SEMESTER 5 OF SESSION 22/23 (A231)
SKIP3013 CONCEPT AND PARADIGM IN PROGRAMMING LANGUAGES
Project: Object-Oriented Programming languages

Group:

PREPARED BY:

NAME	MATRIC NUMBER
MUHAMMAD IMAN ZULHAKIM BIN MOHD ZULKHORI	288430
MOHAMAD TAHFIZZUDEEN BIN MOHD HANAPI	288334
MUHAMMAD ALIF BIN MUHAMMAD IBRAHIM	287814
FADZLI NAIM BIN JAFYUSRI	287855

PREPARED FOR:
ASSOC. PROF. DR. ROSHIDI BIN DIN

SKIP3013 CONCEPT AND PARADIGM IN PROGRAMMING LANGUAGES
Project: Object-Oriented Programming languages

MUHAMMAD IMAN ZULHAKIM BIN MOHD ZULKHORI (288430)
MOHAMAD TAHFIZZUDEEN BIN MOHD HANAPI (288334)
MUHAMMAD ALIF BIN MUHAMMAD IBRAHIM (287814)
FADZLI NAIM BIN JAFYUSRI (287855)

ASSOC. PROF. DR. ROSHIDI BIN DIN
School Of Computing,
College of Art & Sciences,
Universiti Utara Malaysia.

1. **Introduction:**
    a. The Zoo Management System is an application built in Java that demonstrates important Object-Oriented Programming (OOP) ideas in a real-world setting. Modularity, reusability, and simplicity are the guiding concepts of this system's modelling of a simplified zoo setting. This application is designed to showcase the practical use of object-oriented programming (OOP) principles by creating classes and interfaces.

2. **Objective:**
    a. The Zoo Management System follows SOLID principles and uses basic object-oriented programming concepts such classes, encapsulation, abstraction, inheritance, polymorphism, and interfaces. This project's overarching goal is to clarify these ideas and show how to apply them in a Java-based setting.

3. **Scope:**
    a. The goal of the Zoo Management System is to provide an expandable and modular system that can manage the fundamental data about zoo animals. Classes depicting animals, particularly lions, are designed and implemented in the system. The system also shows how feeding behaviour interfaces and polymorphism improve code flexibility and maintainability.

4. System Development:

a. Class Design and Implementation:

i. **Understanding**:
Classes are the backbone of any object-oriented programme. To represent zoo-related things, we have created and implemented the Animal, Lion, and LionFeeding classes in this project.

ii. **Implementation**:
Animal behaviour and properties are encapsulated in the classes through the use of attributes and methods. The system's entities will be represented in a clear and organised manner because of this.

```
// 1. Class Design and Implementation
class Animal {
    String name;
    int age;
```

Figure 1: Implementation of Class Design and Implementation

b. Encapsulation and Abstraction:

i. **Understanding**:
Methods that operate on data are bundled together in an encapsulated unit. The act of giving a simplified picture of an entity is called abstraction. The makeSound method in the Animal class exemplifies abstraction, whereas private access modifiers accomplish encapsulation in our project.

ii. **Implementation**:
Data is encapsulated using private access modifiers, and an abstraction of the animal's sound is provided by the makeSound method.

```
    // 2. Encapsulation and Abstraction
    // Encapsulation: Data hiding using private access modifiers
    // Abstraction: Providing a simple interface for the complex underlying implementation
    public void makeSound() {
        System.out.println("Generic animal sound");
    }
}
```

Figure 2: Implementation of Encapsulation and Abstraction

c. Inheritance:
    i. **Understanding**:
    A class can take on the characteristics and actions of another class through inheritance. As an example of a hierarchical connection, the Lion class derives from the Animal class.
    ii. **Implementation**:
    The Lion class takes its methods and properties from the Animal class, which it extends.

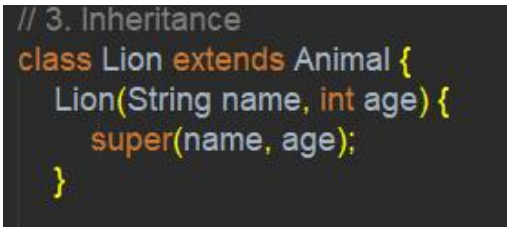

Figure 3: Implementation of the Inheritance

d. **Polymorphism and Interfaces**:
    i. Polymorphism
        1. **Understanding**:
        In Java, polymorphism is defined as having a common way to do things differently. Assume you have a group of animals and you want them to produce a sound. Polymorphism enables you to have a single "makeSound" method for all animals, while each species can produce its own sound. So, a lion can roar, a snake can hiss, and a chicken can cluck, all with the same "makeSound" method.

        2. **Implementation**:
        The Lion class overrides and inherits the makeSound() function from the Animal class. This is a classic example of polymorphism, in which a subclass provides a customised implementation of a method that has already been specified in the superclass.

