Behavioral Pattern: Template Method

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

Describing the template method pattern

- Mail parser

Structure of the template method pattern

Coming Up

Use cases for this pattern

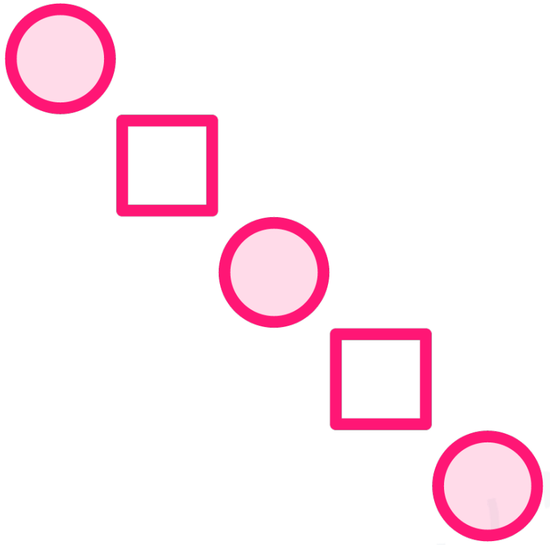
Pattern consequences

Related patterns

Describing the Template Method Pattern

Creational

Structural



Behavioral

Template method

The intent of this pattern is to define the skeleton of an algorithm in an operation, deferring some steps to subclasses. It lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Describing the Template Method Pattern

Email parser scenario

- Read & analyse email across different types of email servers

```
public class EudoraMailParser { }

public class ApacheMailParser { }

public class ExchangeMailParser { }
```

Describing the Template Method Pattern

```
public class EudoraMailParser { }

public class ApacheMailParser { }

public class ExchangeMailParser { }
```

Describing the Template Method Pattern

```
public class EudoraMailParser { }

public class ApacheMailParser { }

public class ExchangeMailParser { }
```

Describing the Template Method Pattern

```
public class EudoraMailParser { }

public class ApacheMailParser { }

public class ExchangeMailParser { }
```

Describing the Template Method Pattern

```
public class ExchangeMailServer
{
      public void FindServer()
      {     // ...
      }

      public void AuthenticateToServer()
      {  // ...
      }

      public string ParseHtmlMailBody(string identifier)
      {  // ...
         return "The mail body";
      }
}
```

Describing the Template Method Pattern

The steps that need to be executed for parsing mail are the same for all MailServer classes, but some implementations can differ

```
public class ExchangeMailServer
{
      public void FindServer()
      {     // ...
      }

      public void AuthenticateToServer()
      {  // ...
      }

      public string ParseHtmlMailBody(string identifier)
      {  // ...
         return "The mail body";
      }
}
```

Describing the Template Method Pattern

The steps that need to be executed for parsing mail are the same for all MailServer classes, but some implementations can differ

```
public class ExchangeMailServer
{
      public void FindServer()
      {     // ...
      }

      public void AuthenticateToServer()
      {  // ...
      }

      public string ParseHtmlMailBody(string identifier)
      {  // ...
         return "The mail body";
      }
}
```

Describing the Template Method Pattern

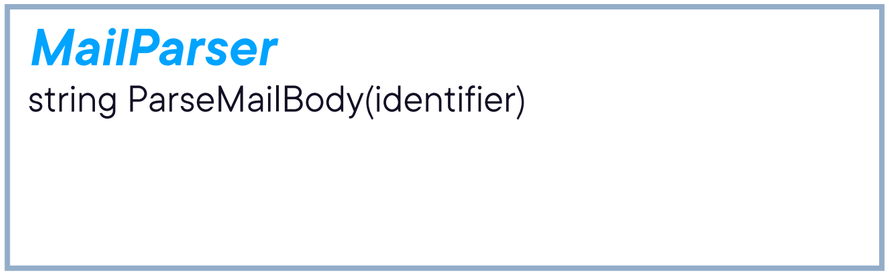
The steps that need to be executed for parsing mail are the same for all MailServer classes, but some implementations can differ

```
public class ExchangeMailServer
{
      public void FindServer()
      {     // ...
      }

      public void AuthenticateToServer()
      {  // ...
      }

      public string ParseHtmlMailBody(string identifier)
      {  // ...
         return "The mail body";
      }
}
```

