CptS 487
Software Design and Architecture

Lesson 27

Design Patterns 11:

Proxy & Visitor

World Class. Face to Face.

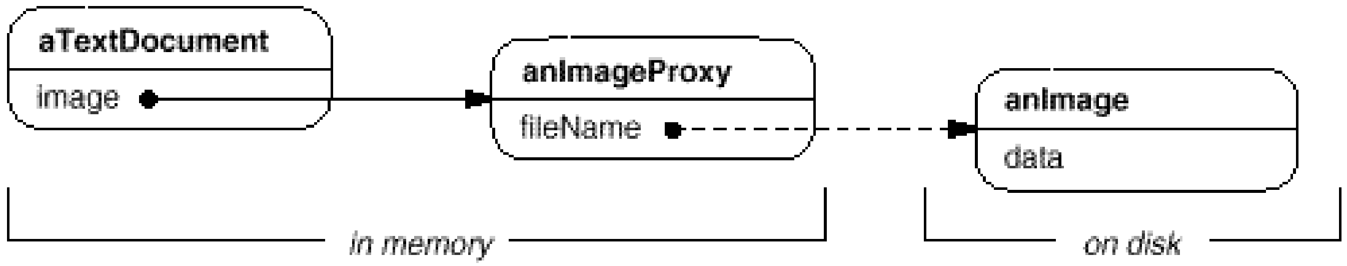
Instructors:

3. Proxy (Object structural pattern)

- Intent
  - Provide a surrogate or placeholder for another object to control access to it
- Also Known As
  - Surrogate
- Motivation
  - A proxy is
    - a person authorized to act for another person
    - an agent or substitute
    - the authority to act for another

3. Proxy

- Motivation
  - There are situations in which a client does not or can not reference an object directly, but wants to still interact with the object
  - A proxy object can act as the intermediary between the client and the target object
  - The proxy object has the same interface as the target object
  - The proxy holds a reference to the target object and can forward requests to the target as required (delegation!)
  - In effect, the proxy object has the authority to act on behalf of the client to interact with the target object



3. Proxy

- Types of Proxies

  Proxy is applicable whenever there is need for a more sophisticated reference to an object than a simple pointer.

  — Remote Proxy - Provides a reference to an object located in a different address space on the same or different machine

  — Virtual Proxy - Allows the creation of a memory intensive object on demand. The object will not be created until it is really needed.

  — Protection (Access) Proxy - Provides different clients with different levels of access to a target object.
    - Ex: KernelProxies in the Choices operating system

  — Cache Proxy - Provides temporary storage of the results of expensive target operations so that multiple clients can share the results

