6.170
Design Project Experiences:
Space Elevator Simulator
Part II

Image removed due to copyright restrictions.

1

"Realistic" Space Elevator
Edwards & Westling, 2002

- **Spacecraft launched into geosynchronous (35,000km) orbit.**
- **Spacecraft lowers thin ribbon toward ground, and moves outward to keep it from falling, eventually ending up at 100,000 km to act as a counterweight.**

Image removed due to copyright restrictions.

- **When ribbon reaches earth, it is tied to a base station (floating platform off the coast of Ecuador).**
- **To strengthen and widen the initial ribbon, climbers, powered by lasers from earth stitch on additional ribbons for 2½ years.**
- **Ribbon is 3 feet wide and supports 13 tons in completed elevator.**

2

Space Elevator Quotes

“The space elevator will be built about 50 years after everyone stops laughing.”
- Arthur C. Clarke, 1985

"It’ll be built 10 years after everybody stops laughing … and I think they have stopped laughing," Arthur C. Clarke, 2003, 2nd Annual Space Elevator conference, New Mexico.

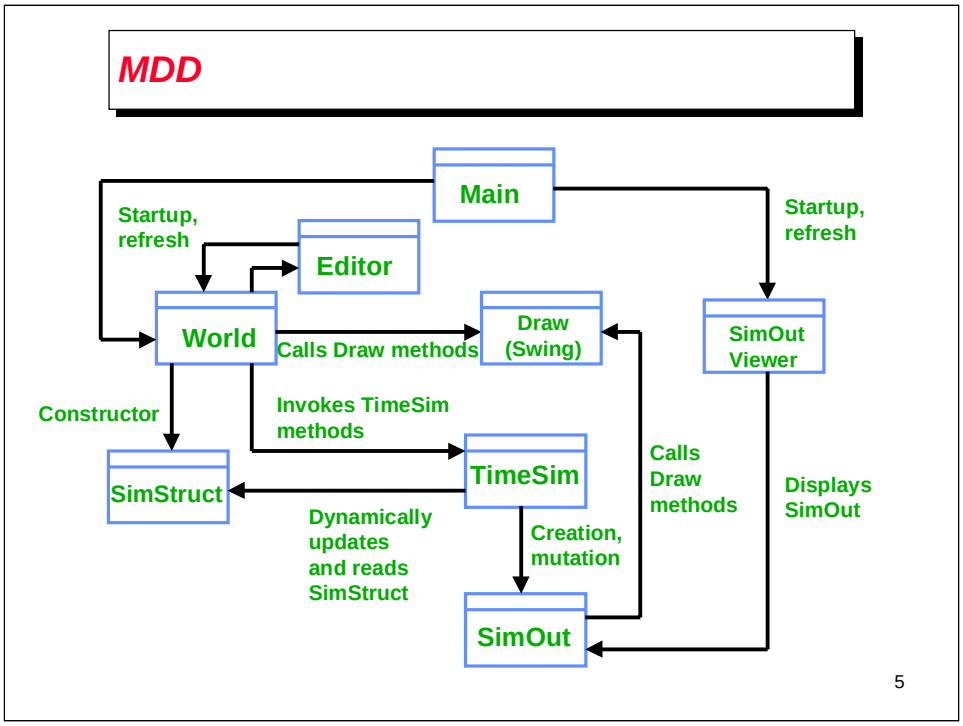
3

Functionality

- **Simulate and view space elevator dynamics**
- **Main objects are Central planet, cable, counterweight, climber**
  - **Cable modeled as masses and springs**
- **Main forces are gravitational, centrifugal and coriolis**
- **Different views:**
  - **Text file**
  - **Swing**
  - → **Java 3D**

4

Because of Coriolis force, the object does not actually deviate from its path, but it appears to do so because of the motion of the coordinate system.



