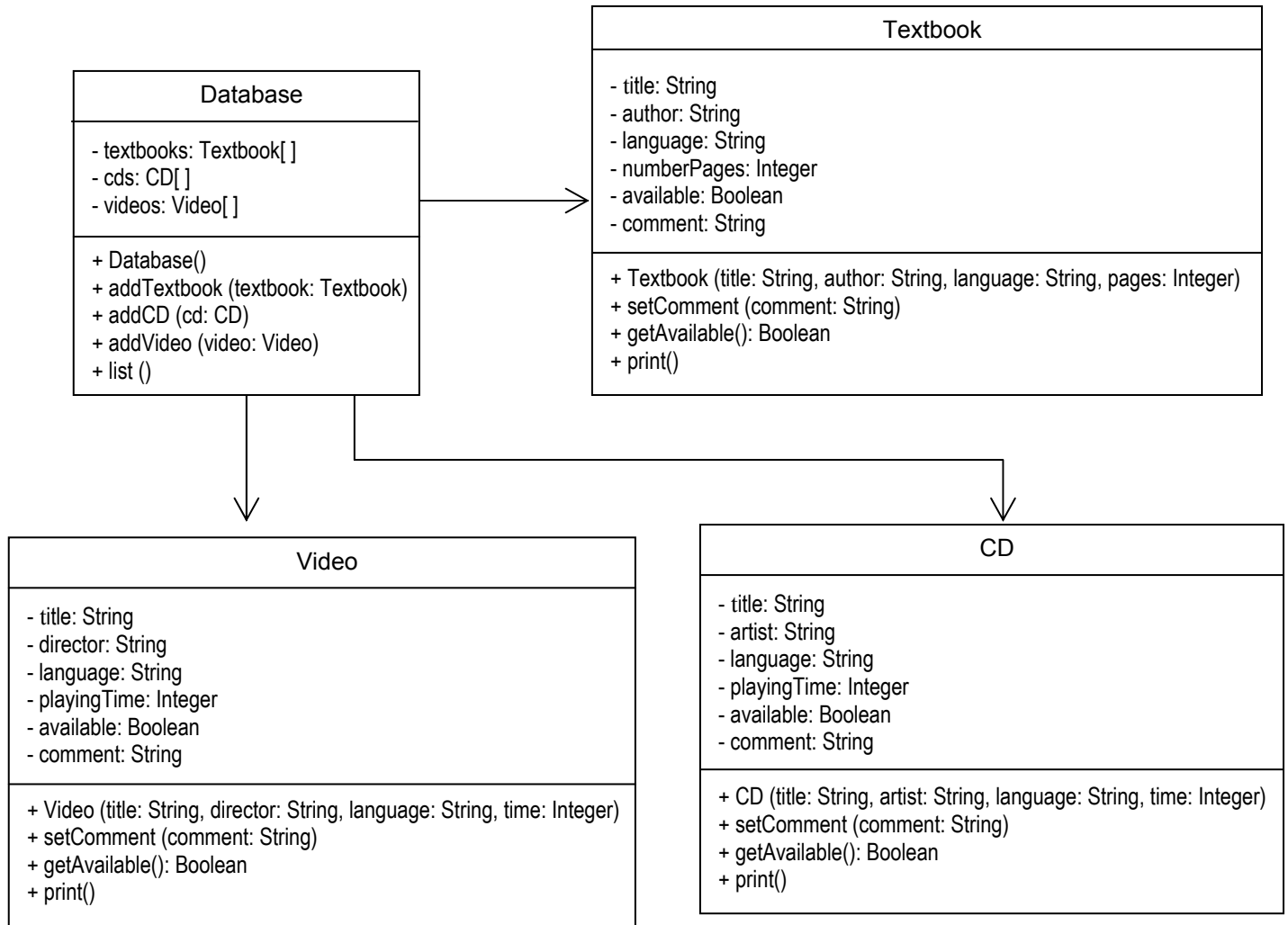
48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

<p><i>Program maintenance</i></p> <p>What if they now want to replace all the CDs with DVDs?</p> <p>How easy is it to change the code?</p>		
<p><i>Program extensibility</i></p> <p>How easy would it be if a new type of resource (e.g. magazine) needs to be recorded?</p> <p>What changes need to be done?</p>		
<p><i>Code readability</i></p> <p>Does polymorphism facilitate reading the code of the system?</p>		