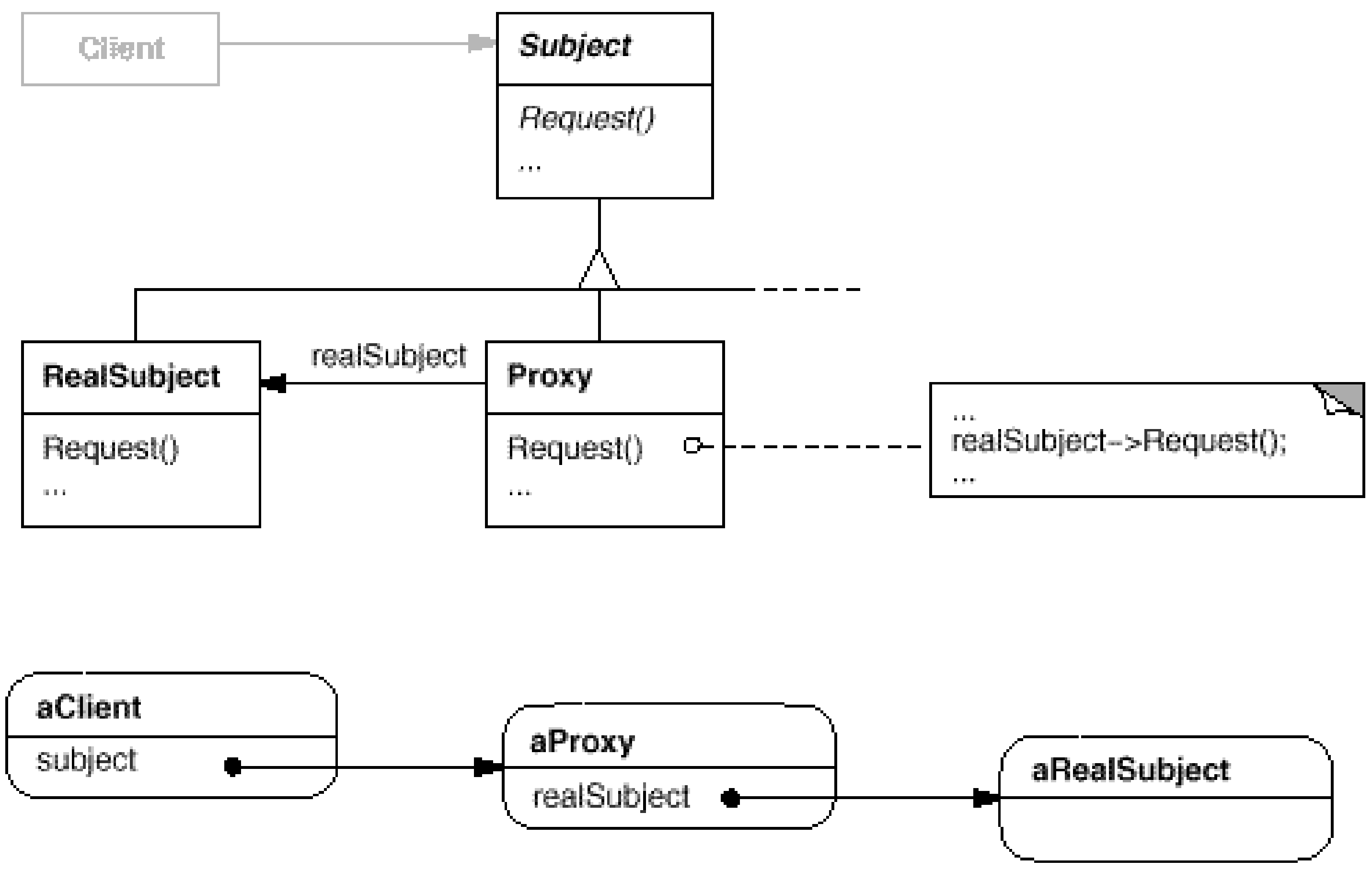
  — Firewall Proxy - Protects targets from bad clients (or vice versa)

  — Synchronization Proxy – Controls the number of clients that simultaneously access a server. For example, a file server proxy ensures that two clients don't attempt to write to the same file at the same time.

  — Smart Reference Proxy - Provides additional actions whenever a target object is referenced such as counting the number of references to the object

3. Proxy

- Structure



3. Proxy

- Types of Proxies

  Proxy is applicable whenever there is need for a more sophisticated reference to an object than a simple pointer.

  - Remote Proxy - Provides a reference to an object located in a different address space on the same or different machine
  - **Virtual Proxy** - Allows the creation of a memory intensive object on demand. The object will not be created until it is really needed.
  - Protection (Access) Proxy - Provides different clients with different levels of access to a target object.
    - Ex: KernelProxies in the Choices operating system
  - Cache Proxy - Provides temporary storage of the results of expensive target operations so that multiple clients can share the results
  - Firewall Proxy - Protects targets from bad clients (or vice versa)
  - **Synchronization Proxy** – Controls the number of clients that simultaneously access a server. For example, a file server proxy ensures that two clients don't attempt to write to the same file at the same time.
  - Smart Reference Proxy - Provides additional actions whenever a target object is referenced such as counting the number of references to the object

Virtual Proxy Example

- Scenario: Consider an image viewer program that lists and displays high resolution photos. The program has to show a list of all photos however it does not need to display the actual photo until the user selects an image item from a list.
- Solution: Use a Virtual Proxy!
- When using a Virtual Proxy:
  - All classes other than the proxy itself must access the target class indirectly through the proxy.