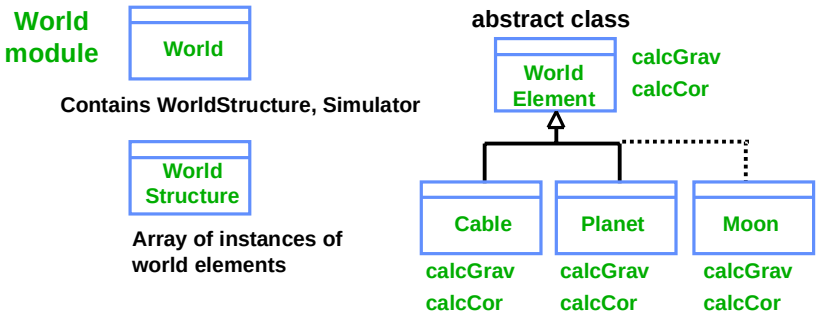
Let's have objects in the World call Draw methods and draw themselves. This makes for a more extensible world because we can add modules and add their draw methods.

Similarly, we will have objects in World call the simulator methods to simulate their dynamics.

In this MDD it is clear that World is the central module/ADT. World and Editor are tightly coupled since the World will need to invoke Editor methods to edit its objects, and the editor will need to invoke World methods to update the objects with new ones, or update object positions.

World objects call methods in Draw and TimeSim to draw or simulate themselves.

Class Hierarchy (partial)



- **Want to define *new* subclasses of WorldElement**
- **Want to define *new* operations on all or subset of WorldElement subclass objects**

6

Let's look at the class hierarchy for the viewer part of the Space Elevator (SE) simulator.

We have a World class, a WorldStructure class and an abstract WorldElement class. A World object contains a WorldStructure object and a Simulator object.

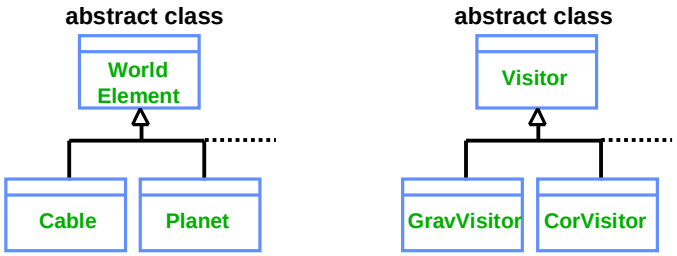
WorldElements can be instantiated as planets, cables, masses, and connectors (springs).

If we want to define a new subclass of WorldElement it is easy to do this, and the subclass will inherit the skeletal implementation of the abstract parent class.

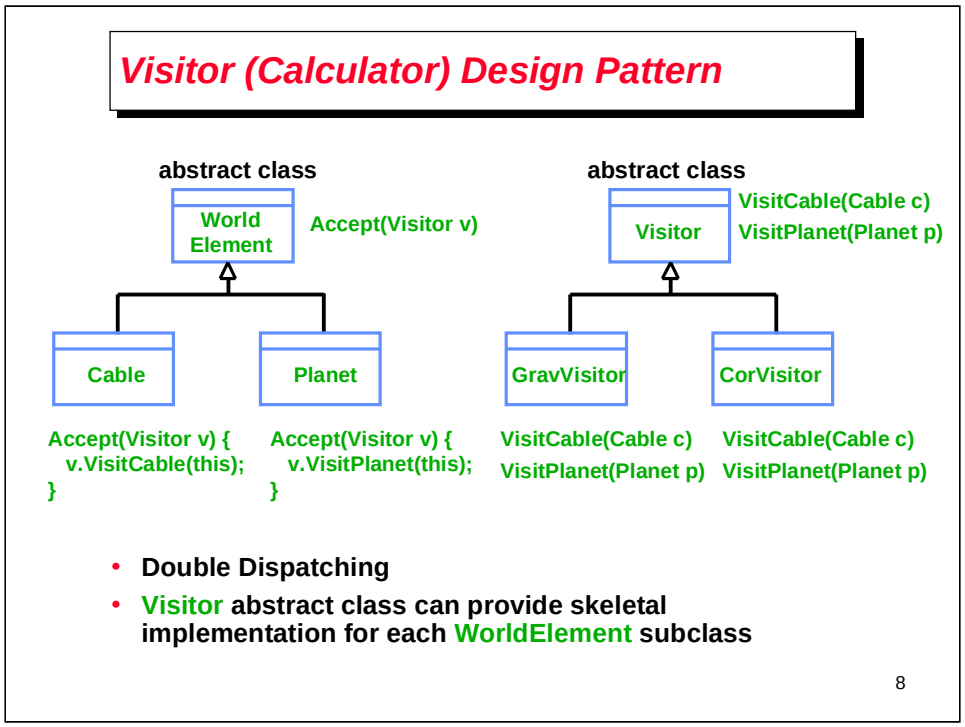
However, defining a new operation for all WorldElements is harder because we have to first provide a skeletal implementation for the abstract class, and then provide implementations for each of the subclasses.

