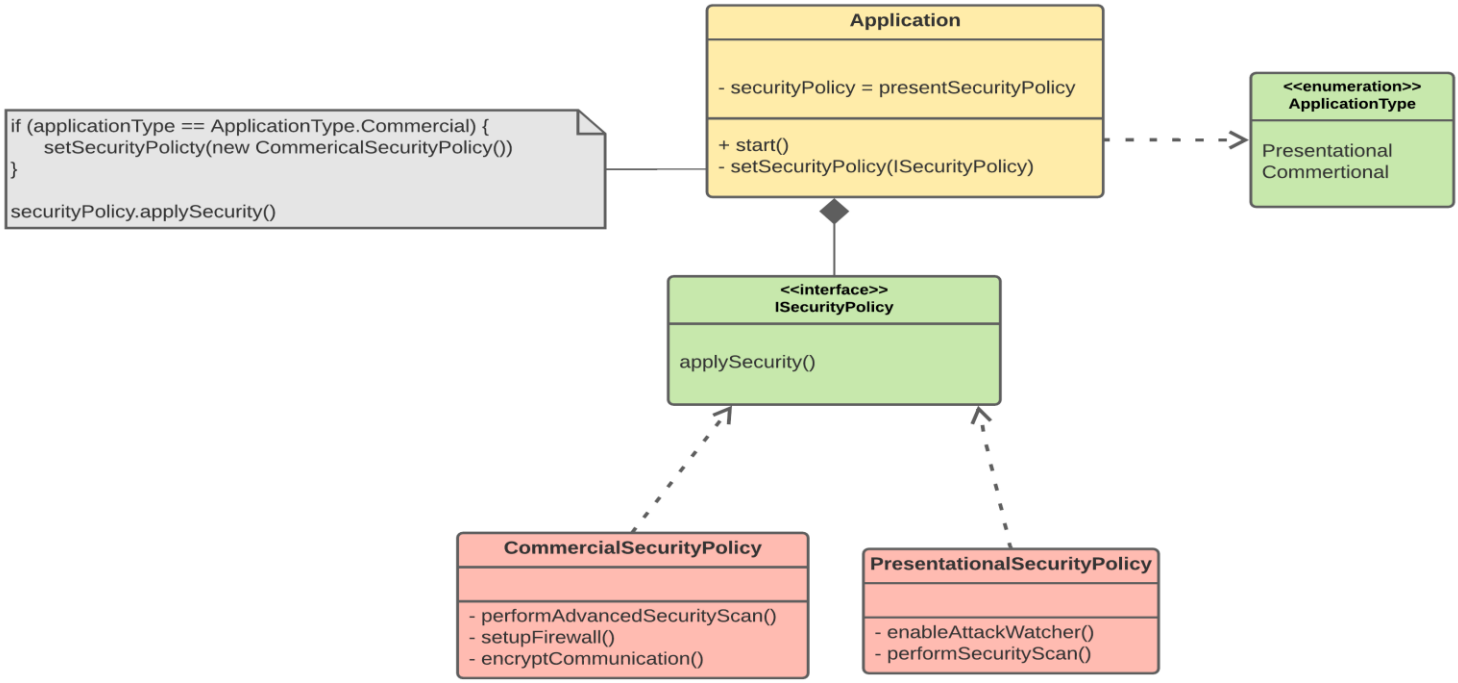
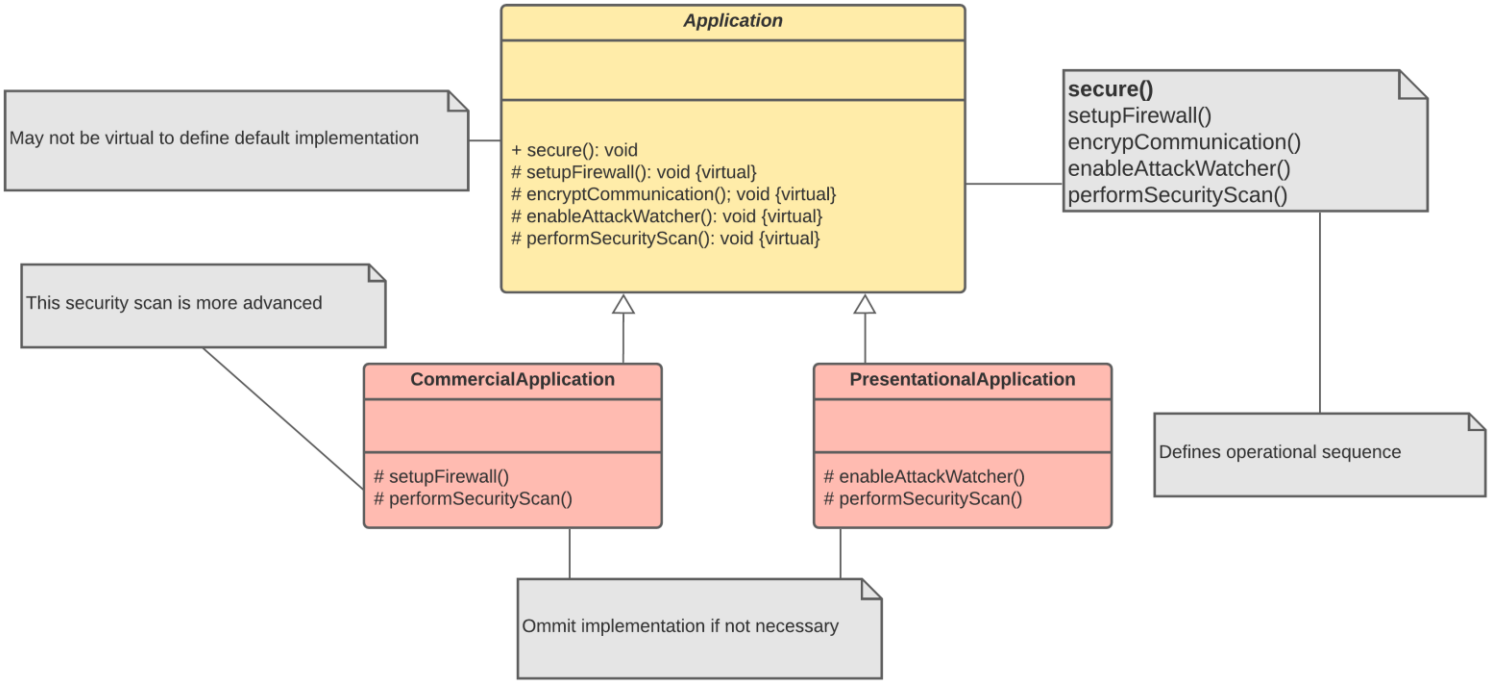