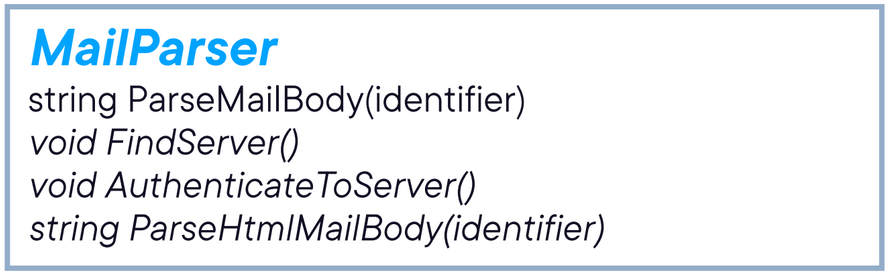
Describing the Template Method Pattern

The steps that need to be executed for parsing mail are the same for all MailServer classes, but some implementations can differ

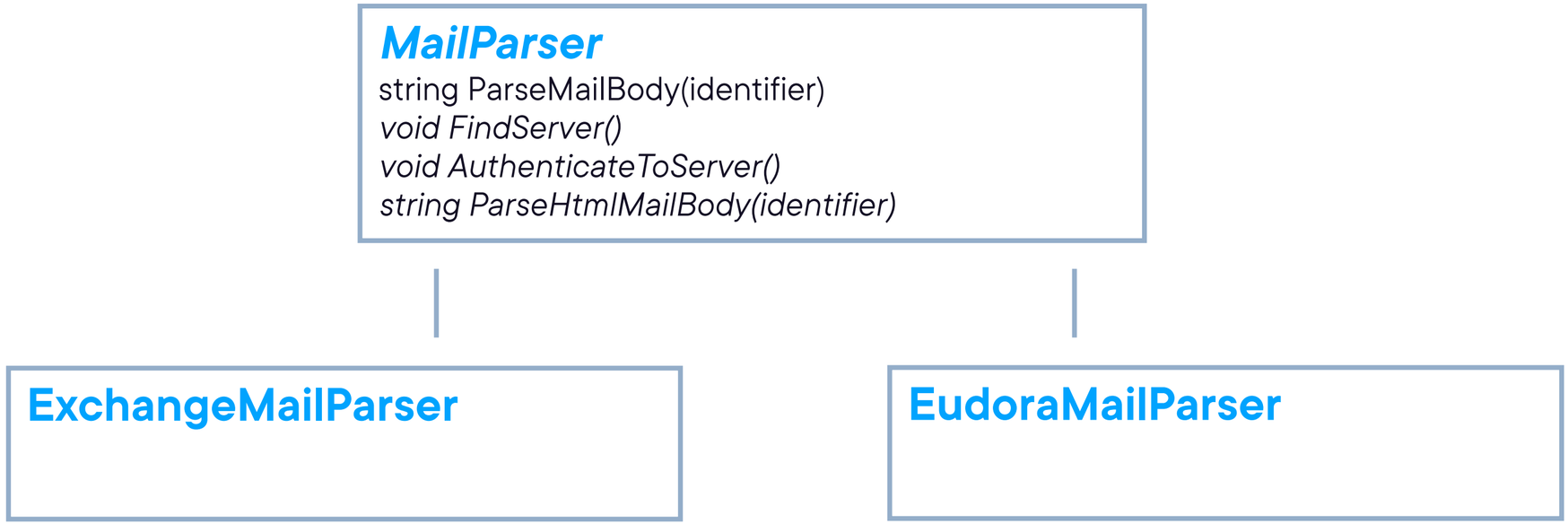
Describing the Template Method Pattern



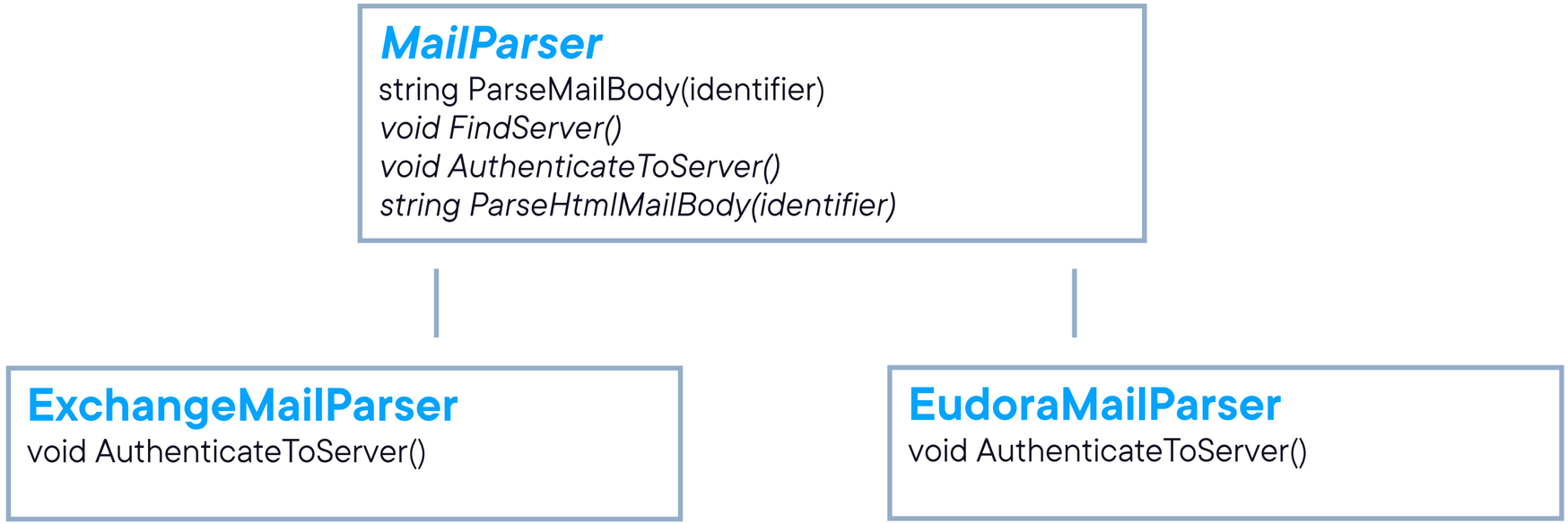
Describing the Template Method Pattern



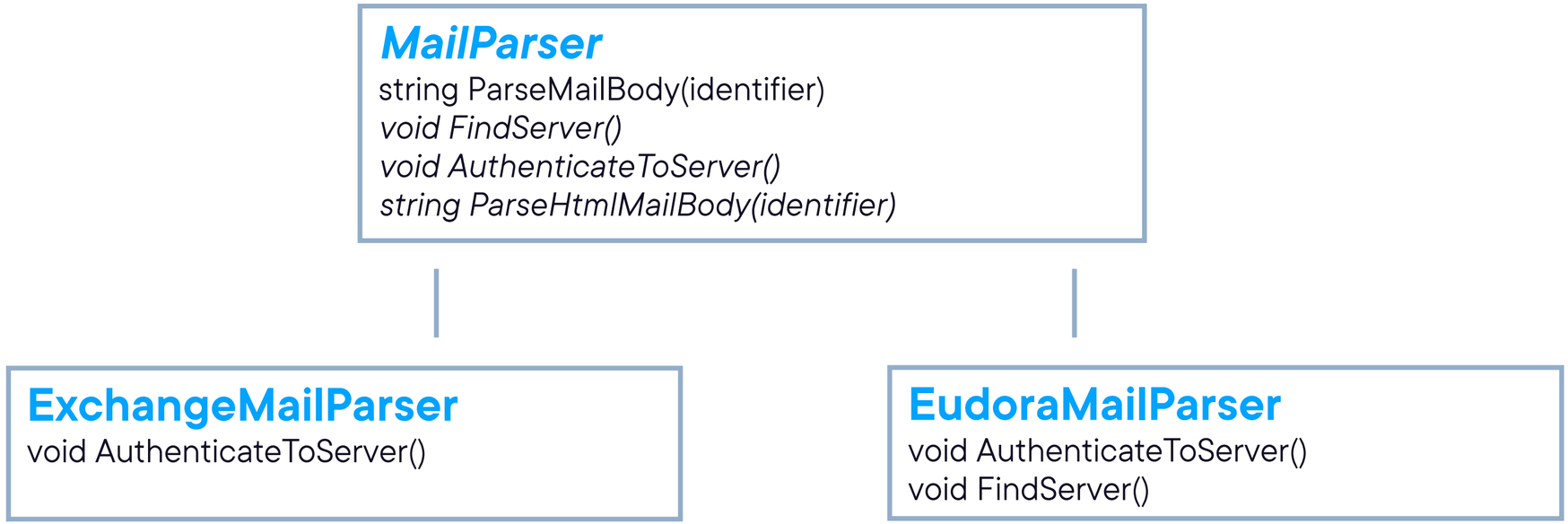
Describing the Template Method Pattern