Virtual Proxy Example

- Image interface representing the Subject :
  - The interface has a single method `showImage`() that the Concrete Images must implement to render an image to screen.

```
// Subject interface
 public interface iImage{
     public void showImage();
}
```

Virtual Proxy Example

- The Proxy implementation
  - The image proxy is a virtual proxy that creates and loads the actual image object on demand, thus saving the cost of loading an image into memory until it needs to be rendered

```
// Proxy.
 public class ImageProxy implements iImage {
  /* Private Proxy data */
  private String imageFilePath;

  /* Reference to RealSubject */
  private Image proxifiedImage;

  public ImageProxy(String imageFilePath) {
       this.imageFilePath= imageFilePath;
  }

  public void showImage() {
       /* create the Image Object only when the image is */
       /*required to be shown*/
       proxifiedImage = new HighResolutionImage(imageFilePath);
       /* now call showImage on realSubject*/
       proxifiedImage.showImage();
  }
}
```

Virtual Proxy Example

- Real subject implementation
  - Concrete and heavyweight implementation of the image interface. The High resolution image, loads a high resolution image from disk, and renders it to screen when showImage() is called

```
// RealSubject
public class HighResolutionImage implements iImage {

   public HighResolutionImage(String imageFilePath) {
        loadImage(imageFilePath);
   }

   private void loadImage(String imageFilePath) {
        /* load Image from disk into memory*/
        /* this is heavy and costly operation*/
   }

   public void showImage() {
        /* Actual Image rendering logic*/
   }

}
```

Virtual Proxy Example

- Image Viewer program
  - the program simply loads two images, and renders only one image, once using the proxy pattern, and another time directly

```
// Image Viewer program
public class ImageViewer {

  public static void main(String[] args) {

  // assuming that the user selects a folder that has 2 images
  //create the 2 images
  iImage highResolutionImg1 = new imageProxy("veryHighResImg1.jpg");
  iImage highResolutionImg2 = new ImageProxy("veryHighResImg2.jpg");


  highResolutionImg1.showImage();


  …will continue on the next slide
```

$\longleftarrow$

- Assume that the user clicks on Image one item in a list.
- This would cause the program to call `showImage()` for that image only
- Note that in this case only image one was loaded into memory

Virtual Proxy Example

- Image Viewer program

```
   // consider using the high resolution image object directly
   iImage highResolutionImgNoProxy1 = new
                HighResolutionImage(“sample/veryHighResImg1.jpg");
   iImage highResolutionImgNoProxy2 = new
                HighResolutionImage("sample/veryHighResImg2.jpeg");


   highResolutionImgNoProxy2.showImage();

}

}
```

- Assume that the user selects image two item from images list
- In this case all images have been loaded into memory and not all have been actually displayed
- This is a waste of memory resources

Synchronization Proxy Example

- **Scenario**: A class library provides a Table class, but does not provide a capability to allow clients to lock individual table rows. We do not have the source code for this class library, but we have complete documentation and know the interface of the Table class. How can we provide a row locking capability for the Table class?
  - Solution: Use a Synchronization Proxy!

Synchronization Proxy Example

- We create a synchronization proxy for `Table` class which provides a row locking capability for the `Table` class
  - **`RowLockTableProxy`**
- Part of the `Table` class, just so we can see its interface:

```
// Real-subject
 public class Table implements ITable {
   ….

     // Get the element.
     public Object getElementAt(int row, int column) {}

     // Set the element.
     public void setElementAt(Object element, int row,
     int column ) {}

     public int getNumberOfRows() {return numrows;}
}
```

Synchronization Proxy Example

- Here is the table proxy:

```
public class RowLockTableProxy implements iTable {

    private Table realTable;
    private Integer[] locks;

    public RowLockTableProxy(Table toLock) {
       realTable = toLock;
       locks = new Integer[toLock.getNumberOfRows()];
       for (int row = 0; row<toLock.getNumberOfRows(); row++)
             locks[row] = new Integer(row);
    }

   public Object getElementAt(int row, int column) {
       synchronized (locks[row]) {
             return realTable.getElementAt(row, column);
        }
}
```

The synchronized keyword in Java creates a block of code referred to as a critical section