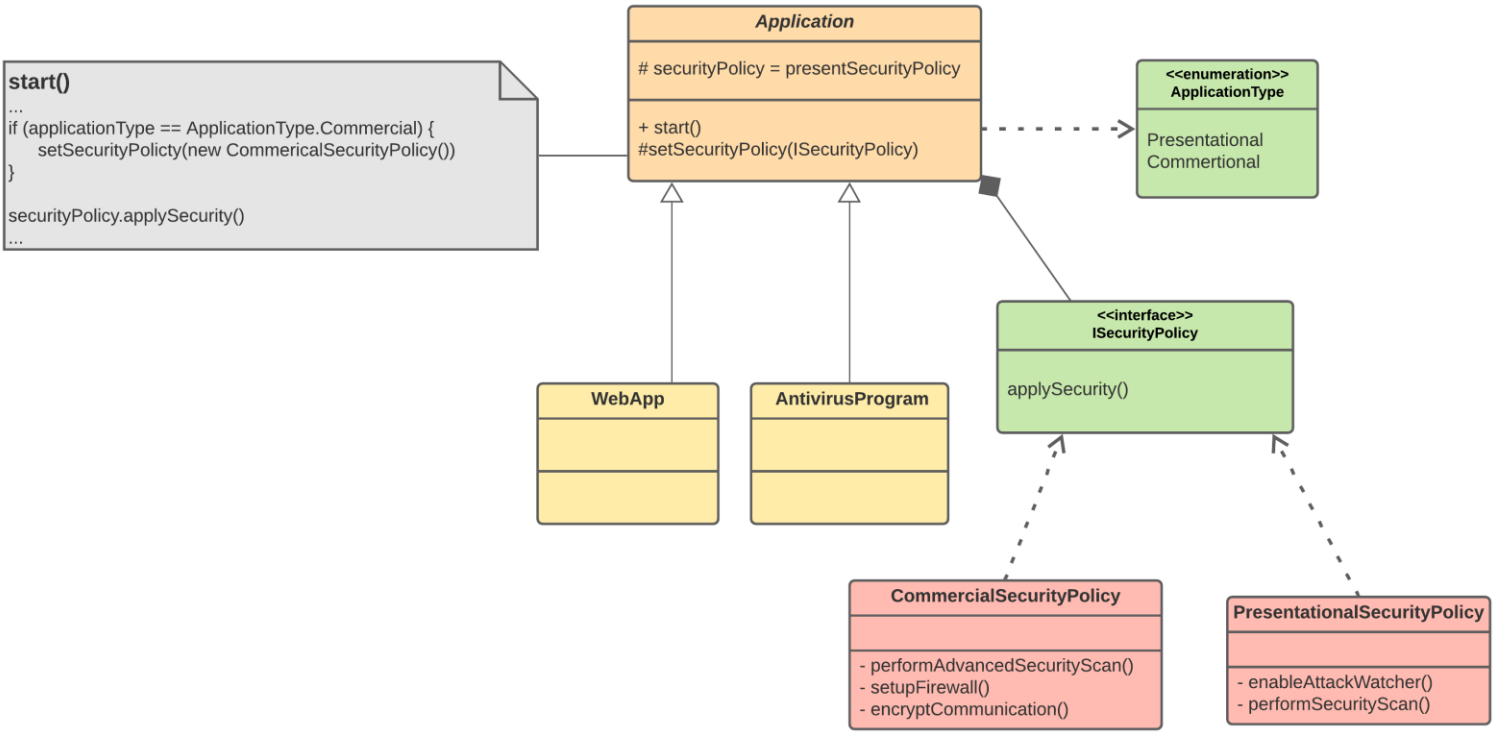
Strategy Base Diagram



