# Tutorial 7: Abstract Classes and Interfaces

1. Consider the following UML diagram

|  |
| --- |
| *GeometricObject* |
| #color: String |
| #GeometricObject() #GeometricObject(String)  +getColor(): String  +setColor(String): void  *+findArea(): double* |

|  |
| --- |
|  |
| Circle |
| –radius : double |
| +Circle(double, String)  +getRadius(): double  +setRadius(double): void  +findArea(): double |

|  |  |
| --- | --- |
|  |  |
| Cylinder | |
| –length : double | |
| +Cylinder()  +Cylinder(double, double, String)  +findVolume(): double | |

* 1. Write the class declaration for the Cylinder class.

*Note: Volume of cylinder = area of circle \* length*

public class Cylinder extends Circle {

    private double length;

    public Cylinder() {

    }

    public Cylinder(double length, double radius, String color) {

        super(length, color);

        this.length = length;

    }

    public double findVolume() {

        return super.findArea() \* length;

    }

}

* 1. Determine the output of the following program segment. Explain your answer.

Circle x = new Cylinder(); if (x instanceof Circle)

System.out.println(“X is a circle”); else

System.out.println(“X is a cylinder”);

* X is a circle
* As the Cylinder class is a subclass of the Circle class, so Cylinder object is a Circle object. Therefore, the instanceof operator will evaluate to a true.
  1. Assuming that the class Rectangle is a child class of GeometricObject, explain why the following statements are invalid.

Circle object1 = new Rectangle(); Circle object2 = new GeometricObject();

* A circle object reference cannot refer to an object of its superclass or a sibling class. An object reference may refer only to an object of its subclass.

1. What are the two main effects of declaring a method as abstract?

* The class must also be declared abstract.
* Any child classes must either override the abstract method or else be declared as abstract.

1. A local college provides basic salary plus various allowances for their academic staff. In this salary scheme, academic staff members are entitled to an amount of total monthly salary based on their staff level (**JL**-junior lecturer, **SL**-senior lecturer and **PS**-program supervisor), education level (**Degree**, **Master** and **PhD**) and year of service.

# Junior Lecturer (JL)

* Degree holder: Basic salary is RM1600 per month.
* Master holder: Basic salary is RM2300 per month.
* PhD holder: Basic salary is RM3000 per month.
* Transportation allowance for junior lecturer is RM220 per month.

# Senior Lecturer (SL)

* Degree holder: Basic salary is RM2300 per month.
* Master holder: Basic salary is RM3000 per month.
* PhD holder: Basic salary is RM3500 per month.
* Transportation allowance for senior lecturer is RM270 per month.

# Program Supervisor (PS)

* Basic salary is RM3500 per month (regardless of education level).
* Transportation allowance for program supervisor is RM300 per month.
* Administrative allowance for program supervisor is RM400 per month.

# General Allowances:

* Entertainment allowance of RM200 per month for all academic staff who have serviced the college more than or equal to 8 years.
* Housing allowance of RM300 per month for all academic staff who have serviced the college more than or equal to 10 years.
  1. Create an abstract superclass named AcademicStaff that stores the common data members. You should include the following in the class:
     + Protected data members.
     + ONE constructor with parameters.
     + *Get* and *set* methods for all data members.
     + Method calcEntertainmentAllowance(): double
     + Method calcHousingAllowance(): double
     + An abstract displayDetail():void method
  2. Create **TWO (2)** subclasses, Lecturer and ProgramSupervisor that inherit from the

AcademicStaff class.

