Answer-1

1. **Which SOLID principle does the Customer class violate?**

   The Customer class violates **Single Responsibility Principle**.

2. **Why does the code violate the principle?**

   According to Single Responsibility Principle, every class should be responsible for only a single functionality. However, in this example, **Customer** class is **responsible for two functionalities:** storing the customer data and sending an email to the customer. Hence, if we shift **email sending functionality to other class**, then the Customer class will have only one responsibility and it will follow Single Responsibility Principle.

Answer-2

1. **Which SOLID principle does the USDollarAccount class violate?**

   The USDollarAccount violates **Liskov Substitution Principle**.

2. **Why does the code violate the principle?**

   According to the Liskov Substitution Principle "Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program." However, in the given example, class **USDollarAccount** extends the class **Account**, and it changes the correctness of the program by multiplying the amount by EXHANGE_RATE in credit() and debit() methods. Hence, we can make the credit() and debit() methods of **AccountOperations** class as abstract. And we can create two separate class **Account** and **USDollarAccount** which extends abstract class **AccountOperations**. In future, if there is **any new type account**, for example **INRAccount**, then it has to extend the **AccountOperations** class and it can implement the credit() and debit() method in its own way. In this way, the correctness of the program is not altered.

Answer-3

1. **Which SOLID principle does the Student class violate?**

   The Student class violates **Single Responsibility Principle**.

2. **Why does the code violate the principle?**

   According to Single Responsibility Principle, "Every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class". However, in the given example, the **Student** class has **multiple functionalities**. First, to **get and set the information for the Student**. Second, **save the students information to a file**. Third, **loading the student information** from the file. Hence, if we separate-out these functionalities, then the Student class will have only single functionality.

Answer-4

1. **Which SOLID principle does the Employer class violate?**

   The Employer class violates **Dependency Inversion Principle**.

2. **Why does the code violate the principle?**

   According to the Dependency Inversion Principle "One should depend on abstractions, not concretions". However, in the given example, the Employer class depends on **HourlyWorker** and **SalaryWorker** classes, which are concrete classes. Hence, if we make the Employer class dependent on the Interfaces and not on the concrete classes, then it will follow the Dependency Inversion Principle. Now, if any new type of worker comes, for example **PermanentWorker**, then it just has to implement the **IWorker** interface, and the Employer class can use **outputWageCostsForAllStaff method** without modifying any of its functionalities.

Answer-5

1. **Which SOLID principle does the following code violate?**

   The code violates **Interface Segregation Principle.**

2. **Why does the code violate the principle?**

   According to Interface Segregation Principle "No client should be forced to depend on methods it does not use." In this example, aquatic insects can't fly, but still it has to implement the **fly()** method as it has implemented the general purpose **IInsect** Interface. Similarly, flying insects can't swim, but still it has to implement **swim()** method as it has implemented the general purpose **IInsect** Interface. Hence, we can divide the **IInsect** interface into three separate interfaces as following: **IInsect, IAquaticInsect, and IFlyingInsect**. In this way, each Interface has now only one purpose.

Answer-6

1. **Which SOLID principle does the following code violate?**

   The code violates **Open/Closed Principle**.

2. **Why does the code violate the principle?**

   According to the Open/Closed Principle "Software entities should be open for extension but closed for modification." In the given example, the **CountryGDPReport** class has a **constructor** (which initializes the objects for both countries) and a method **printCountryGDPReport** (which prints the GDP report for each country). However, if there is a **new country, for example China**, then both the methods need to be updated as a part of this extension. So, **this class and methods will be modified if there is new country,** which violates the Open/Closed Principle. Hence, we can create an interface named **ICountry** and then **CountryGDPReport** should accept only the instance of **ICountry**. In this way, the class is now open for extension, but closed for modification.

Answer-7

1. **Which SOLID principle does the following code violate?**

   The code violates **Interface Segregation Principle.**

2. **Why does the code violate the principle?**

   According to the principle, "Many client-specific interfaces are better than one general-purpose interface." However, in the given example, there is only one interface, which is responsible for multiple functionalities. For example, the Book class doesn't have any Play Time and Cast List, but as it implements the **general interface ILibraryItem,** it has to implement all the methods even if it has nothing to do with it. Similarly, it applies to DVD class as well. So, the **ILibraryItem can be divided into three separate interfaces as following: IBook, IDVD, and ILibraryItem.** Now each interface has it own functionality.