CS 349 Task 5: Visitor and Mediator Patterns

Description

This task demonstrates the Visitor and Mediator patterns as a way of manipulating the Composite, Chain of Responsibility, and Strategy patterns in scalable and extensible ways. The premise is to provide an API for building notional aircraft. As in previous tasks, the specific functionality of the model is not the focus. Except for the structural aspects (and engine power), this code provides data only, no control. Although we do not use the data, for context, it is the notional dimensions and relative position of each three-dimensional box component with respect to its parent. The nature of the components within the context of aircraft is covered in Lecture 26.

The visitor aspect is the primary focus. The intent is to marshal a visitor around an arbitrary compositional aircraft structure and collect the descriptor data above in an XML representation. For example, a notional object of type `MyObject` with identifier "Billy" could be represented in at least three ways, where indentation is irrelevant:

```
<myobject id="Billy"/>
```

or

```
<myobject id="Billy">
</myobject>
```

or

```
<myobject>
  <id>
    Billy
  </id>
</myobject>
```

The mediator aspect is the secondary focus as the engine manager, which serves two roles:

- it coordinates the correct assembly of the engines by ensuring that each wing has the same number of engines.
- it provides collective control over the power of the engines.

Specifications

Your job is to write the code for the following classes and interfaces:

Class Hierarchy

- java.lang.Object
  - **A_Component** (implements I_Identifiable, I_Visitable)
    - **A_ComponentDynamic**
      - **A_ComponentDynamicGear**
        - **ComponentDynamicGearMain**
        - **ComponentDynamicGearNose**
      - **ComponentDynamicAileron**
      - **ComponentDynamicElevator**
      - **ComponentDynamicEngine**
      - **ComponentDynamicFlap**
      - **ComponentDynamicPropeller**
      - **ComponentDynamicRudder**
    - **A_ComponentStatic**
      - **A_ComponentStaticWing**
        - **ComponentStaticWingLeft**
        - **ComponentStaticWingRight**
      - **ComponentStaticAirplane**
      - **ComponentStaticFuselage**
      - **ComponentStaticStabilizerHorizontal**
      - **ComponentStaticStabilizerVertical**
  - **A_Descriptor** (implements I_Visitable)
    - **DescriptorComponent** (implements I_Identifiable)
    - **DescriptorDimensional**
    - **DescriptorPositional**
    - **DescriptorSpatial**
  - **EngineManager**
  - **Test**
  - **Visitor**

Interface Hierarchy

- **I_Identifiable**
- **I_Visitable**

Each class implements the minimum functionality to compose its structure. As such, the description for each is terse and self-explanatory. The format here defines the constructor, followed by any public methods, which generally return something that was provided in the constructor. The XML output is literal except for the italicized entries, which also correspond to the constructor signature. It is not necessary to include newlines, but you are free to do so.

The visitor and sample tester classes are provided under the support link, along with sample XML output.

You may assume that all inputs will be valid, so it is not necessary to implement any defense mechanisms except the engine balance. However, comment your code where you would do so.

A_Component(DescriptorComponent descriptor)

```
getDescriptor()
getID_()
setHost(A_Component host)
getHost()
hasHost()
```

```
<component>
descriptor
</component>
```

A_ComponentDynamic(DescriptorComponent descriptor)

A_ComponentDynamicGear(DescriptorComponent descriptor)

ComponentDynamicGearMain(DescriptorComponent descriptor)

```
<gear-main>
descriptor
</gear-main>
```

ComponentDynamicGearNose(DescriptorComponent descriptor)

```
<gear-nose>
descriptor
</gear-nose>
```

ComponentDynamicAileron(DescriptorComponent descriptor)

```
<aileron>
descriptor
</aileron>
```

ComponentDynamicElevator(DescriptorComponent descriptor)

```
<elevator>
descriptor
</elevator>
```

ComponentDynamicEngine(DescriptorComponent descriptor,
ComponentDynamicPropeller propeller)

getPropeller()
setPower(double power) sets the power of the engine as a percent and prints "engine [*id*] power at *power*".
getPower()

<engine power="*power*">
descriptor
propeller
</engine>

ComponentDynamicFlap(DescriptorComponent descriptor)

<flap>
descriptor
</flap>

ComponentDynamicPropeller(DescriptorComponent descriptor)

<propeller>
descriptor
</propeller>

ComponentDynamicRudder(DescriptorComponent descriptor)