Synchronization Proxy Example

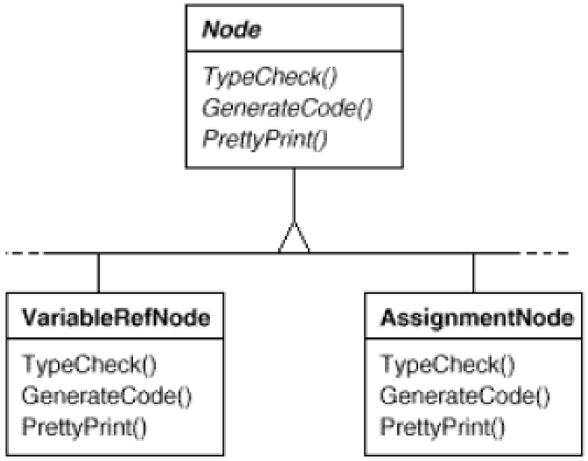
- Table proxy continued:

```
  public void setElementAt(Object element, int row, int column){
        synchronized (locks[row]) {
              return realTable.setElementAt(element, row, column);
        }
  }
  public int getNumberOfRows() {
        return realTable.getNumberOfRows();
  }

}
```

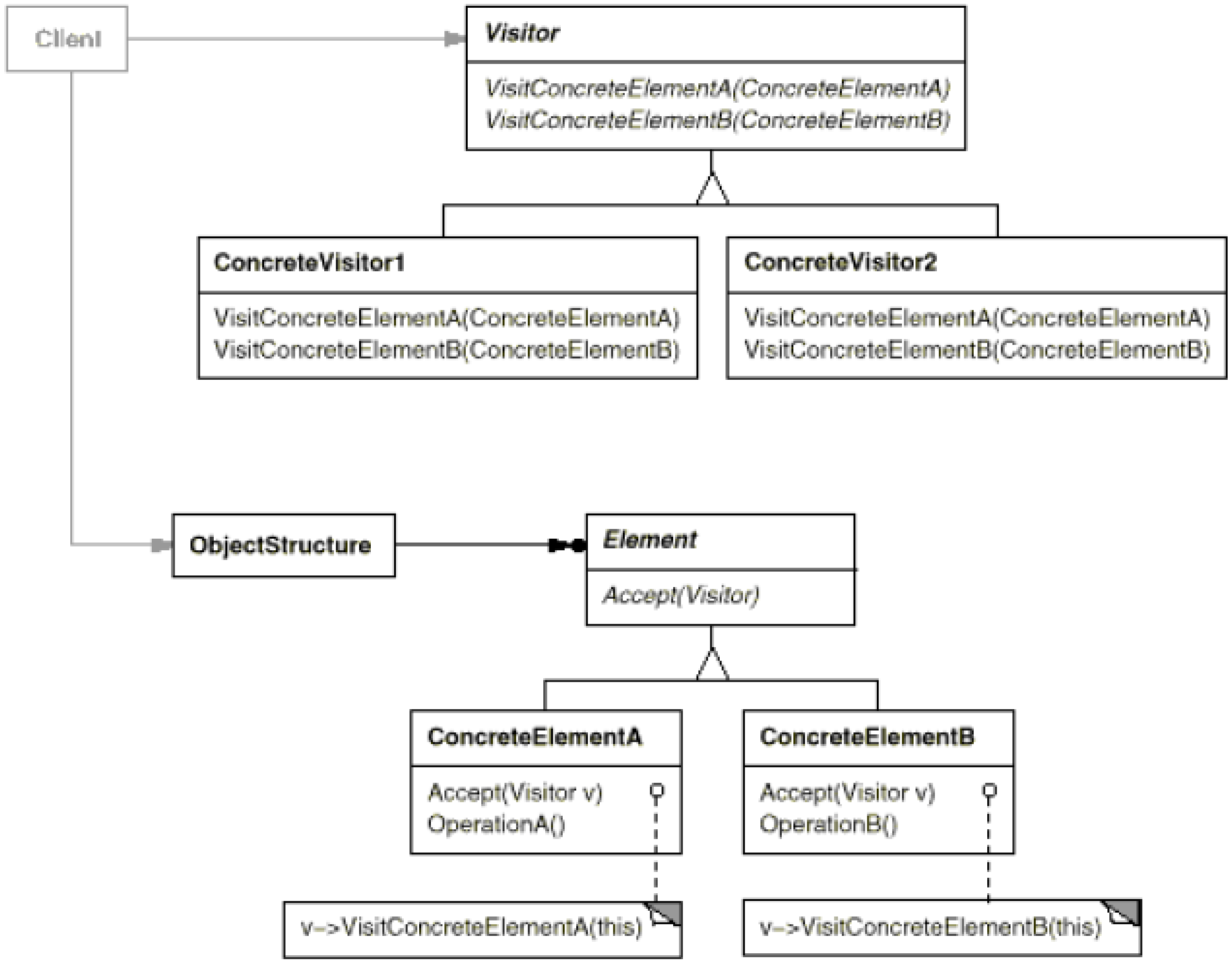
Visitor (Object behavioral pattern)

- Intent
  - Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.
- Motivation
  - A compiler might need to deal with a note on an abstract syntax tree, with operations such as type check, code generation, etc.
  - However, it will need to treat the variable nodes and expression nodes differently.
  - It will ask a “visitor” to perform the operations for it.



