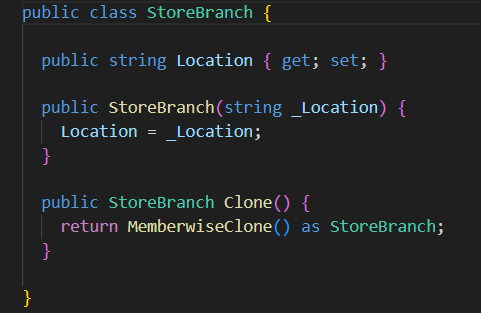
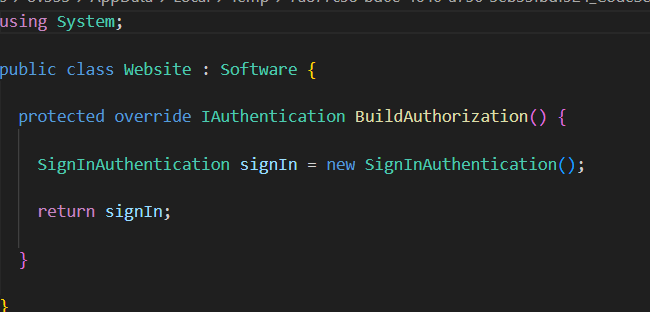
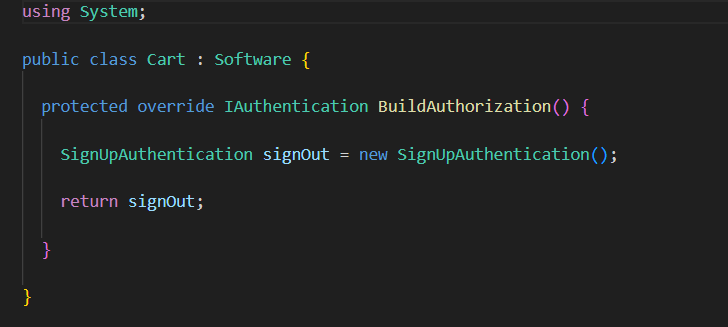
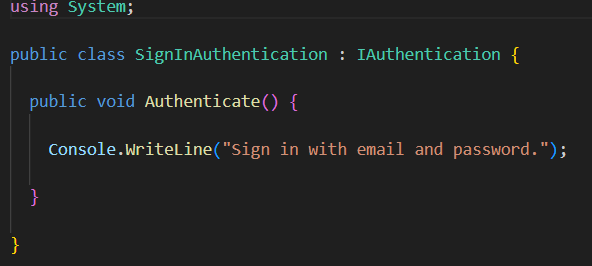
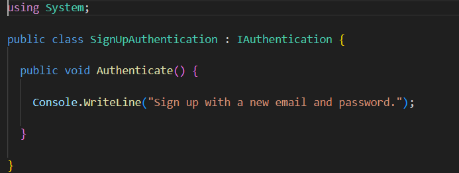
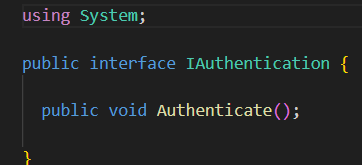
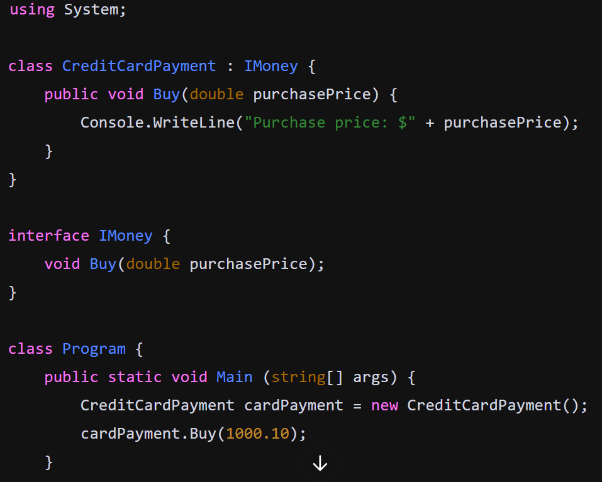
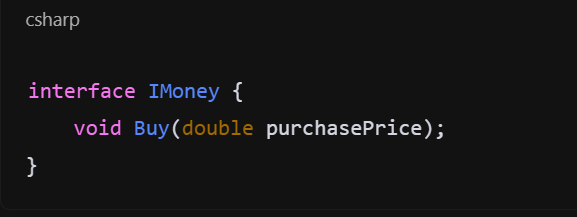
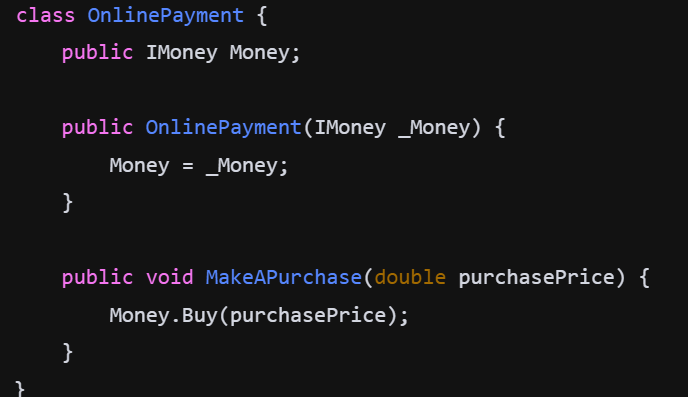
