C++ week 4

Khaula Molapo

1.

Polymorphism is widely used in user interface (UI) systems to handle multiple components in a flexible way. For example, consider a UI system with buttons, text fields, and checkboxes. Each component inherits from a common base class UIElement and implements a draw() function. Polymorphism allows the system to treat all elements as UIElement objects while calling the correct draw() function for each specific type at runtime. This means the UI can store different elements in a single collection and render them without checking their exact types. It simplifies code maintenance and extension; adding a new UI component requires only implementing its draw() function. Polymorphism also supports event handling, allowing different components to respond to user actions appropriately. This approach increases flexibility, reduces code duplication, and ensures a scalable and maintainable UI system capable of handling diverse elements efficiently.

2.

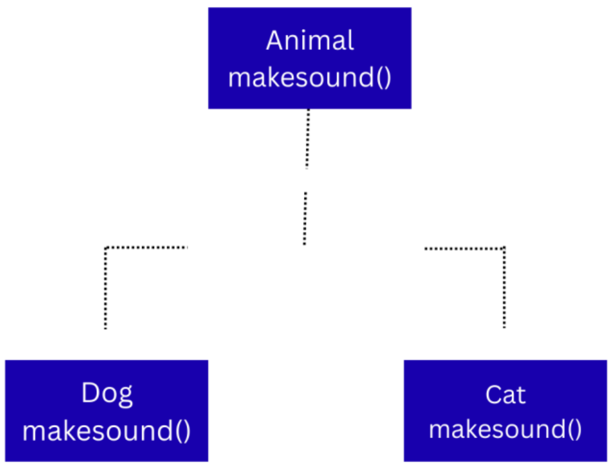
A pure virtual function is a function in a base class declared with = 0 that must be overridden in derived classes. It defines an interface without providing an implementation, allowing polymorphism in C++. Pure virtual functions enable the creation of abstract classes that cannot be instantiated directly but can define a common interface for all derived classes. For example, a base class Shape may declare a pure virtual function draw(). Each specific shape like Circle or Rectangle implements draw(). This ensures consistent behavior while allowing different implementations. Pure virtual functions are key to designing flexible, extensible, and polymorphic class hierarchies in object-oriented programming.

3.

This program demonstrates polymorphism using a UI system with buttons and text fields. Writing it in Visual Studio Code helped me understand how base class pointers can call derived class methods at runtime. Using a pure virtual function ensured that all UI components implement draw(). The program shows how polymorphism allows flexible and scalable design, making it easier to manage multiple objects uniformly. Testing the program clarified the runtime behavior of virtual functions and memory allocation for derived objects. This hands-on experience highlighted the importance of abstract classes and polymorphism in creating maintainable, extensible object-oriented systems.

4.

The UML diagram represents a polymorphic class hierarchy for a UI system. The base class UIElement contains a pure virtual function draw(). Derived classes Button, TextField, and Checkbox override draw() with their own implementations. Arrows indicate inheritance relationships. The diagram visually shows how a single interface (UIElement) can support multiple specific behaviors. Polymorphism allows the system to handle all elements uniformly while executing the correct behavior at runtime. This visual representation helps understand abstract classes, method overriding, and runtime polymorphism. UML diagrams make it easier to design, communicate, and maintain object-oriented systems efficiently.


Animal
makesound()
Dog
makesound()
Cat
makesound()