Tutorial 1 - Solution

Introduction to Object-Oriented Modeling

Table of contents

1 Multiple Choice Questions

1. **What is Encapsulation in OOP?**
   - **Answer: B) Combining data and functions that operate on them.** Encapsulation is the OOP principle that combines the data (attributes) and the methods (functions) that operate on the data into a single unit called class. It also restricts direct access to some of the object's components, which can protect the object's internal state from unauthorized access and modification.
2. **Which of the following is an example of Polymorphism in OOP?**
   - **Answer: C) Multiple classes having methods with the same name but different implementations.** Polymorphism allows methods with the same name to have different implementations in different classes. This can be achieved through method overloading within a class or method overriding between parent and child classes in inheritance.
3. **Which OOP principle hides the internal state of an object and only exposes operations?**
   - **Answer: B) Abstraction.** Abstraction focuses on hiding the internal implementation details of a system and exposing only the necessary parts of it to the outside world. It allows focusing on what an object does instead of how it does it.
4. **Choose the correct statement about 'Loose Coupling' in software design.**

- **Answer: A) It makes it easier to modify and maintain software.** Loose coupling refers to a design goal that seeks to reduce the interdependencies between components of a system to make it easier to modify, extend, and maintain.

2 Descriptive Questions

5. **Describe the difference between an 'Abstract Class' and an 'Interface'.**

- An abstract class is a class that cannot be instantiated and is designed to be inherited by other classes. It can contain both abstract methods (which do not have an implementation and must be implemented by subclasses) and concrete methods (which have an implementation). Interfaces, on the other hand, are contracts that define a set of methods that implementing classes must provide. Interfaces cannot contain any concrete methods (methods with an implementation) and do not maintain any state. Abstract classes are used when some common implementation among related classes should be shared, whereas interfaces are ideal for defining a common protocol for unrelated classes.

6. **Why does software such as the Linux kernel or Mozilla Firefox tend to get more complex over time?**

- Software complexity increases over time due to the accumulation of features, bug fixes, and enhancements. As more code is added to accommodate new functionalities, the interdependencies between various parts of the software increase, making it harder to maintain and understand. This phenomenon, often referred to as "software bloat," is driven by the need to meet evolving user requirements, security updates, and compatibility with new hardware or software environments.

7. **How do techniques such as information hiding, polymorphism, and interfaces promote loose coupling?**

- Information hiding, polymorphism, and interfaces promote loose coupling by minimizing the interdependencies between components of a system. Information hiding achieves this by encapsulating details within modules, reducing the reliance of other parts of the system on those internal workings. Polymorphism allows for the interchangeability of objects with different implementations, enabling a more flexible and modular design. Interfaces define contracts for interaction, allowing different components to communicate through well-defined protocols without needing to know the specifics of each other's implementations. Together, these techniques facilitate easier modification, extension, and maintenance of software.

8. **Explain what the waterfall model of software development lifecycle is with a detailed overview of each phase.**

- The waterfall model is a sequential design process used in software development, where progress flows steadily downwards through several phases like a waterfall. The phases include:
  - **Requirements Analysis:** This phase involves gathering all the specific requirements for the software to be developed. It is crucial for setting the foundation for the entire project.
  - **System Design:** Based on the requirements analysis, the system's architecture and design are planned, outlining the software's structure, components, modules, and interface.
  - **Implementation:** In this phase, the actual coding of the software takes place based on the design documents created in the previous phase.
  - **Integration and Testing:** After implementation, all the pieces are brought together, and the software is tested as a whole to find and fix bugs and ensure that it meets the initial requirements.
  - **Deployment:** Once testing is complete and the software is bug-free, it is deployed to the production environment where the end-users can begin to use it.
  - **Maintenance:** After deployment, the software will require updates, improvements, and bug fixes based on user feedback and evolving requirements.

This model is best suited for projects with well-understood requirements where changes are unlikely during the development process.