Table 1. Comparison of solutions

Solution 1



```

import java.util.ArrayList;
import java.util.Iterator;

/**
 * The database class provides a facility to store Book, Video and CD
 * objects. A list of all books, CDs and videos can be printed to the
 * terminal.
 */
public class Database
{
    private ArrayList textbooks;
    private ArrayList cds;
    private ArrayList videos;

    /**
     * Construct an empty Database.
     */
    public Database()
    {
        textbooks = new ArrayList();
        cds = new ArrayList();
        videos = new ArrayList();
    }
}
/**

```

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
    * Add a textbook to the database.
    */
public void addTextbook(Textbook textbook)
{
    textbooks.add(textbook);
}

/**
 * Add a CD to the database.
 */
public void addCD(CD cd)
{
    cds.add(cd);
}

/**
 * Add a video to the database.
 */
public void addVideo(Video video)
{
    videos.add(video);
}

/**
 * Print a list of all currently stored CDs and videos to the text terminal.
 */
public void list()
{
    // print list of textbooks
    for(Iterator iter = textbooks.iterator(); iter.hasNext(); ) {
        Textbook textbook = (Textbook)iter.next();
        textbook.print();
        System.out.println();    // empty line between items
    }
    // print list of CDs
    for(Iterator iter = cds.iterator(); iter.hasNext(); ) {
        CD cd = (CD)iter.next();
        cd.print();
        System.out.println();    // empty line between items
    }
    // print list of videos
    for(Iterator iter = videos.iterator(); iter.hasNext(); ) {
        Video video = (Video)iter.next();
        video.print();
        System.out.println();    // empty line between items
    }
}

}

/*****
 * The Textbook class represents a textbook object.
 *****/
public class Textbook
{
    private String title;
    private String author;
    private String language;
    private int numberPages;
    private boolean available;
    private String comment;