Visitor Design Pattern

- **Set up WorldElement so arbitrary operations can be added easily**
- **Factor out the operations in a separate hierarchy of classes, all extending the abstract class Visitor**
  - **Each subclass represents one operation**



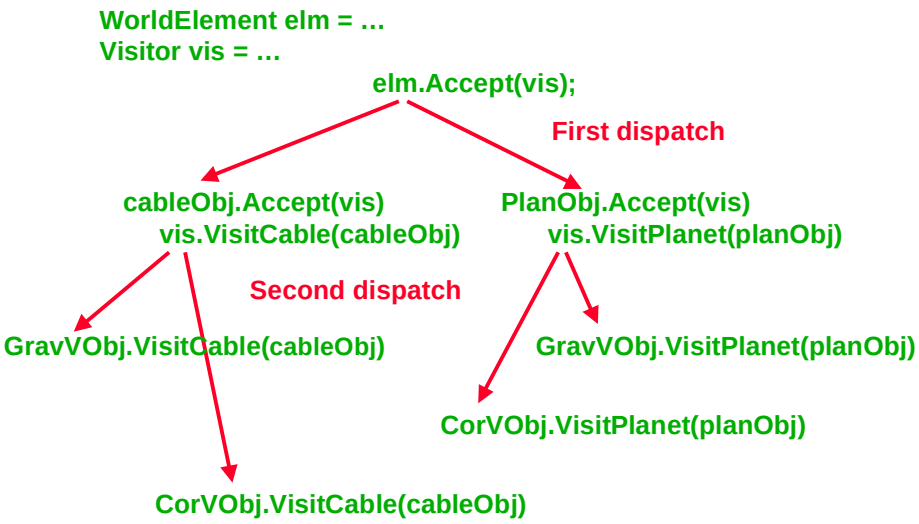
7



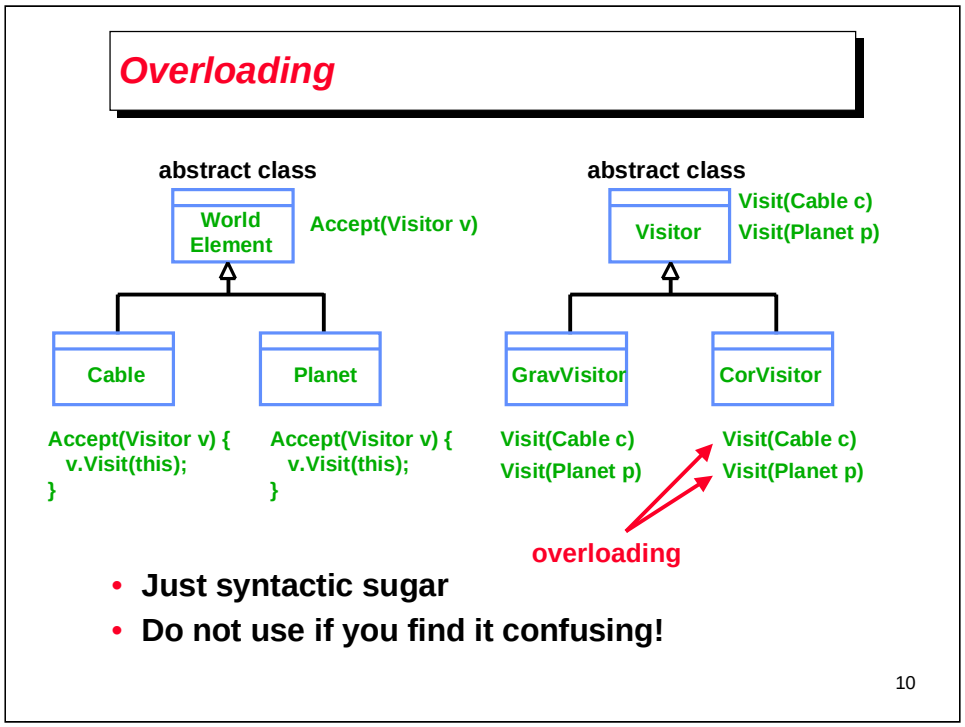
For example, in a cable, computing the tension in the cable. Then the gravitational force is added to the tension by the GravVisitor subclass.