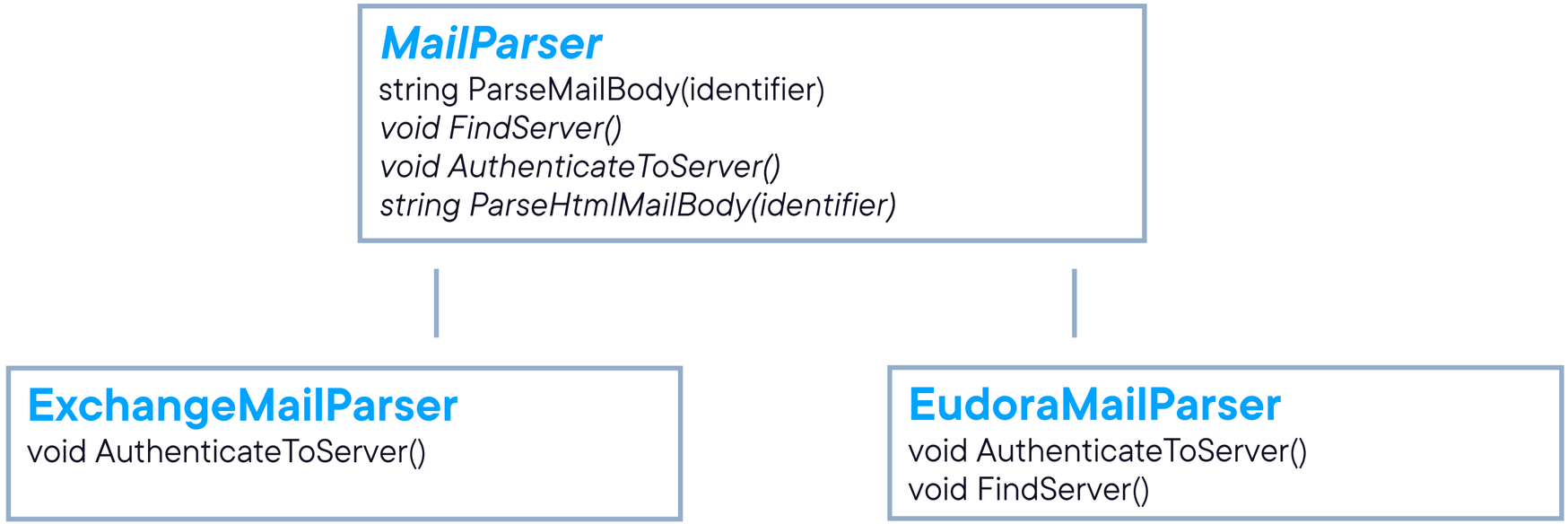
Describing the Template Method Pattern



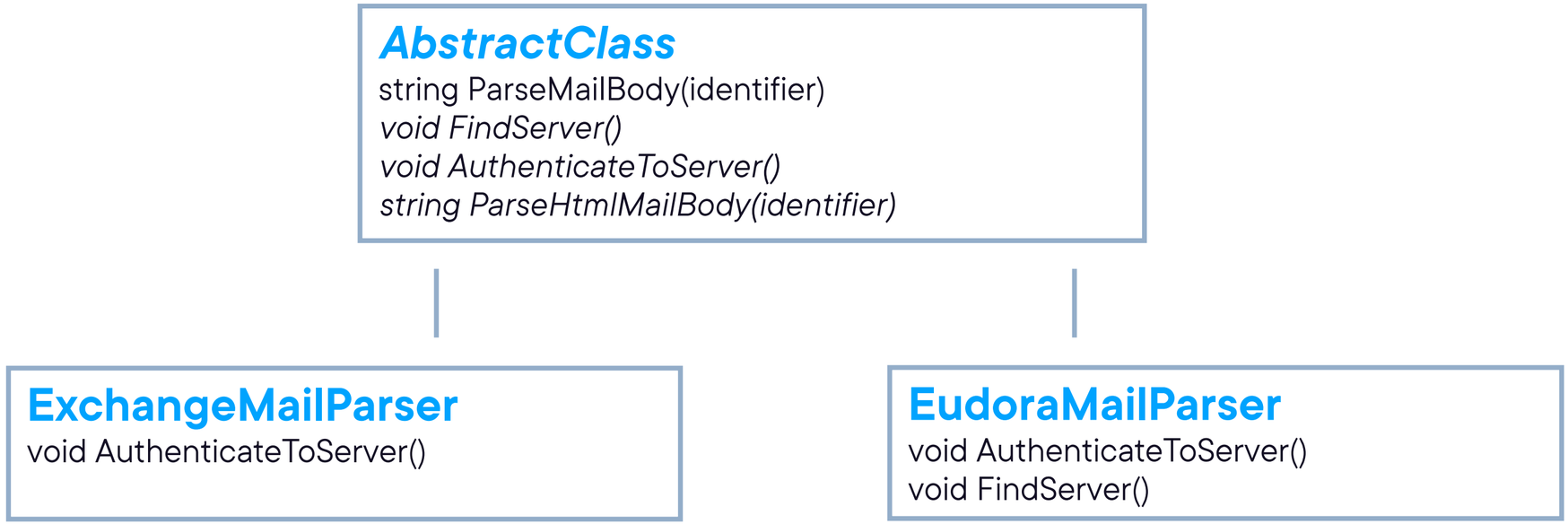
Describing the Template Method Pattern



Structure of the Template Method Pattern

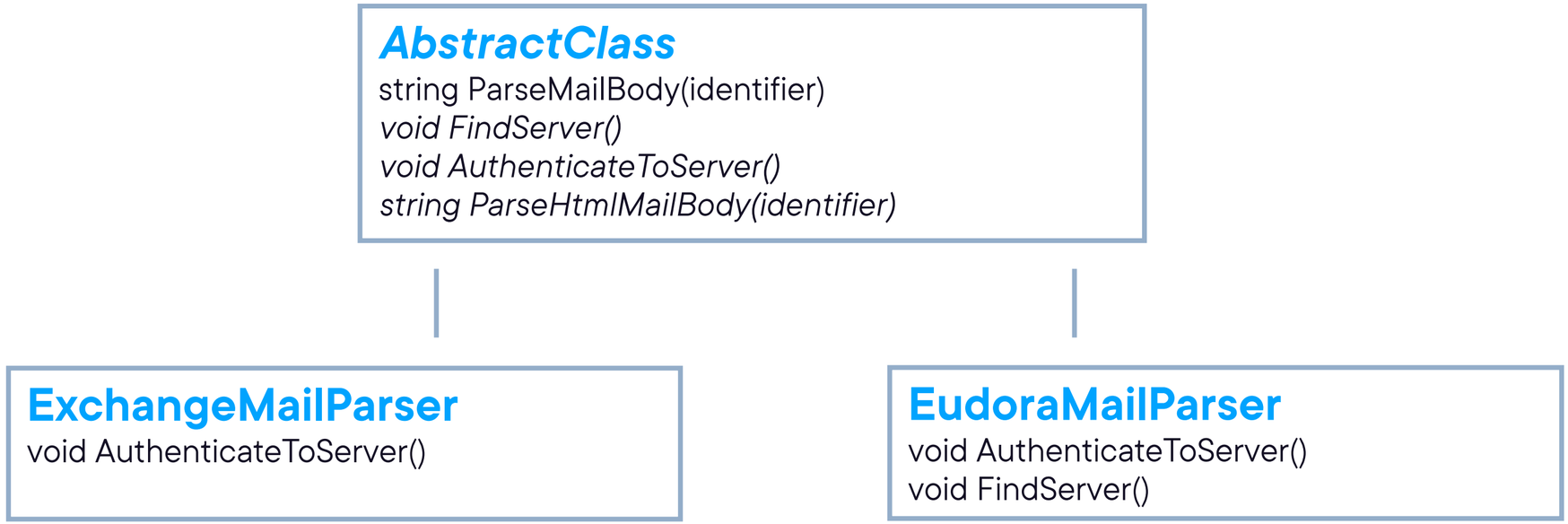