    /**
     * Initialize the Book.
     */
    public Textbook(String title, String author, String language, int pages)
    {
        this.title = title;
        this.author = author;
        this.language = language;
        numberPages = pages;
        available = true;
    }
}
```

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
        comment = "<no comment>";
    }
    /**
     * Enter a comment for this Book.
     */
    public void setComment(String comment)
    {
        this.comment = comment;
    }

    /**
     * Return the flag indicating whether this book is available.
     */
    public boolean getAvailable()
    {
        return available;
    }

    /**
     * Print details about this book to the text terminal.
     */
    public void print()
    {
        System.out.print("Title: " + title);
        if(available) System.out.println("");
        else System.out.println();
        System.out.println("Comment:  " + comment);
        System.out.println("Language: " + language);
        System.out.println("Number Pages: " + numberPages + " pages");
        System.out.println("Author: " + author);
    }
}

/*****
 * The CD class represents a CD object.
 *****/
public class CD
{
    private String title;
    private String artist;
    private String language;
    private int playingTime;
    private boolean available;
    private String comment;

    public CD(String title, String artist, String language, int time)
    {
        this.title = title;
        this.artist = artist;
        this.language = language;
        playingTime = time;
        available = true;
        comment = "<no comment>";
    }

    public void setComment(String comment)
    {
        this.comment = comment;
    }

    public boolean getAvailable()
    {
        return available;
    }

    public void print()
    {
        System.out.print("Title: " + title);
        if(available) System.out.println("");
        else System.out.println();
        System.out.println("Comment:  " + comment);
        System.out.println("Language: " + language);
        System.out.println("Playing time: " + playingTime + " mins");
    }
}
```

```
        System.out.println("Artist: " + artist);
    }
}

/*****
 * The Video class represents a video object.
 * *****/
public class Video
{
    private String title;
    private String director;
    private String language;
    private int playingTime;
    private boolean available;
    private String comment;

    public Video(String title, String director, String language, int time)
    {
        this.title = title;
        this.director = director;
        this.language = language;
        playingTime = time;
        available = true;
        comment = "<no comment>";
    }

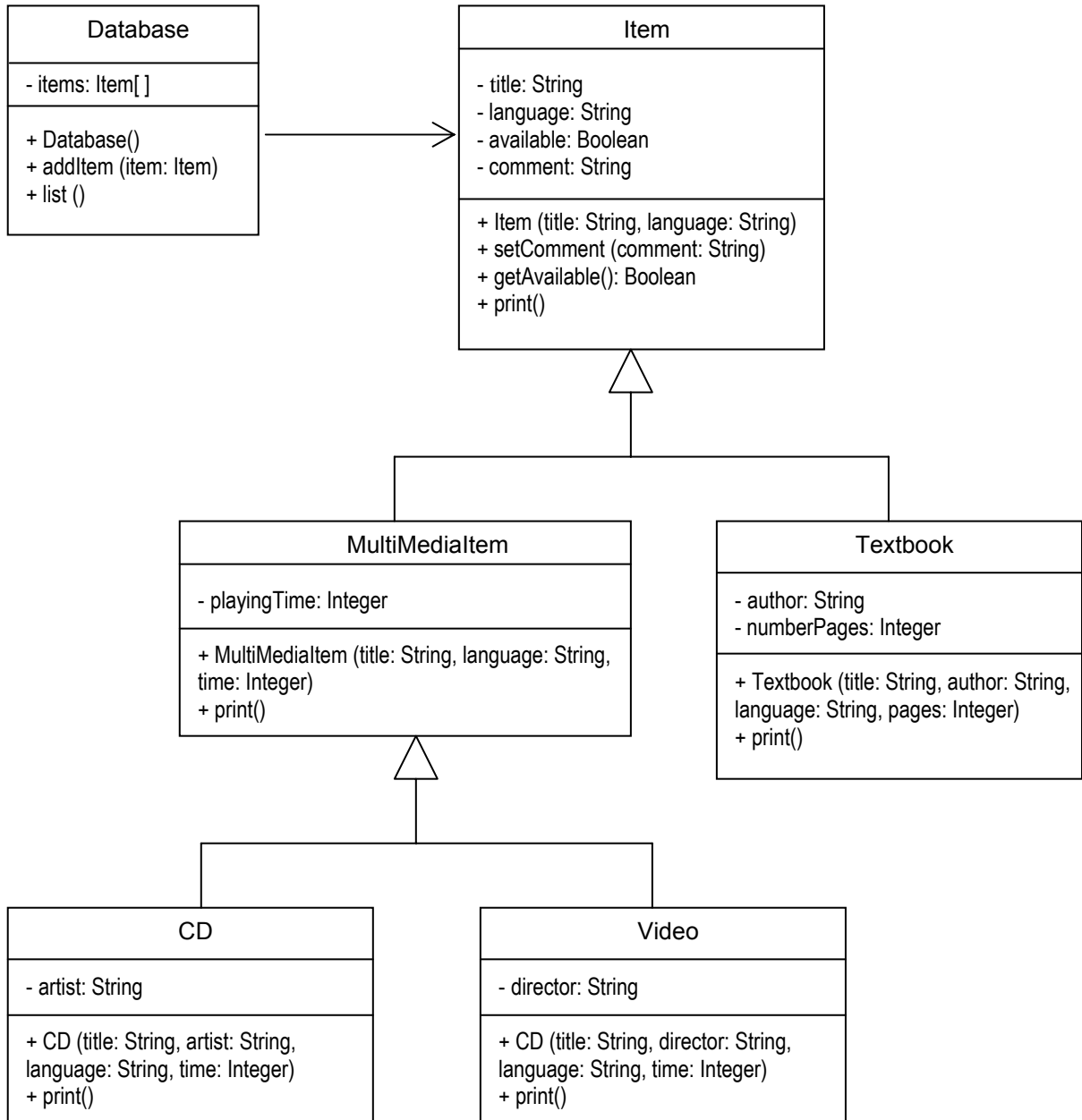
    public void setComment(String comment)
    {
        this.comment = comment;
    }

    public boolean getAvailable()
    {
        return available;
    }

    public void print()
    {
        System.out.print("Title: " + title);
        if(available) System.out.println("*");
        else System.out.println();
        System.out.println("Comment: " + comment);
        System.out.println("Language: " + language);
        System.out.println("Playing time: " + playingTime + " mins");
        System.out.println("Director: " + director);
    }
}
```

Self-Study Module: Inheritance and Polymorphism

Solution 2



48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
import java.util.ArrayList;
import java.util.Iterator;

/*****
 * The database class provides a facility to store CD and video
 * objects. A list of all textbooks, CDs and videos can be printed to the terminal.
 *****/
public class Database
{
    private ArrayList items;

    public Database()
    {
        items = new ArrayList();
    }

    public void addItem(Item item)
    {
        items.add(item);
    }

    /**
     * Print a list of all currently stored CDs and videos to the
     * text terminal.
     */
    public void list()
    {
        for(Iterator iter = items.iterator(); iter.hasNext(); )
        {
            Item item = (Item)iter.next();
            item.print();
            System.out.println(); // empty line between items
        }
    }
}

