This sample demonstrates Interceptor feature in Composer, how to implement an interceptor, and how to tell Composer what to intercept.

Project name: "K.Interceptor"

For information on how to get the code, and run the sample, please see About Basic Samples .

Description

The main program logic (functionality) of this sample is similar to the Notification sample . But it is enhanced with
Composition Listeners
, and
Emitted Classes
as shown in previous examples, to allow it to intercept component communication and print a log of method call hierarchies.

A sample interceptor, called LogToConsoleInterceptor, is implemented to print each entry and exit from methods on the components intercepted. In both BeforeCall and AfterCall methods, information about the method being called (or returned) are printed on the screen.

To place the interceptor around components, a Composition Listener class named LogToConsoleListener is implemented. When a component is created by Composer, the listener is notified. LogToConsoleListener then wraps the component in a dynamically created wrapper object that intercepts method calls to the component, and replaces it in the Composer's context.

Here's how the interception is performed: Since we don't want to write code dependent on the interfaces in the project, the IClassEmitter contract is used to create a dynamic implementation of the component. The dynamically created Type then delegates execution of its methods to the InterceptingAdapterEmittedTypeHandler which is capable of both wrapping the original component, and intercepting the calls.

The LogToConsoleListener that wraps created components are registered using a <RegisterCompositionListener> element in the CalculatorComposition.xml file.

There were no modifications to the components and contracts of this sample compared to the previous example to allow interception mechanism to work.

When you run the sample, the console output shows how the interceptors are working, and when each component is being wrapped by the listener. For example, first three lines of the output is:

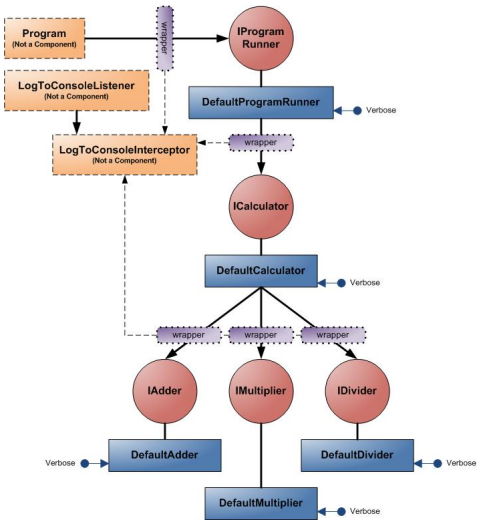
```
  CONSTRUCTOR  - DefaultProgramRunner
LISTENER     - OnComponentCreated: IProgramRunner
             - Wrapping component for logging.
```

This shows the DefaultProgramRunner is being wrapped by the listener as soon as its constructor finishes creating a new instance. The following output snippet is from a method call, where the "Subtract" calculation is being performed:

```
  BEFORE       - ICalculator.Subtract(67, 12)
METHOD CALL  - DefaultCalculator.Subtract(67, 12)
BEFORE       - IAdder.Add(67, -12)
METHOD CALL  - DefaultAdder.Add(67, -12)
AFTER        - IAdder.Add(67, -12) -> 55
AFTER        - ICalculator.Subtract(67, 12) -> 55
67 - 12 = 55
```

It shows how the interceptor is notified before, and after each method call.

