**Java data types, Enum, etc., and OOPS**

1. Write program for Runtime Polymorphism and Compile time polymorphism

*package* com.mv.week1.part1.questions;  
  
*class* SimpleCalculator {  
  
 *public* SimpleCalculator() {  
 System.out.println("In SimpleCalculator constructor");  
 }  
  
 *public int* add(*int* a, *int* b) {  
 *return* a + b;  
 }  
  
 *public int* add(*int* a, *int* b, *int* c) {  
 *return* a + b + c;  
 }  
  
 *public int* add(*int[]* nums) {  
 *int* n = nums.length;  
 *int* sum = 0;  
  
 *// Simple for loop  
 for* (*int* i = 0; i < n; i++)  
 sum += nums[i];  
 *return* sum;  
 }  
}  
  
*class* Calculator *extends* SimpleCalculator {  
 *public* Calculator() {  
 *// Super must be first statement in derived class constructor body  
 super*();  
 System.out.println("In Calculator constructor");  
 }  
  
 *@Override  
 public int* add(*int[]* nums) {  
 *int* sum = 0;  
  
 *// for each loop  
 for* (*int* x : nums)  
 sum += x;  
 *return* sum;  
 }  
}  
  
*public class* Polymorphism {  
  
 *public static void* main(String*[]* args) {  
 SimpleCalculator calc = *new* SimpleCalculator();  
  
 *// Compile Time polymorphism -> Method Overloading* System.out.println(calc.add(1, 2));  
 System.out.println(calc.add(1, 2, 3));  
  
 *//Runtime Polymorphism -> Method Overriding* SimpleCalculator calc2 = *new* Calculator();  
 System.out.println(calc2.add(*new int*[]{1, 2, 3, 4}));  
  
  
  
 }  
  
  
}

1. Explain why composition is preferred over Inheritance?

Composition is preferred over inheritance for several reasons:

* **Flexibility and Reusability**: With inheritance, a subclass is tightly coupled to its superclass, and it inherits all the properties and methods of the superclass. This can lead to inflexible code that is difficult to reuse or modify. With composition, classes can be composed of smaller, more modular components that can be easily reused and modified.
* **Avoiding the Fragile Base Class Problem**: The Fragile Base Class Problem is a common issue with inheritance, where changes to the superclass can have unintended effects on the subclasses. For example, if the derived class overrides a method in parent class. Any modification to that method in parent class can lead to a problem in a derived class. So, the parent or derived class method will have to changed accordingly to resolve this issue.

With composition, the component can be modified independently of the rest of the codebase, which can reduce the risk of unintended consequences.

* **Separation of Concerns**: Inheritance can lead to code that is tightly coupled and difficult to separate into separate concerns. Composition, on the other hand, allows for more modular and reusable components that can be combined in different ways to achieve different goals.

Overall, composition can lead to more flexible, maintainable, and reusable code than inheritance.

1. Can a class extend Enum? Can Enum implement an interface? What is ordinal in Enum?

Class cannot extend Enum.

*enum* OrderStatus *implements* MyInterface {  
 DISPATCHED,  
 SHIPPED,  
 OUT\_FOR\_DELIVERY,  
 DELIVERED;  
  
 String orderStatus;  
  
 *@Override  
 public void* setStatus(String orderStatus) {  
 *this*.orderStatus = orderStatus;  
 }  
}  
  
*// Cannot extend from enum. Causes compilation error  
//class A extends OrderStatus{  
//  
//}*

Yes, Enum can implement an interface

*interface* MyInterface {  
  
 *public void* setStatus(String orderStatus);  
  
}  
  
*// Enum can implement interface  
enum* OrderStatus *implements* MyInterface {  
 DISPATCHED,  
 SHIPPED,  
 OUT\_FOR\_DELIVERY,  
 DELIVERED;  
  
 String orderStatus;  
  
 *@Override  
 public void* setStatus(String orderStatus) {  
 *this*.orderStatus = orderStatus;  
 }  
}

ordinal() method returns the position of the enum constant.

*enum* OrderStatus *implements* MyInterface {  
 DISPATCHED,  
 SHIPPED,  
 OUT\_FOR\_DELIVERY,  
 DELIVERED;  
  
 String orderStatus;  
  
 *@Override  
 public void* setStatus(String orderStatus) {  
 *this*.orderStatus = orderStatus;  
 }  
}

*// ordinal() method returns the position of the enum constant  
 for* (OrderStatus status : OrderStatus.values())  
 System.out.println(status + " " + status.ordinal());  
*// DISPATCHED 0  
// SHIPPED 1  
// OUT\_FOR\_DELIVERY 2  
// DELIVERED 3*

1. Explain for loop and for each loop. Which is faster?
2. Write a program for enum which has a private constructor which accepts 2 arguments.

*package* com.mv.week1.part1.questions;  
  
*enum* Days {  
 MONDAY(1, "Mon"),  
 TUESDAY(2, "Tue"),  
 WEDNESDAY(3, "Wed"),  
 THURSDAY(4, "Thurs"),  
 FRIDAY(5, "Fri"),  
 SATURDAY(6, "Sat"),  
 SUNDAY(7, "Sun");  
  
 *private int* dayOfTheWeek;  
 *private* String abbreviation;

*// Enum can have only private and default constructors  
 private* Days(*int* x, String abbreviation) {  
 System.out.println("In private parameterized constructor " + x);  
 *this*.dayOfTheWeek = x;  
 *this*.abbreviation = abbreviation;  
 }  
}

1. Explain the types of Association with example in Java.

Uploaded on GitHub.

1. Write a program to explain the behavior of each access modifiers in inheritance.

Uploaded on GitHub (week1/part1/questions).