The Coriolis force is added in the CorVisitor subclass.

Double Dispatching

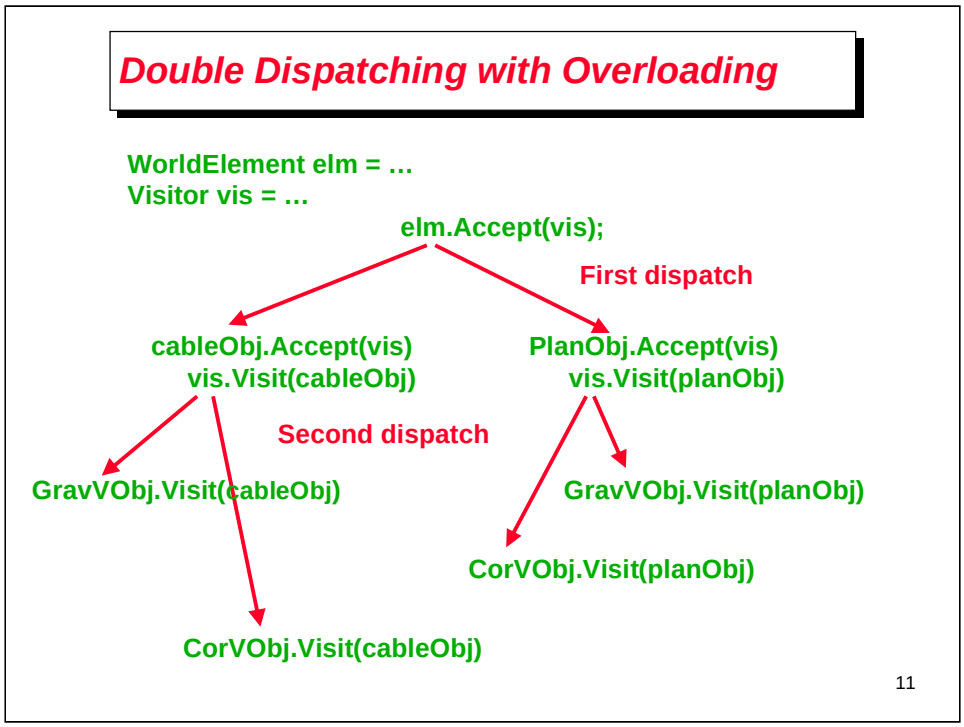


9



For example, in a cable, computing the tension in the cable. Then the gravitational force is added to the tension by the GravVisitor subclass.

The Coriolis force is added in the CorVisitor subclass.



The Java dispatcher will do the right thing when you overload method names. Depending on the argument type, the appropriate method will be called.

Visitor Usage

- **In the TimeSim simulator module, we can now write code like:**

```
for(i = 0; i < WorldStructureObj.Elements.size(); i++) {
    elm = WorldStructureObj.Elements.get(i);
    for(j = 0; j < visitorsVector.size(); j++) {
        vis = visitorsVector.get(j);
        elm.Accept(vis);
    }
}
```

- **The above code does *not* have to change even if we add new WorldElement subclasses or new Visitor subclasses (operations)**
  - **Need to add visit methods for each new element in each Visitor subclass**