Structure of the Template Method Pattern

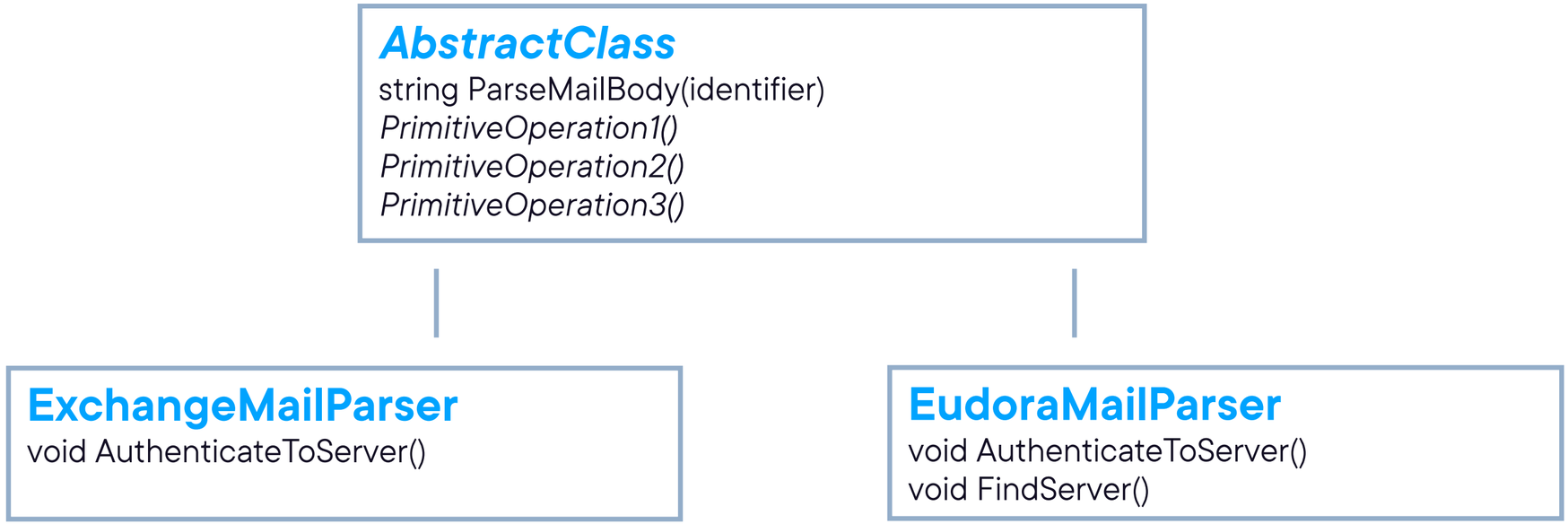


`AbstractClass` **defines abstract primitive operations that concrete subclasses define to implement steps of an algorithm**