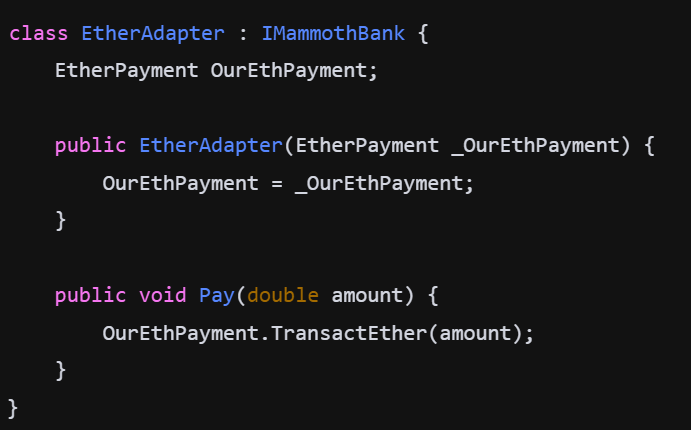
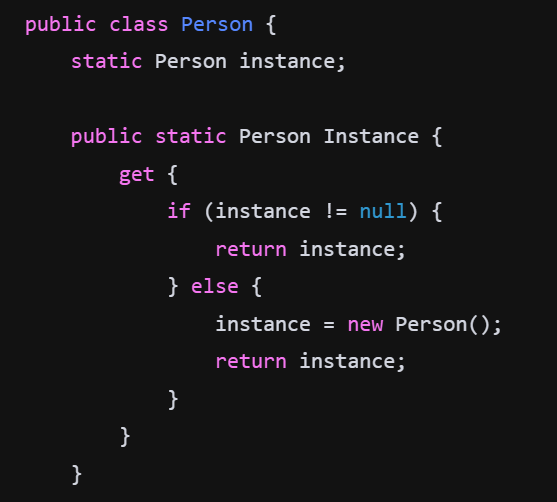
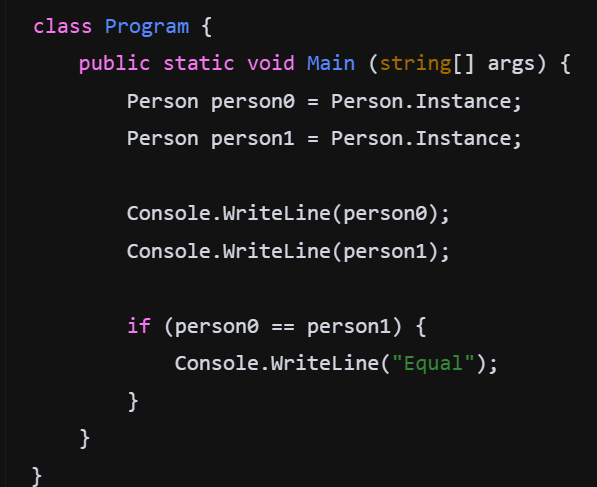
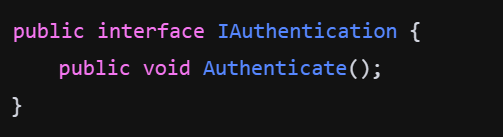
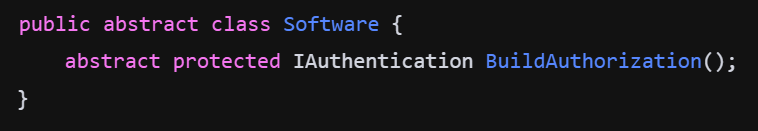
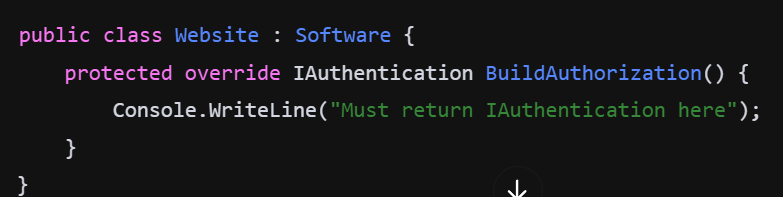
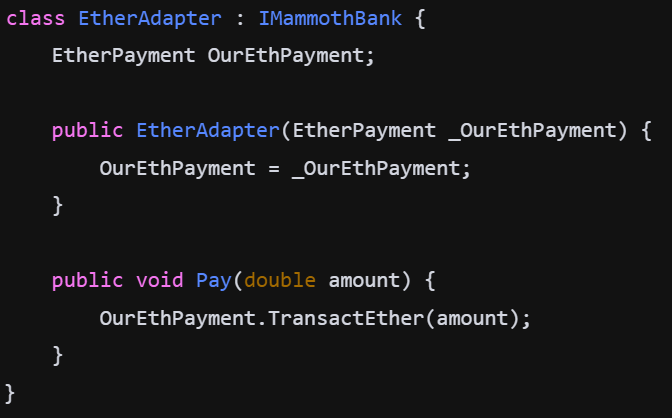
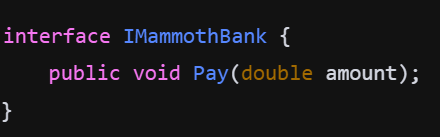
1. Code: aby
   1. Code segment:
      1. 