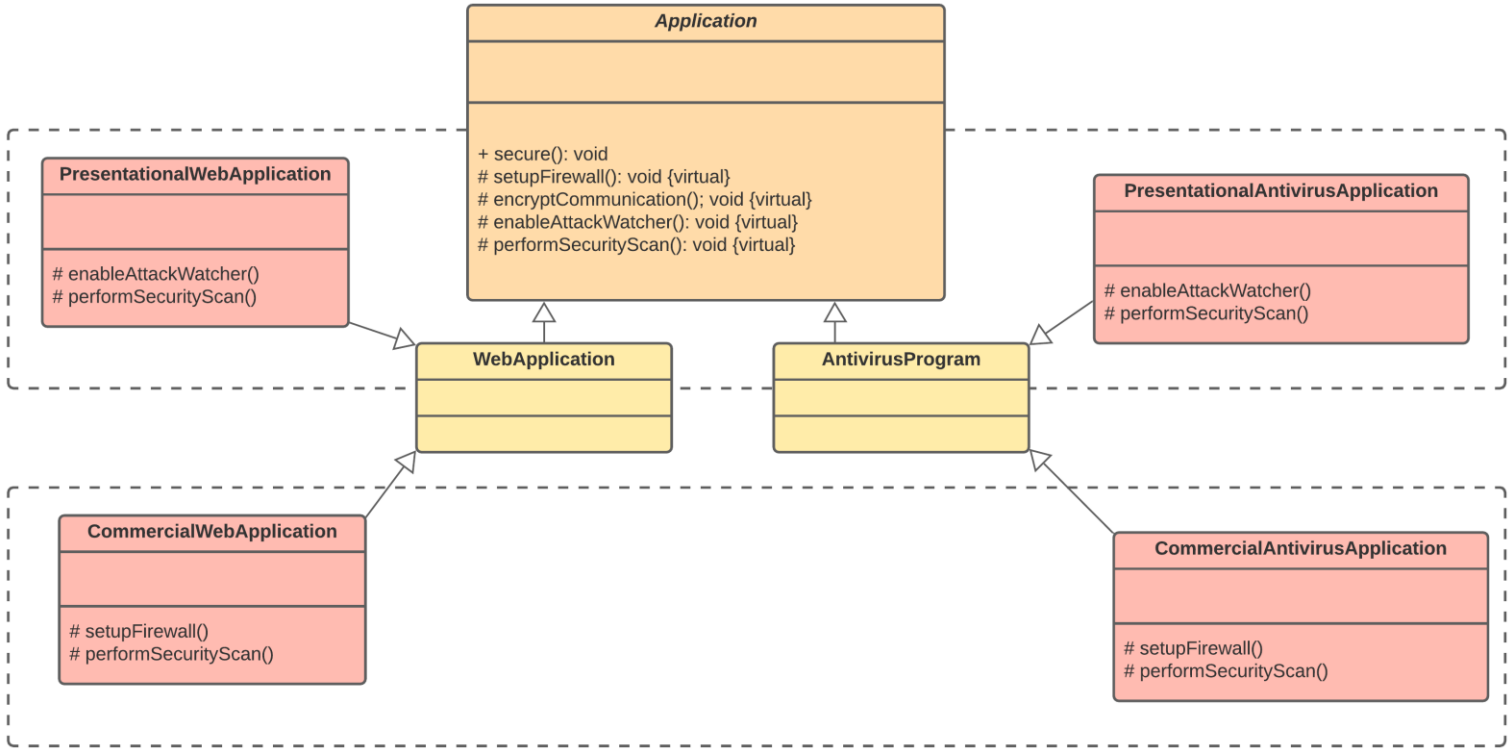
Template Base Diagram



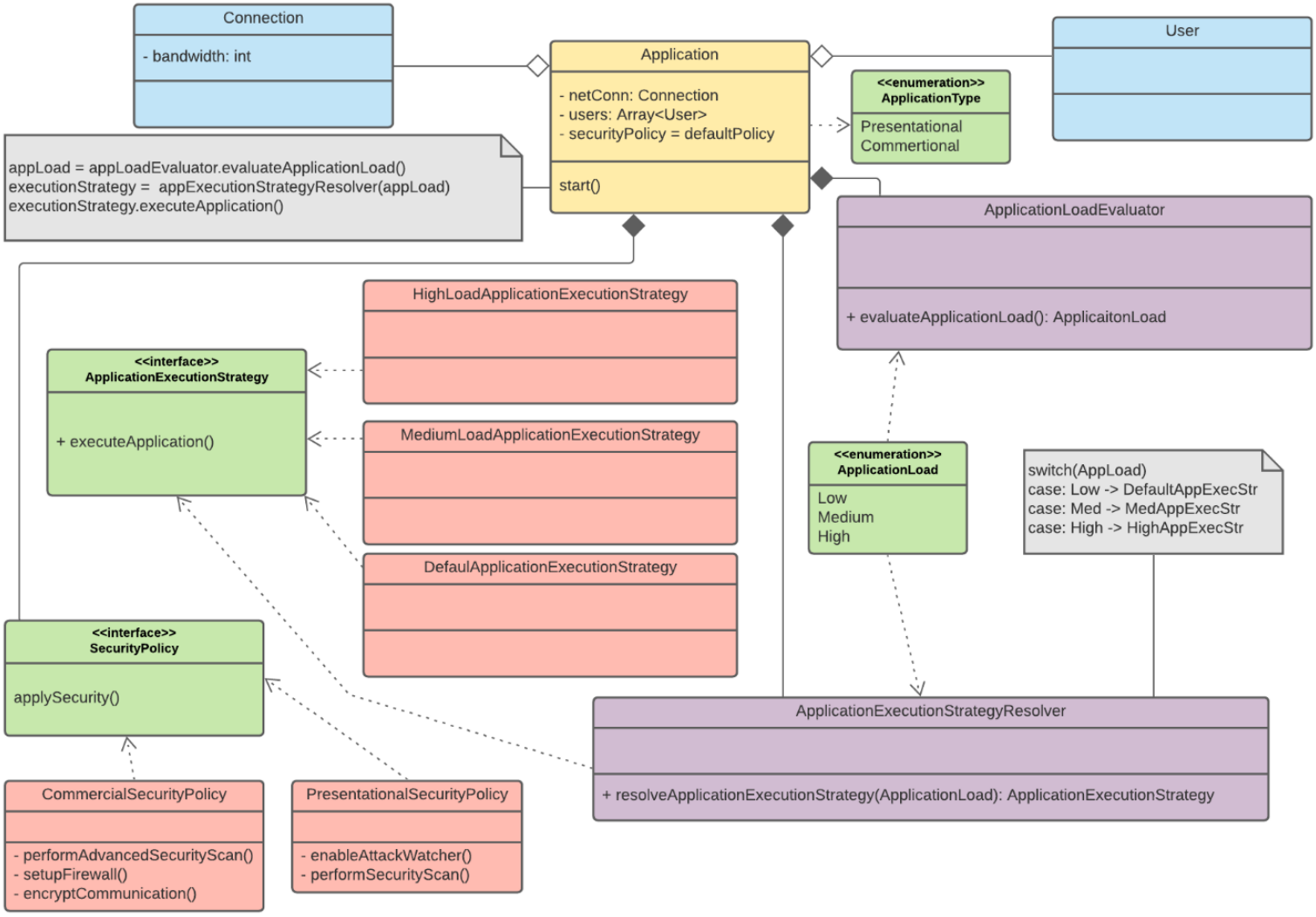
Strategy Requirement A



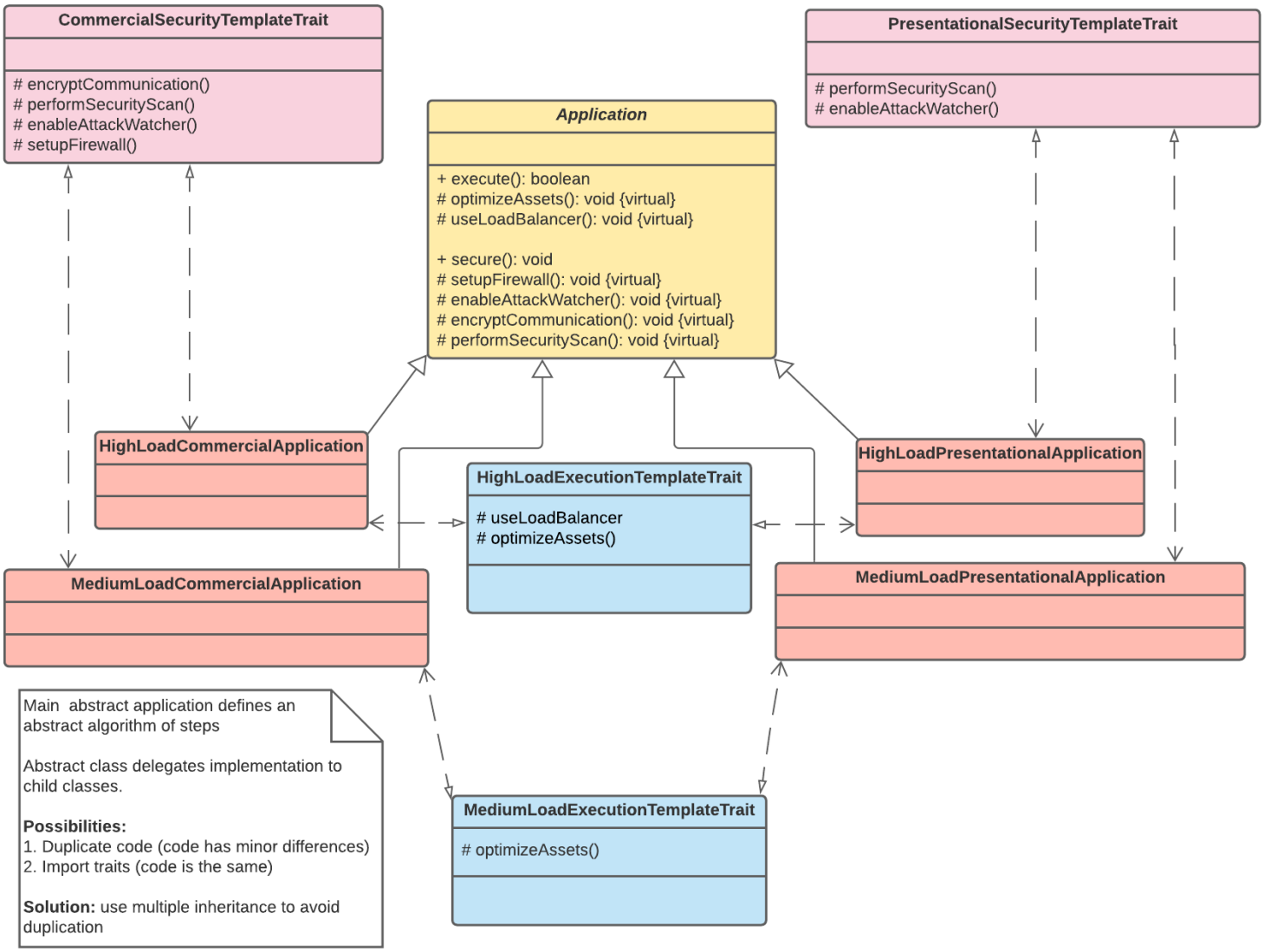
Template Requirement A



Strategy Requirement B



Template Requirement B



Conclusions

In the OOP community it is commonly agreed to prefer composition over inheritance, therefore Strategy pattern in this regard is favorable.

Pattern essence:

Strategy pattern defines various implementations of the same algorithm.

Method Template pattern defines an abstract sequence of steps in the base class and leaves the implementation to children classes. Each child class has freedom to define how desired outcome is achieved.

Base situation:

Strategy – composition

Template – inheritance

Strategy: use different algorithms depending on context

Template: use same algorithm with steps implemented differently

Requirement A: domain expansion – new entity shares functionality

Strategy: use exact same defined algorithm (defined in strategy)

Template: always define new algorithm, but maintain execution sequence

If algorithm in the new entity is the same as in previous, will result in code duplication

Requirement B: domain expansion – new functionality to the same entity

Strategy: flexible (decoupled, no conflicts)

Template: not flexible

Requires to define a precise case of a class, each feature results in exponential class growth

If programming language does not support multiple inheritance, requires code duplication