Structure of the Template Method Pattern

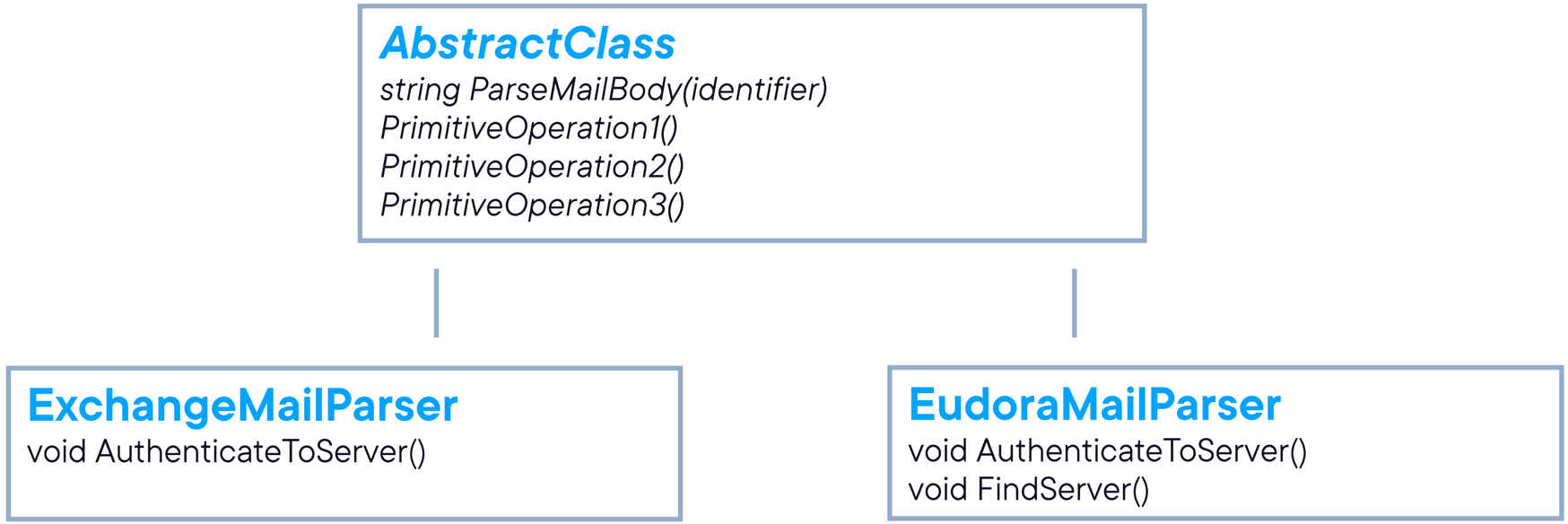


Structure of the Template Method Pattern

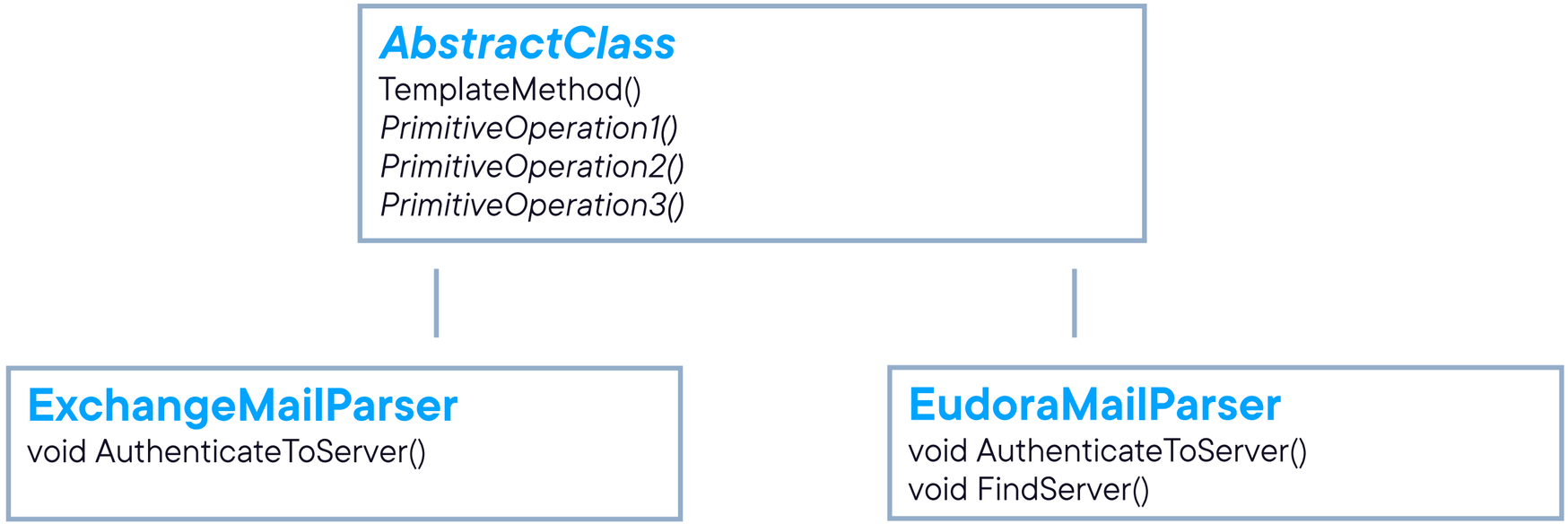


`AbstractClass` **implements a template method defining the skeleton of an algorithm**