            - In Animal, makeSound() returns "Generic animal sound."
            - In Lion, makeSound() is overrided to output "Roar!".

```java
package learnjava;

// 1. Class Design and Implementation
class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // 2. Encapsulation and Abstraction
    // Encapsulation: Data hiding using private access modifiers
    // Abstraction: Providing a simple interface for the complex underlying implementation
    public void makeSound() {
        System.out.println("Generic animal sound");
    }
}

// 3. Inheritance
class Lion extends Animal {
    Lion(String name, int age) {
        super(name, age);
    }

    // 4. Polymorphism and Interfaces
    // Polymorphism: Overriding the makeSound method in the Lion class
    // Interfaces: Implementing an interface for feeding behavior
    public void makeSound() {
        System.out.println("Roar!");
    }
}
```

makeSound() in Lion class override makeSound() in Animal class

Figure 4: Implementation of the Polymorphism.

ii. Interfaces

1. **Understanding**:
   The capabilities of a class in Java are defined by its interface, which is similar to a contract. Consider these a manual of sorts. To illustrate, imagine a "feed" rule in a "Feeding" interface. There needs to be a "feed" method in every class that uses this interface. It's comparable to stating, "If you want to feed an animal, you must know what animal to feed .".

2. **Implementation**:
   An interface Feeding was defined with a method named feed(). This interface was implemented by the LionFeeding class. Doing so guarantees that every class that uses the Feeding interface will have a feed() method.

```java
// Interface for feeding behavior
interface Feeding {
    void feed();
}

// Implementing feeding behavior for Lion
class LionFeeding implements Feeding {
    public void feed() {
        System.out.println("Feeding the lion");
    }
}
```

Figure 5: Implementation of Interface.

e. SOLID Principle:
   i. As a framework for creating scalable and maintainable software, the SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) are essential. A strong and extensible system is guaranteed by the project's adherence to these principles.
   ii. Code-wise, the classes and interfaces follow the Liskov Substitution Principle, are purpose-built to handle a particular task at a time, and are extensible but immutable.

f. User Interaction:
   i. Showing how the system can be used in a real-world setting, user interaction includes the main class that produces instances of the defined classes and interfaces.
   ii. Coding Demo: The ZooManagementSystem class shows how to communicate with the user by making instances, showing data, and calling functions.

```
public class ZooManagementSystem {
    public static void main(String[] args) {
        // Creating instances of Animal and Lion classes
        Animal genericAnimal = new Animal("Generic Animal", 5);
        Lion lion = new Lion("Simba", 3);

        // Displaying information and behavior
        System.out.println("Generic Animal Information:");
        System.out.println("Name: " + genericAnimal.name);
        System.out.println("Age: " + genericAnimal.age);
        genericAnimal.makeSound();
        System.out.println();

        System.out.println("Lion Information:");
        System.out.println("Name: " + lion.name);
        System.out.println("Age: " + lion.age);
        lion.makeSound();

        // Using Interface for feeding behavior
        LionFeeding lionFeeding = new LionFeeding();
        lionFeeding.feed();
    }
}
```

Figure 6: Main Class

```
Generic Animal Information:
Name: Generic Animal
Age: 5
Generic animal sound

Lion Information:
Name: Simba
Age: 3
Roar!
Feeding the lion
```

Figure 7: Output

5. Conclusion:

a. To sum up, the Zoo Management System project is a great example of how to use Object-Oriented Programming principles to build a flexible and customisable system. This project demonstrates basic OOP concepts in action by using classes, interfaces, and following the SOLID methodology. The user-friendliness and efficacy of the proposed solution in zoo administration are demonstrated through the system's interaction. In sum, the project is a teaching tool for object-oriented programming (OOP) in Java.

6. Code
    a. Animal.java

```
// 1. Class Design and Implementation
class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // 2. Encapsulation and Abstraction
    // Encapsulation: Data hiding using private access modifiers
    // Abstraction: Providing a simple interface for the complex underlying
implementation
    public void makeSound() {
        System.out.println("Generic animal sound");
    }
}

// 3. Inheritance
class Lion extends Animal {
    Lion(String name, int age) {
        super(name, age);
    }


    // 4. Polymorphism and Interfaces
    // Polymorphism: Overriding the makeSound method in the Lion class
    // Interfaces: Implementing an interface for feeding behavior
    public void makeSound() {
        System.out.println("Roar!");
    }
}

// Interface for feeding behavior
interface Feeding {
    void feed();
}

// Implementing feeding behavior for Lion
class LionFeeding implements Feeding {
    public void feed() {
        System.out.println("Feeding the lion");
    }
}
```

b. ZooManagementSystem.java(Main Class)

```
public class ZooManagementSystem {
    public static void main(String[] args) {
        // Creating instances of Animal and Lion classes
        Animal genericAnimal = new Animal("Generic Animal", 5);
        Lion lion = new Lion("Simba", 3);

        // Displaying information and behavior
        System.out.println("Generic Animal Information:");
        System.out.println("Name: " + genericAnimal.name);
        System.out.println("Age: " + genericAnimal.age);
        genericAnimal.makeSound();
        System.out.println();

        System.out.println("Lion Information:");
        System.out.println("Name: " + lion.name);
        System.out.println("Age: " + lion.age);
        lion.makeSound();

        // Using Interface for feeding behavior
        LionFeeding lionFeeding = new LionFeeding();
        lionFeeding.feed();
    }
}
```