12

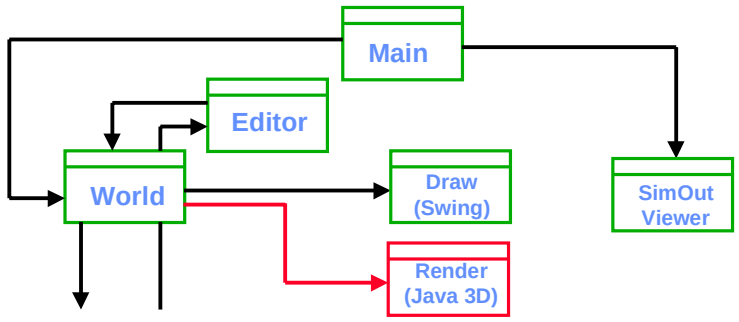
Accept is defined appropriately for each WorldElement concrete subclass and operation pair. It could be a no-op, that is, it may not do anything.

Visitor makes it easy to add new operations, but it makes it harder to add new elements. Each new element must get its VisitElement() method in each Visitor class.

Three other points:

- A Visitor can visit elements that are unrelated through inheritance. In our example Cable and Planet do not need to be related.
- A Visitor can accumulate state as they visit each element of an object structure in an instance variable of the visitor. In normal operations such state would have to be passed as arguments to the operation.
- A Visitor may need access to state that otherwise may be declared private.

Viewer Functionality



- **The Java 3D API provides a set of object-oriented interfaces to build, render, and control the behavior of 3D objects.**

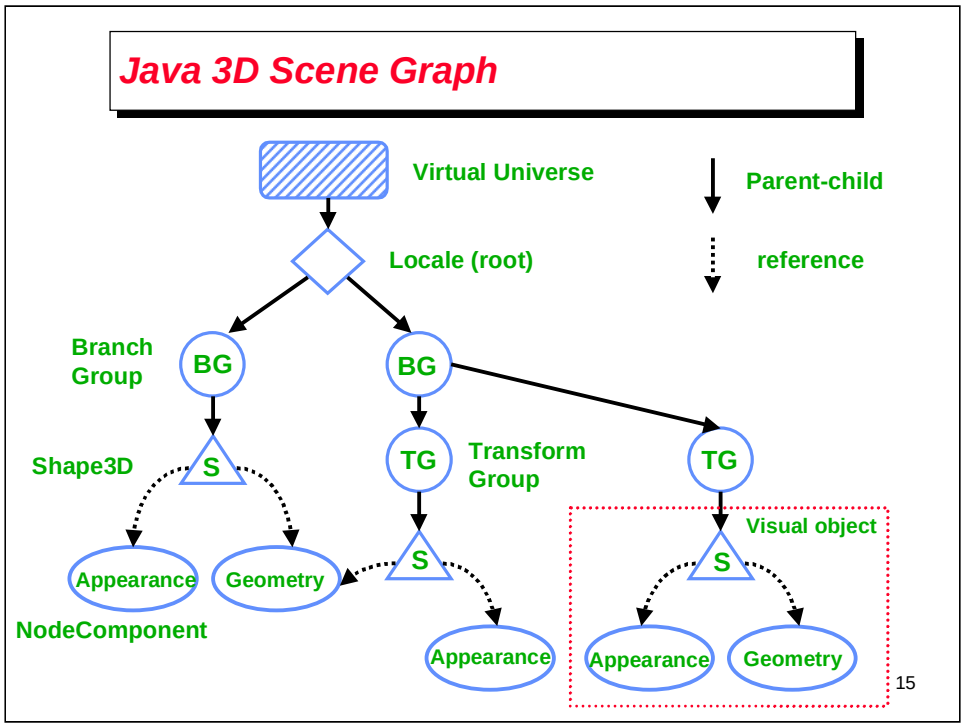
13

Today, we'll look briefly at the Java 3D functionality in the simulator.

Java 3D API

- **Hierarchy of Java classes which serve as an interface to a 3D rendering system**
- **Geometric objects reside in a virtual universe**
- **Details of rendering handled automatically**
  - **Objects rendered in parallel using Java threads**
- **A Java 3D program creates instances of objects and places them a tree structure called a scene graph**
  - **Scene graph completely specifies the contents of the virtual universe and how it is to be rendered**

14



Only a single virtual universe in a Java 3D program. Can have multiple but they cannot communicate with each other.