   2. Patterns used:
      1. Prototype Pattern
   3. Usage justification
      1. **Prototype pattern**: is utilized to enable the creation of new *StoreBranch* instances by cloning existing ones. This method allows for efficient object creation when similar objects are needed, reducing the need for repetitive initialization code and simplifying the management of object properties.
2. Code: cwm
   1. Code segment:
      1. Factory method patterns:
         1. 
         2. A screen shot of a computer program

            Description automatically generated
         3. 
      2. Strategy patterns
         1. 
         2. 
      3. Adapter patterns
         1. 
   2. Patterns used:
      1. Factory methods, strategy, adapter pattern
   3. Usage justification
      1. **Factory Method**: The pattern allows subclasses to define their own authentication creation logic, promoting extensibility and separation of concerns within the Software class hierarchy.
      2. **Strategy:** This pattern enables dynamic selection of authentication algorithms at runtime, providing flexibility in how authentication is handled, as seen in the SignInAuthentication and SignUpAuthentication classes.
      3. **Adapter:** The IAuthentication interface facilitates integration of various authentication methods, allowing them to work interchangeably within the Software class without requiring modifications to the existing code.
3. Code: kef
   1. Code segment:
      1. 
   2. Patterns used:
      1. Adapter pattern
   3. Usage justification
      1. **Adapter Pattern:** The IMoney interface allows different payment methods, like CreditCardPayment, to implement their own version of the Buy method. This makes it easy to add new payment types later without changing the existing code. It promotes flexibility by letting various payment options work together seamlessly.
4. Code: kir
   1. Code segment:
      1. Adapter
         1. 
      2. Composite
         1. 
   2. Patterns used:
      1. Adapter, composite
   3. Usage justification
      1. **Adapter Pattern:** The IMoney interface allows different payment methods, like CreditCardPayment, to implement their own version of the Buy method. This enables seamless integration of various payment options without altering existing code.
      2. **Composite Pattern:** The OnlinePayment class allows different payment methods to be treated uniformly, acting as a unified interface for making purchases. It facilitates handling multiple payment implementations while promoting loose coupling.
5. Code: pyx
   1. Code segment:
      1. Adapter pattern
         1. A black screen with colorful text

            Description automatically generated
         2. 
   2. Patterns used: adapter pattern.
   3. Usage justification
      1. **Adapter Pattern**: The EtherAdapter acts like a bridge, allowing the EtherPayment class to work with the IMammothBank interface. This makes it easy to use different payment methods in a unified way, letting you switch or add payment options without changing much code. As well as storing the EtherAdapter data onto the EtherPayment for later usage.
6. Code: ret
   1. Code segment:
      1. 
      2. 
   2. Patterns used:
      1. Singleton
   3. Usage justification
      1. **Singleton Pattern**: The Person class ensures that only one instance of itself can be created. The Instance property checks if an instance already exists; if not, it creates one. This way, both person0 and person1 refer to the same Person instance, demonstrating that the Singleton pattern is effectively implemented.
7. Code: rom
   1. Code segment:
      1. No pattern code segments.
   2. Patterns used:
      1. No patterns .
   3. Usage justification
      1. No usage patterns.
8. Code: tow
   1. Code segment:
      1. Strategy
         1. 
      2. Template
         1. 
         2. 
   2. Patterns used:
      1. Strategy and template
   3. Usage justification
      1. **Strategy Pattern:** The IAuthentication interface allows for various authentication strategies to be implemented. This design enables different classes to define their own authentication methods, giving flexibility and the ability to switch strategies easily.
      2. **Template Method Pattern:** The Software class defines an abstract method BuildAuthorization, which must be implemented by subclasses like Website. This establishes a template for building authorization while allowing subclasses to provide specific implementations.
9. Code: vug
   1. Code segment:
      1. Adapter
         1. 
      2. Strategy
         1. 
   2. Patterns used:
      1. Adapter Pattern and strategy
   3. Usage justification
      1. **Adapter Pattern**: The EtherAdapter allows the EtherPayment class to implement the IMammothBank interface. This enables the EtherPayment to be used in a context where the IMammothBank interface is required, promoting flexibility in payment options.
      2. **Strategy Pattern**: The IMammothBank interface defines a payment method, allowing various implementations (like EtherAdapter) to be used interchangeably. This pattern provides flexibility to choose different payment strategies at runtime.