Structure of the Template Method Pattern

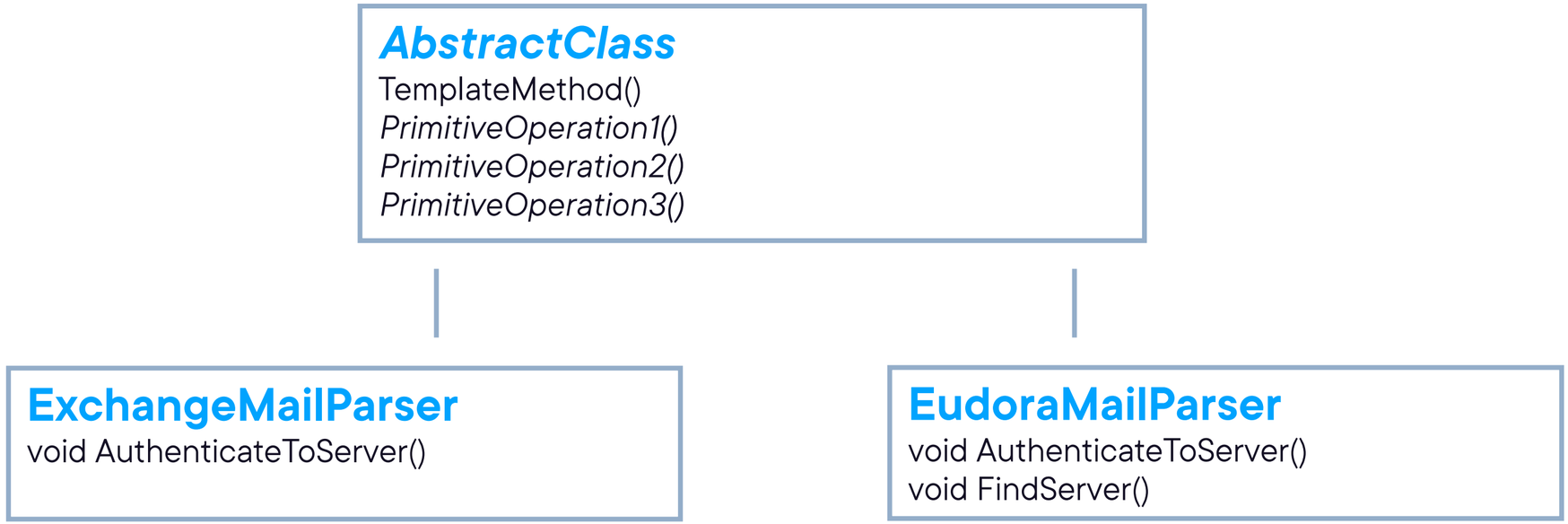


Structure of the Template Method Pattern

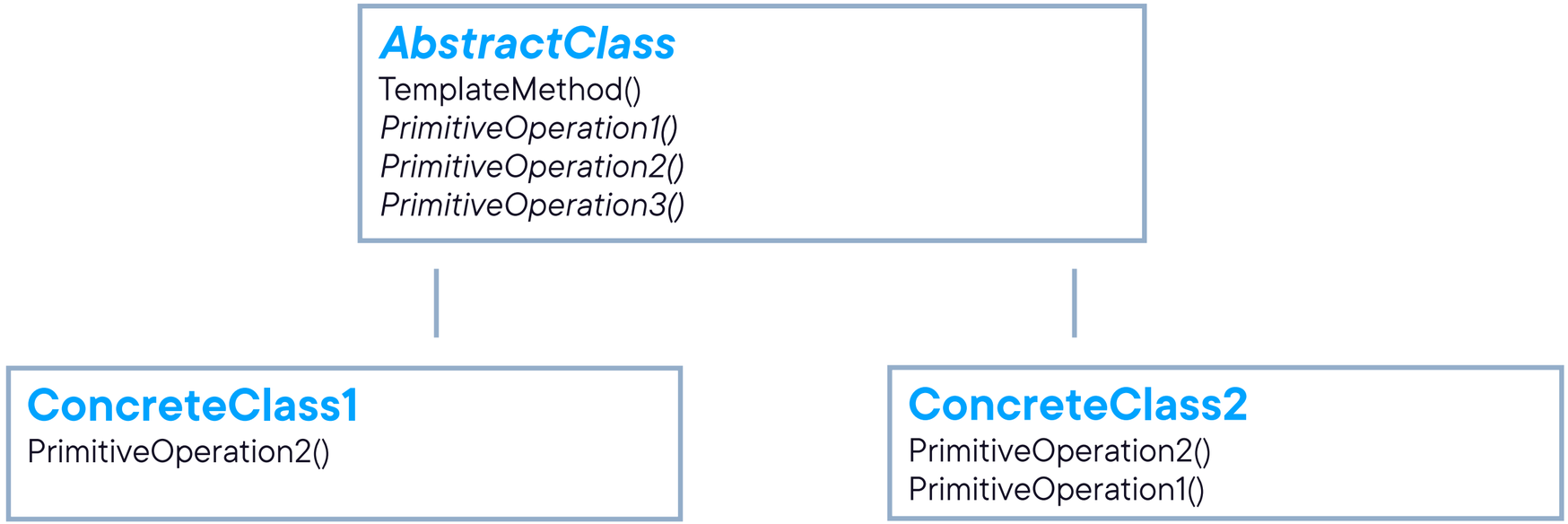


`ConcreteClass` **implements the primitive operations to carry out subclass-specific steps of the algorithm**

Structure of the Template Method Pattern



Structure of the Template Method Pattern



Demo

Implementing the template method pattern

Use Cases for the Template Method Pattern

When you want to implement invariant parts of an algorithm only once, and want to leave it to subclasses to implement the rest of the behavior

When you want to control which part of an algorithm subclasses can vary

When you have a set of algorithms that don’t vary much

Use Cases for the Template Method Pattern

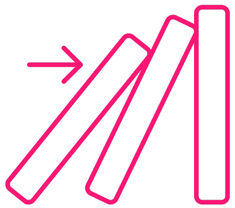
Document generation

Sorting algorithms

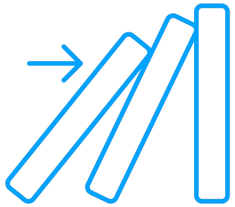
Online shopping cart

Automated testing frameworks

Pattern Consequences

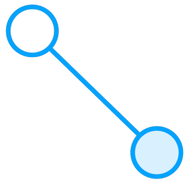


Template methods are a fundamental technique for code reuse

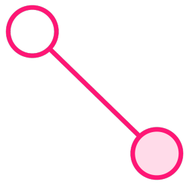


Template methods cannot be changed: the order of methods they call is fixed

Related Patterns



Factory method
Factory method can be viewed as a specialization of template method. Template methods often use factory methods as part of their skeleton structure.



Strategy
Template method allows varying part of an algorithm through inheritance: a static approach. Strategy allows behavior to be switched at runtime, via composition: a dynamic approach.

Summary

Intent of the template method pattern:

- To define the skeleton of an algorithm in an operation, deferring some steps to subclasses

Summary

Implementation:

- Define the template method on the abstract class and don't allow overriding it
- Mark methods that differ per subclass as `abstract`
- Mark methods that differ for some subclasses as `virtual`

Up Next:

Behavioral Pattern: Strategy