/*****
 * The Item class represents an item in LIOU.
 * This class serves as a superclass for more specific items.
 *****/
public class Item
{
    private String title;
    private boolean available;
    private String comment;
    private String language;

    public Item(String title, String language)
    {
        this.title = title;
        available = true;
        this.language = language;
        comment = "<no comment>";
    }

    public void setComment(String comment)
    {
        this.comment = comment;
    }

    public boolean available()
    {
        return available;
    }

    public void print()
    {
        System.out.print("Title: " + title);
        if(available) System.out.println("**");
        else System.out.println();
    }
}
```

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
        System.out.println("Comment: " + comment);
        System.out.println("Language: " + language);
    }
}

/*****
 * This subclass of the class Item represents a textbook object.
 *****/
public class Textbook extends Item
{
    private String author;
    private int numberPages;

    public Textbook(String title, String author, String language, int pages)
    {
        super(title, language);
        this.author = author;
        numberPages = pages;
    }

    public void print()
    {
        super.print();
        System.out.println("Number of Pages: " + numberPages + "pages");
        System.out.println("Author: " + author);
    }
}

/*****
 * This subclass of Item represents multi-media resources.
 *****/
public class MultiMediaItem extends Item
{
    private int playingTime;

    public MultiMediaItem(String title, String language, int time)
    {
        super(title, language);
        playingTime = time;
    }

    public void print()
    {
        super.print();
        System.out.println("Playing Time: " + playingTime + "mins");
    }
}

/*****
 * This subclass of MultiMediaItem represents a CD object.
 *****/
public class CD extends MultiMediaItem
{
    private String artist;

    public CD(String title, String artist, String language, int time)
    {
        super(title, language, time);
        this.artist = artist;
    }

    public void print()
    {
        super.print();
        System.out.println("Artist: " + artist);
    }
}

/*****
```

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
* This subclass of the class MultiMediaItem represents a video object.
*****/

public class Video extends MultiMediaItem
{
    private String director;

    public Video(String theTitle, String theDirector, String theLanguage, int time)
    {
        super(theTitle, theLanguage, time);
        director = theDirector;
    }

    public String getDirector()
    {
        return director;
    }

    public void print()
    {
        super.print();
        System.out.println("Director: " + director);
    }
}
```

Exercise 2 - Examining inheritance in Java

This exercise uses excerpts of the code of the Pigworld program used in the Object-Oriented Programming subject several semesters ago. Read the code and draw an inheritance chart for the Pigworld program.

Based on the code, answer the following questions:

1. Why are some of the methods in the Animal class *protected*? For example, the useEnergy method.
2. Which attributes are exclusive to the Thing class, and are *not* inherited by its descendants?
3. Which methods does the Wolf class inherit? Does it explicitly make use of any of these inherited methods itself?
4. Why would the designer have chosen to make the Thing and Animal classes *abstract* classes? Examine the code and discuss the possible reasons for the design decision.
5. Which methods are abstract methods and why?
6. Which methods are overridden?

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

Partial Pigworld Code Listing

Some comments, methods, and attributes have been deleted from the original version of this code listing. Only the relevant parts for the exercise have been preserved. This code will not compile as it is not complete.

```
/*
 * @author Ryan Heise
 */
public class World extends Panel
{
    private Cell[][] cells;
    private List things;

    public World(int colCount, int rowCount, Frame frame)
    {
        this.colCount = colCount;
        this.rowCount = rowCount;

        cells = new Cell[colCount][rowCount];
        things = new ArrayList();

        //...
    }
}

public class Cell extends Panel
{
    private World world;
    private Point position;
    private Thing occupant;

    Cell(World world, int xpos, int ypos)
    {
        this.world = world;
        position = new Point(xpos, ypos);

        setLayout(new BorderLayout());
    }
    //...
}

public abstract class Thing extends Panel
{
    private World world;
    private Cell currentCell;
    protected int energy;

    protected void addToWorld(World world, int x, int y)
    {
        this.world = world;
        world.addThing(this, x, y);
    }

    public boolean exists()
    {
        return world != null;
    }

    public abstract void doSomething();

