Code Up Close

Let's take a closer look at how the AbstractClass is defined, including the template method and primitive operations.

Here we have our abstract class; it is declared abstract and meant to be subclassed by classes that provide implementations of the operations.

Here's the template method. It's declared final to prevent subclasses from reworking the sequence of steps in the algorithm.

```
abstract class AbstractClass {

    final void templateMethod() {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
    }

    abstract void primitiveOperation1();

    abstract void primitiveOperation2();

    void concreteOperation() {
        // implementation here
    }
}
```

The template method defines the sequence of steps, each represented by a method.

In this example, two of the primitive operations must be implemented by concrete subclasses.

We also have a concrete operation defined in the abstract class. This could be overridden by subclasses, or we could prevent overriding by declaring concreteOperation() as final. More about this in a bit...

Code Way Up Close

Now we're going to look even closer at the types of method that can go in the abstract class:

We've changed the templateMethod() to include a new method call.

```
abstract class AbstractClass {

    final void templateMethod() {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
        hook();
    }

    abstract void primitiveOperation1();

    abstract void primitiveOperation2();

    final void concreteOperation() {
        // implementation here
    }

    void hook() {}

}
```

We still have our primitive operation methods; these are abstract and implemented by concrete subclasses.

A concrete operation is defined in the abstract class. This one is declared final so that subclasses can't override it. It may be used in the template method directly, or used by subclasses.

A concrete method, but it does nothing!

We can also have concrete methods that do nothing by default; we call these "hooks." Subclasses are free to override these but don't have to. We're going to see how these are useful on the next page.