Locale objects are landmarks in the Virtual Universe. Typically, there is only one Locale object in a Virtual Universe.

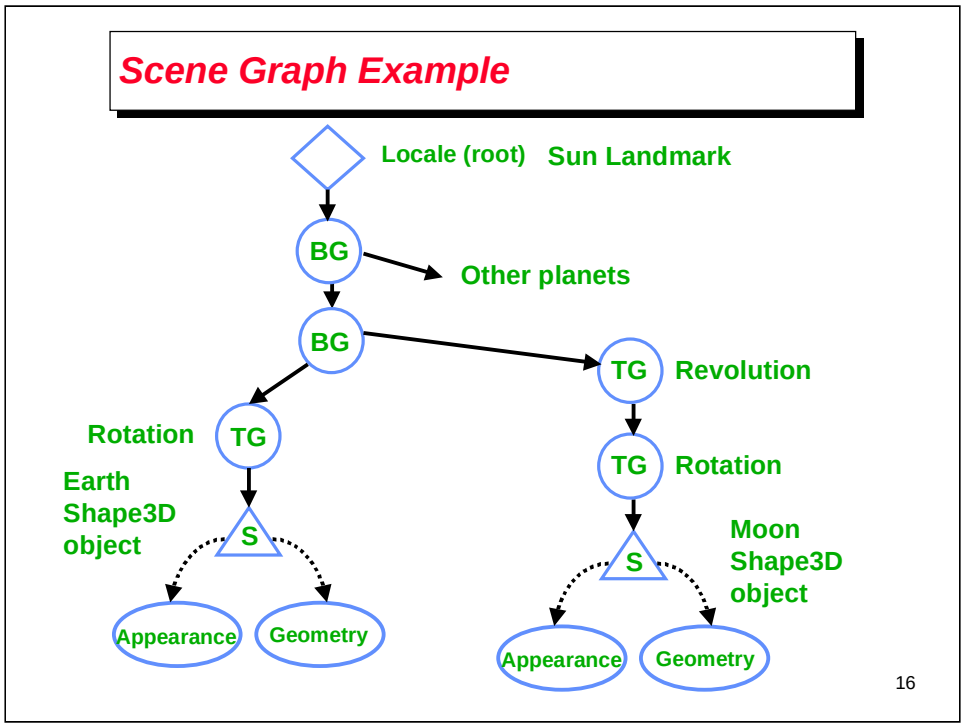
Branch Group objects collect together a set of visual objects (or Shape3D objects).

Transform Group objects specify the transformations that can be performed on the child Shape3D objects.

The visual Shape3D object such as a cube or a sphere consists of NodeComponent references such as Appearance and Geometry.

Each scene graph is rooted at a Locale object.

The fundamental constraint in a Java 3D scene graph is that each Shape3D object has to have a single path to the root Locale object.



Here is a simple example of a scene graph representing the Moon rotating around the Earth, and both of them rotating around their own axes.

The Locale object represents the Sun, the reference in this virtual universe (or solar system). Note how two successive transforms can be applied to a Shape3D object, in this case the moon.

The entire Branch Group could be transformed through another Transform Group (TG) to create rotation of the earth around the Sun.

Project "Report"

- **Significant discussions prior to coding**
  - **What should the functionality be?**
    - » **Not an issue in Gizmoball/AntiChess**
  - **2 Weeks of MDDs, Class Hierarchies and Object models**
- **About 3600 lines of code**
- **Getting the physics "right" took a while**
  - **Some of the physics is still questionable**
  - **Not an issue in Gizmoball/AntiChess**
- **Lots of fun, especially because Matt and Lee debugged my code!**

17

Nine Stages of My Design Project

- **Wild Euphoria.**
- **Growing Concern.**
- **Near Total Disillusionment.**
- **Unmitigated Disaster.**
- **Search For The Guilty.**
- **Punishment Of The Innocent.**
- **Scream for Professional Help.**
- **Declare Victory!**
- **Blatant Self-Promotion.**

18

Demonstration

19