    protected void handleTime()
    {
        doSomething();
        repaint();
    }
}
```

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
void destroy()
{
    world = null;
}
//. . .
}

public abstract class Animal extends Thing
{
    private int age;
    private boolean isHungry = true;

    protected void handleTime()
    {
        super.handleTime();
        useEnergy();
        grow();
    }
    private void grow()
    {
        age++;
    }

    public void eat(Thing food)
    {
        getWorld().removeThing(food);
        energy += food.getEnergy();
        if (energy >= stomachFullLevel)
        {
            isHungry = false;
        }
    }

    public void move(Direction direction)
    {
        Cell targetCell = getCurrentCell().getNextCell(direction);
        if (targetCell != null)
            targetCell.take(this);
    }

    public void move(double directionInDegrees)
    {
        move(new Direction(directionInDegrees));
    }

    protected void useEnergy()
    {
        useEnergy(1);
    }

    protected void useEnergy(int amount)
    {
        energy = Math.max(energy - amount, 0);
        if (energy < stomachEmptyLevel)
        {
            isHungry = true;
        }
    }

    public boolean isHungry()
    {
        return isHungry;
    }
    //. . .
}
```

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
public class Pig extends Animal
{
    private Pig mother;
    private Pig father;
    private int tiredness = 0;

    private Pig(World world, int x, int y, Pig mother, Pig father)
    {
        this(world, x, y); // gets another constructor to do the dirty work.
        this.mother = mother; // replaces null values of other constructor
        this.father = father;
    }

    public void doSomething()
    {
        if (isTired())
        {
            rest();
        }
        else if (isHungry())
        {
            lookForFood();
        }
        else
        {
            // do nothing
        }
    }

    public void lookForPig()
    {
        Echo echo = findNearest(Pig.class);

        if ( echo != null )
        {
            double distance = echo.getDistance();
            Direction direction = echo.getDirection();
            Cell targetCell = getCurrentCell().getNextCell(direction);
            Thing nextThing = targetCell.getOccupant();

            if ( nextThing instanceof Pig )
            {
                Pig otherPig = (Pig) nextThing;
                if ( otherPig.loveAttack(this) )
                {
                    useEnergy(stomachEmptyLevel / 2);
                    increaseTiredness(5);
                }
            }
            else if ( nextThing instanceof PigFood )
            {
                eat(nextThing);
                move(direction);
            }
            else
            {
                move(direction);
            }
        }

        } // if echo != null

    } // method lookForPig
}

public class Wolf extends Animal
{
    public Wolf(World world)
```

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

```
{
    // in this context, "this" is a call to the constructor below
    this(world, -1, -1);
}

public Wolf(World world, int x, int y)
{
    // layout this gui component
    setLayout(new BorderLayout());
    add(new WolfDisplay(), BorderLayout.CENTER);
    add(new EnergyDisplay(), BorderLayout.EAST);
    add(new AgeDisplay(), BorderLayout.SOUTH);

