LP2A Written Exam

Thursday May 15th, 2026
Duration 1h30
No documents

Exercise #0 Back to the lectures (4 points)

Among the 4 pillars of Object Oriented Programming (OOP), which one(s) is (are), to your opinion, most distinguish OOP from imperative programming (such as with C language for instance)?

Exercise #1 Is it dynamic binding or not? (12 points)

For this Exercise the code is available in annexe

Question #1 (1pt): Explain why the makeNoise() method is declared as abstract in the Animal class. What would happen if it were not abstract?

Question #2 (1pt): Describe how encapsulation is implemented in the Animal class. Identify all private fields and explain how access is controlled through public methods.

Question #3 (1 pt): The setAge(int age) method checks that age > 0. What OOP principle does this demonstrate? Why is this validation important?

Question #4 (1 pt): When makeNoise() is called on an Animal reference pointing to a Dog, why is the Dog's version executed instead of the Animal's? Explain the concept involved.

Question #5 (1 pt): Why can't the methods play() and purr() be called directly on an Animal variable? Provide a code example where such a call would fail at compile time.

Question #6 (1 pt): The eat() method is not abstract in Animal, but all animals eat. Discuss the advantages and disadvantages of having a concrete eat() method in the abstract class.

Question #7 (3 pt): Now imagine we have a Human class has a method feed(Animal animal). Explain how this demonstrates a key OOP principle. Why is this approach more secure than direct method calls to specific animal types? Write this Human class using the already existing classes.

Question #8 (3 pt): What is actually the output of the main program?

Exercise #2 Comparator (5 points)

You can find below the code corresponding to a class named Person which is holding data and methods corresponding to the type person and a class SortPeople which is aimed at sorting out a ArrayList of Person. We want the code to be as flexible as possible so that we decided to use a class PersonComparator which can be used as a parameter of the sort method from the Collections.

```
class Person {
    private String name;
    private int age;
    private double salary;

    public Person(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    // Getters
    public String getName() { return name; }
    public int getAge() { return age; }
    public double getSalary() { return salary; }

    @Override
    public String toString() {
        return String.format("%s (%d, $%.2f)", name, age, salary);
    }
}
```

```
import java.util.*;

public class SortPeople {

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

```
        List<Person> people = new ArrayList<>();
        people.add(new Person("Alice", 25, 50000));
        people.add(new Person("Bob", 25, 60000));
        people.add(new Person("Charlie", 30, 55000));
        people.add(new Person("David", 35, 45000));

        // Use the dedicated comparator class
        PersonComparator comparator = new PersonComparator();
        Collections.sort(people, comparator);

        // Print sorted list
        for (Person p : people) {
            System.out.println(p);
        }
    }
}
```

Javadoc for the Comparator interface

public interface Comparator<T>

A comparison function, which imposes a *total ordering* on some collection of objects. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps), or to provide an ordering for collections of objects that don't have a natural ordering.

int compare(T o1, T o2)

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

Question #1 (3 pts): Write the class PersonComparator which is implementing the Comparator Interface and is comparing Person by their age.

Question #2 (2 pts): Imagine we need to compare the persons by their salary, what should we do?

Annexe Code for Exercise #1

```
abstract class Animal {
    private String name;
    private int age;
    private boolean isDomestic;

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

    public String getName() { return name; }
    public int getAge() { return age; }
    public boolean isDomestic() { return isDomestic; }

    public void setAge(int age) {
        if (age > 0) this.age = age;
    }

    public abstract void makeNoise();

    public void eat() {
        System.out.println(this.getName() + " eats");
    }

    public void sleep() {
        System.out.println(this.getName() + " sleeps");
    }
}
```

```
class Dog extends Animal {
    private String race;

    public Dog(String name, int age, String race, boolean isDomestic) {
        super(name, age, isDomestic);
        this.race = race;
```

```
    }

    @Override
    public void makeNoise() {
        System.out.println("WOOF!");
    }

    public void play() {
        System.out.println(this.getName() + " plays");
    }

     public String getRace() {
          return race;
     }

     public void setRace(String race) {
          this.race = race;
     }
}
```

```
class Cat extends Animal {
    private boolean isCurious;

    public Cat(String name, int age, boolean isCurious, boolean
isDomestic) {
        super(name, age, isDomestic);
        this.setCurious(isCurious);
    }

    @Override
    public void makeNoise() {
        System.out.println("MEOW!");
    }

    public void purr() {
        System.out.println(this.getName() + " purrs");
    }
```

```
      public boolean isCurious() {
            return isCurious;
      }

      public void setCurious(boolean isCurious) {
            this.isCurious = isCurious;
      }
}
```

```
class Lion extends Animal {
    public Lion(String name, int age, boolean isDomestic) {
        super(name, age, isDomestic);
    }

    @Override
    public void makeNoise() {
        System.out.println("ROAR!");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Animal[] animals = {
            new Dog("Rex", 3, "Labrador", true),
            new Cat("Minou", 2, true, true),
            new Dog("Boby", 5, "Bulldog", true),
            new Cat("Felix", 4, false, true),
            new Lion("Bartoldi", 6, false)
        };

        for (Animal animal : animals) {
            animal.makeNoise();
            animal.eat();
            animal.sleep();
        }
    }
}
```