Dependency Diagram



Sample Output

```
 CONSTRUCTOR  - DefaultProgramRunner
LISTENER     - OnComponentCreated: IProgramRunner
          - Wrapping component for logging.
CONSTRUCTOR  - DefaultCalculator
LISTENER     - OnComponentCreated: ICalculator
          - Wrapping component for logging.
CONSTRUCTOR  - DefaultAdder
LISTENER     - OnComponentCreated: IAdder
          - Wrapping component for logging.
SET CONFIG   - DefaultAdder.Verbose(True)
LISTENER     - OnComponentComposed: IAdder
NOTIFICATION - DefaultAdder: OnCompositionComplete.
LISTENER     - OnComponentRetrieved: IAdder
SET PLUG     - DefaultCalculator.Adder(IAdder4)
CONSTRUCTOR  - DefaultMultiplier
LISTENER     - OnComponentCreated: IMultiplier
          - Wrapping component for logging.
SET CONFIG   - DefaultMultiplier.Verbose(True)
LISTENER     - OnComponentComposed: IMultiplier
NOTIFICATION - DefaultMultiplier: OnCompositionComplete.
LISTENER     - OnComponentRetrieved: IMultiplier
SET PLUG     - DefaultCalculator.Multiplier(IMultiplier5)
CONSTRUCTOR  - DefaultDivider
LISTENER     - OnComponentCreated: IDivider
          - Wrapping component for logging.
SET CONFIG   - DefaultDivider.Verbose(True)
LISTENER     - OnComponentComposed: IDivider
NOTIFICATION - DefaultDivider: OnCompositionComplete.
LISTENER     - OnComponentRetrieved: IDivider
SET PLUG     - DefaultCalculator.Divider(IDivider6)
SET CONFIG   - DefaultCalculator.Verbose(True)
```

```
LISTENER     - OnComponentComposed: ICalculator
NOTIFICATION - DefaultCalculator: OnCompositionComplete.
LISTENER     - OnComponentRetrieved: ICalculator
SET PLUG     - DefaultProgramRunner.Calculator(ICalculator3)
SET CONFIG   - DefaultProgramRunner.Verbose(True)
LISTENER     - OnComponentComposed: IProgramRunner
NOTIFICATION - DefaultProgramRunner: OnCompositionComplete.
LISTENER     - OnComponentRetrieved: IProgramRunner
BEFORE       - IProgramRunner.Run()

METHOD CALL  - DefaultProgramRunner.Run()

BEFORE       - ICalculator.Add(67, 12)
METHOD CALL  - DefaultCalculator.Add(67, 12)
BEFORE       - IAdder.Add(67, 12)
METHOD CALL  - DefaultAdder.Add(67, 12)
AFTER        - IAdder.Add(67, 12) -> 79
AFTER        - ICalculator.Add(67, 12) -> 79
67 + 12 = 79

BEFORE       - ICalculator.Subtract(67, 12)
METHOD CALL  - DefaultCalculator.Subtract(67, 12)
BEFORE       - IAdder.Add(67, -12)
METHOD CALL  - DefaultAdder.Add(67, -12)
AFTER        - IAdder.Add(67, -12) -> 55
AFTER        - ICalculator.Subtract(67, 12) -> 55
67 - 12 = 55

BEFORE       - ICalculator.Multiply(67, 12)
METHOD CALL  - DefaultCalculator.Multiply(67, 12)
BEFORE       - IMultiplier.Multiply(67, 12)
METHOD CALL  - DefaultMultiplier.Multiply(67, 12)
AFTER        - IMultiplier.Multiply(67, 12) -> 804
AFTER        - ICalculator.Multiply(67, 12) -> 804
67 * 12 = 804

BEFORE       - ICalculator.Divide(67, 12)
METHOD CALL  - DefaultCalculator.Divide(67, 12)
BEFORE       - IDivider.Divide(67, 12)
METHOD CALL  - DefaultDivider.Divide(67, 12)
AFTER        - IDivider.Divide(67, 12) -> 5
AFTER        - ICalculator.Divide(67, 12) -> 5
BEFORE       - ICalculator.Remainder(67, 12)
METHOD CALL  - DefaultCalculator.Remainder(67, 12)
BEFORE       - IDivider.Remainder(67, 12)
METHOD CALL  - DefaultDivider.Remainder(67, 12)
```

```
AFTER       - IDivider.Remainder(67, 12) -> 7
AFTER       - ICalculator.Remainder(67, 12) -> 7
67 / 12 = 5 (with remainder = 7)

AFTER       - IProgramRunner.Run() -> <null>
```