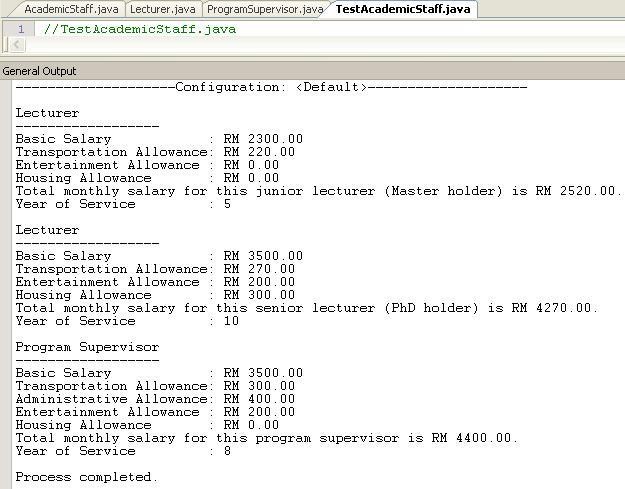
Identify the additional data members needed in each subclass and ensure that the following are included in each of the subclasses.

* + - Private data member(s) (if any).
    - Constructor with parameters.
    - *Get* and *set* methods for data member(s)
    - An overridden displayDetail() method (refer to sample output as shown in Figure 1)
  1. Write a program segment to demonstrate the capabilities of the classes you have defined in

1. and (b) above.

Your program should declare an array of **THREE (3)** objects and display the total monthly salary for each level of academic staff accordingly. A sample output of your program is shown in Figure 1.

# Figure 1



// AcademicStaff.java

public abstract class AcademicStaff {

    protected int yearOfService;

    public AcademicStaff(int year) {

        yearOfService = year;

    }

    public void setYearOfService(int year) {

        yearOfService = year;

    }

    public int getYearOfService() {

        return yearOfService;

    }

    public double calcEntertainmentAllowance() {

        double entertainmentAllowance;

        if (getYearOfService() >= 8) {

            entertainmentAllowance = 200;

        } else {

            entertainmentAllowance = 0;

        }

        return entertainmentAllowance;

    }

    public double calcHousingAllowance() {

        double housingAllowance;

        if (getYearOfService() >= 10) {

            housingAllowance = 300;

        } else {

            housingAllowance = 0;

        }

        return housingAllowance;

    }

    public abstract void displayDetail();

}

// Lecturer.java

public class Lecturer extends AcademicStaff {

    private String staffLevel;

    private String educationLevel;

    public Lecturer(String staffL, String educationL, int year) {

        super(year);

        setStaffLevel(staffL);

        setEducationLevel(educationL);

    }

    public void setStaffLevel(String level) {

        staffLevel = level;

    }

    public String getStaffLevel() {

        return staffLevel;

    }

    public void setEducationLevel(String level) {

        educationLevel = level;

    }

    public String getEducationLevel() {

        return educationLevel;

    }

    public void displayDetail() {

        double basicSalary = 0;

        double transportationAllowance = 0;

        String level = "";

        if (getStaffLevel().equals("JL")) {

            if (educationLevel.equals("Degree")) {

                basicSalary = 1600;

            } else if (educationLevel.equals("Master")) {

                basicSalary = 2300;

            } else if (educationLevel.equals("PhD")) {

                basicSalary = 3000;

            }

            transportationAllowance = 220;

            level = "Junior Lecturer";

        }

        else if (getStaffLevel().equals("SL")) {

            if (educationLevel.equals("Degree")) {

                basicSalary = 2300;

            } else if (educationLevel.equals("Master")) {

                basicSalary = 3000;

            } else if (educationLevel.equals("PhD")) {

                basicSalary = 3500;

            }

            transportationAllowance = 270;

            level = "Senior Lecturer";

        }

        double totalSalary = basicSalary + transportationAllowance +

                super.calcEntertainmentAllowance() + super.calcHousingAllowance();

        System.out.printf("Basic Salary : RM %.2f\n", basicSalary);

        System.out.printf("Transportation Allowance: RM %.2f\n", transportationAllowance);

        System.out.printf("Entertainment Allowance : RM %.2f\n",

                super.calcEntertainmentAllowance());

        System.out.printf("Housing Allowance : RM %.2f\n", super.calcHousingAllowance());