<rudder>
descriptor
</rudder>

A_ComponentStatic(DescriptorComponent descriptor)

ComponentStaticAirplane(DescriptorComponent descriptor,
ComponentStaticFuselage fuselage)

getFuselage()

<airplane>
descriptor
fuselage
</airplane>

```
ComponentStaticFuselage(DescriptorComponent descriptor,
                        ComponentStaticWing wingLeft,
                        ComponentStaticWing wingRight,
                        ComponentStaticStabilizerHorizontal stabilizerLeft,
                        ComponentStaticStabilizerHorizontal stabilizerRight,
                        ComponentStaticStabilizerVertical stabilizerVertical,
                        ComponentDynamicGearNose gear)
```

`getGear()`
`getStabilizerLeft()`
`getStabilizerRight()`
`getStabilizerVertical()`
`getWingLeft()`
`getWingRight()`
`getEngineManager()`, which coordinates the collective power of the engines on the wings

```
<fuselage>
descriptor
wingLeft
wingRight
stabilizerLeft
stabilizerRight
stabilizerVertical
gear
</fuselage>
```

```
ComponentStaticStabilizerHorizontal(DescriptorComponent descriptor,
                                    ComponentDynamicElevator elevator)
```

`getElevator()`

```
<stabilizer-horizontal>
descriptor
elevator
</stabilizer-horizontal>
```

```
ComponentStaticStabilizerVertical(DescriptorComponent descriptor,
                                  ComponentDynamicRudder rudder)
```

`getRudder()`

```
<stabilizer-vertical>
descriptor
elevator
</stabilizer-vertical>
```

```
A_ComponentStaticWing(DescriptorComponent descriptor,
                      java.util.List<ComponentDynamicEngine> engines,
                      java.util.List<ComponentDynamicAileron> ailerons,
                      ComponentDynamicFlap flap,
                      ComponentDynamicGearMain gear)
```

```
getAilerons()
getEngines()
getFlap()
getGear()
```

```
<wing>
descriptor
engines
ailerons
flap
gear
</wing>
```

```
ComponentStaticWingLeft(DescriptorComponent descriptor,
                        java.util.List<ComponentDynamicEngine> engines,
                        java.util.List<ComponentDynamicAileron> ailerons,
                        ComponentDynamicFlap flap,
                        ComponentDynamicGearMain gear)
```

```
ComponentStaticWingRight(DescriptorComponent descriptor,
                         java.util.List<ComponentDynamicEngine> engines,
                         java.util.List<ComponentDynamicAileron> ailerons,
                         ComponentDynamicFlap flap,
                         ComponentDynamicGearMain gear)
```

```
A_Descriptor()
```

```
DescriptorComponent(String id,
                    DescriptorSpatial descriptor)
```

```
getDescriptorSpatial()
getID_()
```

```
<descriptor id="id">
descriptor
</descriptor>
```

```
DescriptorDimensional(double height,
                      double width,
                      double depth)
```

```
getDepth()
getHeight()
getWidth()
```

```
<dimensions height="height" width="width" depth="depth"/>
```

```
DescriptorPositional(double x,
                     double y,
                     double z)
```

```
getX()
getY()
getZ()
```

```
<position x="x" y="y" z="z"/>
```

```
DescriptorSpatial(DescriptorPositional position,
                  DescriptorDimensional dimensions)
```

```
getDimensions()
getPosition()
```

```
<descriptor-spatial>
position
dimensions
</descriptor-spatial>
```

```
I_Identifiable
```

```
getID_()
```

```
I_Visitable
```

```
visit_(Visitor visitor)
```

```
EngineManager(ComponentStaticFuselage host)
```

`getHost()`

`void registerEngineLeft(ComponentDynamicEngine engine)` keeps track of the engines being managed on the left wing.

`void registerEngineRight(ComponentDynamicEngine engine)` keeps track of the engines being managed on the right wing.

`List<ComponentDynamicEngine> getEngines()` returns the engines being managed.

`void setPower(double power)` sets the power of the engines being managed.

`void commit()` validates whether the engines are balanced. If so, it prevents any further engines from being registered; otherwise, throw a `RuntimeException` with an appropriate message.

Deliverables

Submit all your source files in a zip file.

It is not necessary to comment your code. Do not package your code. You may add any code to facilitate your design, but you must not deviate from the specifications above. Code that does not compile will not be graded.

Submit a plain-text `readme.txt` that indicates what does not work. If you believe everything works, indicate so.