    addToWorld(world, x, y);
}

public void doSomething()
{
    lookForPig();    // wolves are relentless!
}
}
```

Self-Study Module: Inheritance and Polymorphism

Exercise 3 - Lab: Inheritance and Polymorphism

Resources:

Recommended reading:

- Wu, Thomas. An Introduction to OOP with Java. Chapter 14: Inheritance and Polymorphism.
- Schildt, Herbert. Java™2 The Complete Reference. Chapter 8: Inheritance.

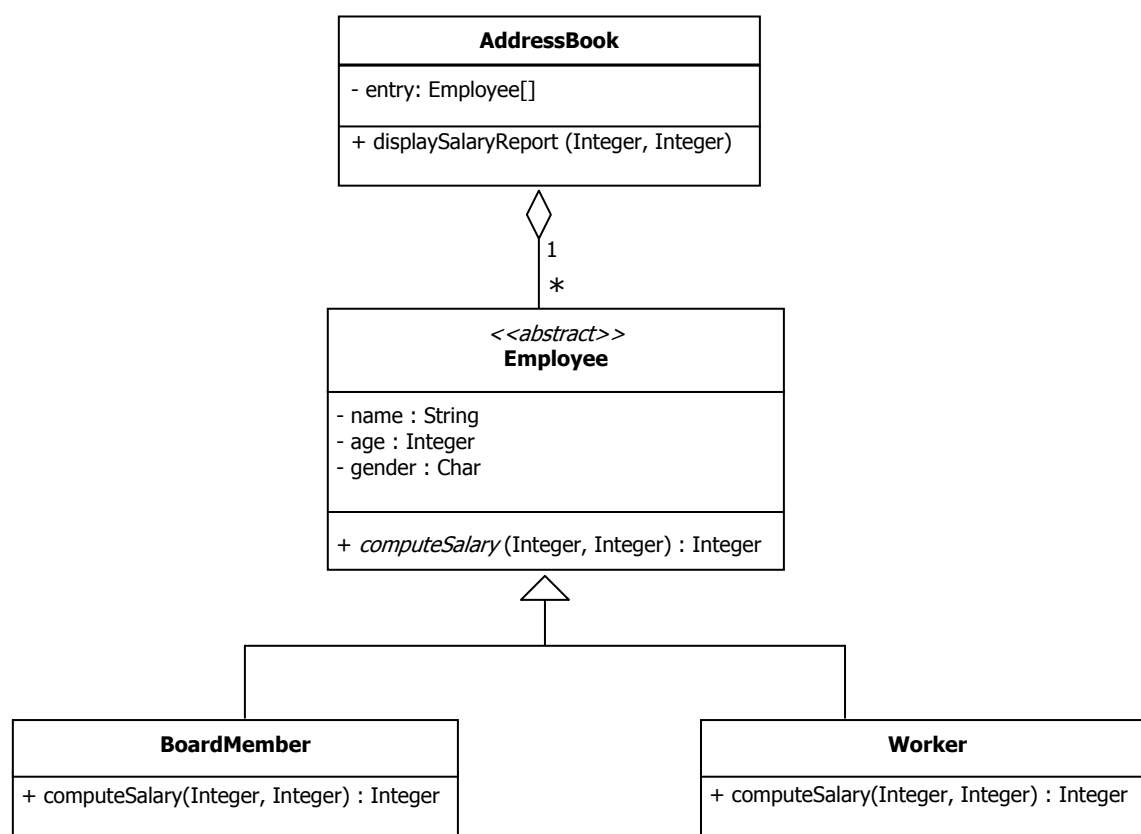
Source Code:

- AddressBookLab7 BlueJ Project (Downloadable from UTSONline OOD Course Documents/ Self-Study Modules).

Task

The same company described in the Laboratory Exercise on Java Collections has decided to categorise their staff in two main classes: those who are part of the management departments and those who work in the factories. This classification is needed due to the different formulas used to calculate the salary of employees of each category.

The IT team has modified the class diagram of the original system to meet the new requirements, getting as a result the next diagram (drawn using UML notation):



Note: Not all the attributes and operations of the classes are included in the class diagram. Check the code of the classes to get all the details.

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

The first visible change is the addition of the method `displaySalaryReport(baseSalary: Integer, days: Integer)` as part of the class `AddressBook`. This method displays on the system output (e.g. BlueJ's terminal window) the list of all the employees with their corresponding salary amount. These salaries are calculated by calling the `computeSalary` method of each one of the objects representing an employee.

As you can see, the `Employee` class has become an *abstract class* that contains all the common attributes and operations an employee has, including a new *abstract method* called `computeSalary` whose job is exactly that: calculate the salary of the employee according to a fixed formula that uses the input values. Two new classes have been created: `BoardMember` and `Worker` being both subclasses of the class `Employee`.

Your task consists of modifying the provided `AddressBook` BlueJ project to fulfil the shown design, including writing the code for the new method `displaySalaryReport`.

The method `computeSalary(baseSalary: Integer, days: Integer): Integer` within each one of the two subclasses (`BoardMember` and `Worker`) must return an integer corresponding to the salary of an employee that is calculated based on the following formulas:

- For the `BoardMember` class : **salary = baseSalary + days*35 + age*100**
- For the `Worker` class : **salary = baseSalary + days*45 + age*50**

where **salary** is the Integer value to be returned by the method; **baseSalary** is the first input parameter, an Integer value indicating the base salary amount for the employees; **days** is the second parameter, an Integer value indicating the number of days the salary employees has been working without a payment; and **age** is the age of the employee whose salary is being calculated.

The `displaySalaryReport(baseSalary: Integer, days: Integer)` method should call the `computeSalary` method of each one of the employee objects the `AddressBook` object contains and display the name of the employee and its corresponding salary, a list similar to:

John Smith	\$ 3500
Jennifer Pole	\$ 2980
Lucia Anaya	\$ 4560

Once you finish use the `Lab7Tester` class to test your modifications.

48024 Object Oriented Design

Self-Study Module: Inheritance and Polymorphism

Review Questions

1. Why did the designers choose to make computeSalary an abstract method?
2. With the current design, would it be possible to create the report without using Java Polymorphism? Explain your answer.
3. What does the following Java class declaration line mean?
`class MyFirstFrame extends Frame implements ActionListener`