        System.out.printf("Total monthly salary for this %s (%s holder) is RM %.2f.\n", level, this.getEducationLevel(),

                totalSalary);

        System.out.printf("Year of Service : %d\n", super.getYearOfService());

    }

}

// ProgramSupervisor.java

public class ProgramSupervisor extends AcademicStaff {

    public ProgramSupervisor(int year) {

        super(year);

    }

    public void displayDetail() {

        double basicSalary = 3500;

        double transportationAllowance = 300;

        double administrativeAllowance = 400;

        double totalSalary = basicSalary + transportationAllowance + administrativeAllowance +

                super.calcEntertainmentAllowance() + super.calcHousingAllowance();

        System.out.printf("Basic Salary : RM %.2f\n", basicSalary);

        System.out.printf("Transportation Allowance: RM %.2f\n", transportationAllowance);

        System.out.printf("Administrative Allowance: RM %.2f\n", administrativeAllowance);

        System.out.printf("Entertainment Allowance : RM %.2f\n",

                super.calcEntertainmentAllowance());

        System.out.printf("Housing Allowance : RM %.2f\n", super.calcHousingAllowance());

        System.out.printf("Total monthly salary for this program supervisor is RM %.2f.\n", totalSalary);

        System.out.printf("Year of Service : %d\n", super.getYearOfService());

    }

}

// tutorial7q3.java

public class tutorial7q3 {

    public static void main(String[] args) {

        AcademicStaff Staff[] = new AcademicStaff[3];

        Staff[0] = new Lecturer("JL", "Master", 5);

        Staff[1] = new Lecturer("SL", "PhD", 10);

        Staff[2] = new ProgramSupervisor(8);

        for (int x = 0; x < Staff.length; x++) {

            if (Staff[x] instanceof Lecturer) {

                System.out.println("\nLecturer");

            } else if (Staff[x] instanceof ProgramSupervisor) {

                System.out.println("\nProgram Supervisor");

            }

            System.out.println("------------------");

            Staff[x].displayDetail();

        }

        System.out.println("\nProcess completed.");

    }

}

1. Briefly describe the following core object-oriented concepts:
   1. Encapsulation

* Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
  1. Inheritance
* Inheritance is the process by which one object acquires the properties of another object. By use of inheritance, an object would need only to define those qualities that make it unique within its class. It can inherit its general attributes from its parent.
  1. Polymorphism
* Polymorphism is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.

1. What is wrong with the interface declaration below? Write the correct interface declaration.

public interface MyInterface{ void aMethod(int value){

System.out.println(“Hi, Mom”);

}

}

* Interface can only have abstract methods.

public interface MyInterface {

    public abstract void aMethod(int value);

}

1. Explain the differences between *interface* and *abstract class*.

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| A class may extend only one abstract class. | A class may implement several interfaces. |
| Can extend another abstract or non-abstract  class. | Can extend only another interface. |
| Can have methods with bodies defined as well  as methods without bodies. | Cannot have any method with a defined  body. |
| Can contain both instance variables and constants. | Can contain only constants. |

1. What is the result of attempting to compile the following program:
2. interface A{
3. int num=5;
4. private void sub2();
5. abstract void sub3(); 5. }

6.

1. class B extends A {} 8.
2. class C implements A {
3. public void sub2(){ }
4. void sub3() { }

12. }

Error in line 3: modifier private not allowed for interface

Error in line 7: should use the keyword *implements* rather than *extends*

Error in line 11: attempting to assign weaker access privileges

1. (a) Create an interface named OutOfService with a void method named

performRepair().

public interface OutOfService {

    public abstract void performRepair();

}

1. Create a class named Machine that inherits from an Item class and also the OutOfService interface. The performRepair() method should display an appropriate message indicating the status of the repair work.

public class Machine extends Item implements OutOfService {

    public void performRepair() {

        System.out.println("In progress");

    }

}