Visitor (Object behavioral pattern)

- Structure

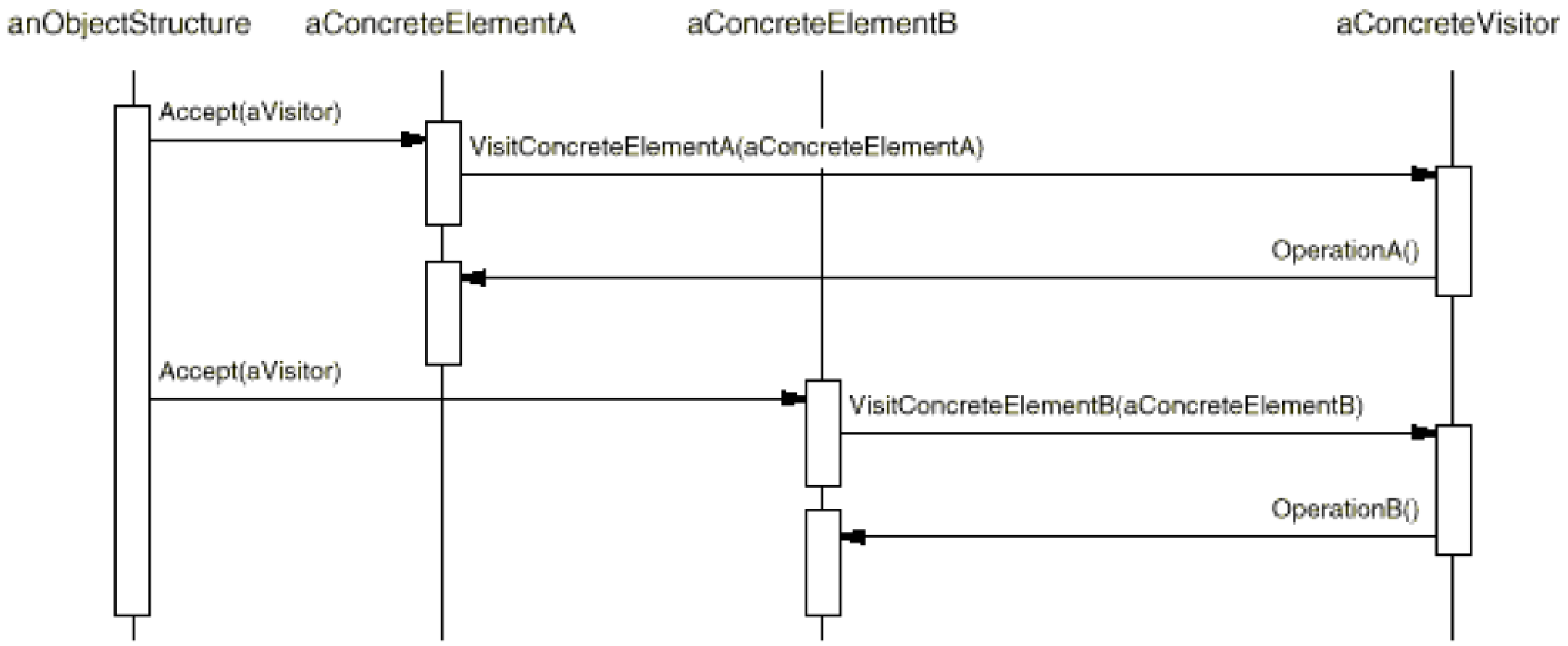

Client
Visitor
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor1
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor2
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ObjectStructure
Element
Accept(Visitor)
ConcreteElementA
Accept(Visitor v)
OperationA()
ConcreteElementB
Accept(Visitor v)
OperationB()
v->VisitConcreteElementA(this)
v->VisitConcreteElementB(this)

Visitor (Object behavioral pattern)

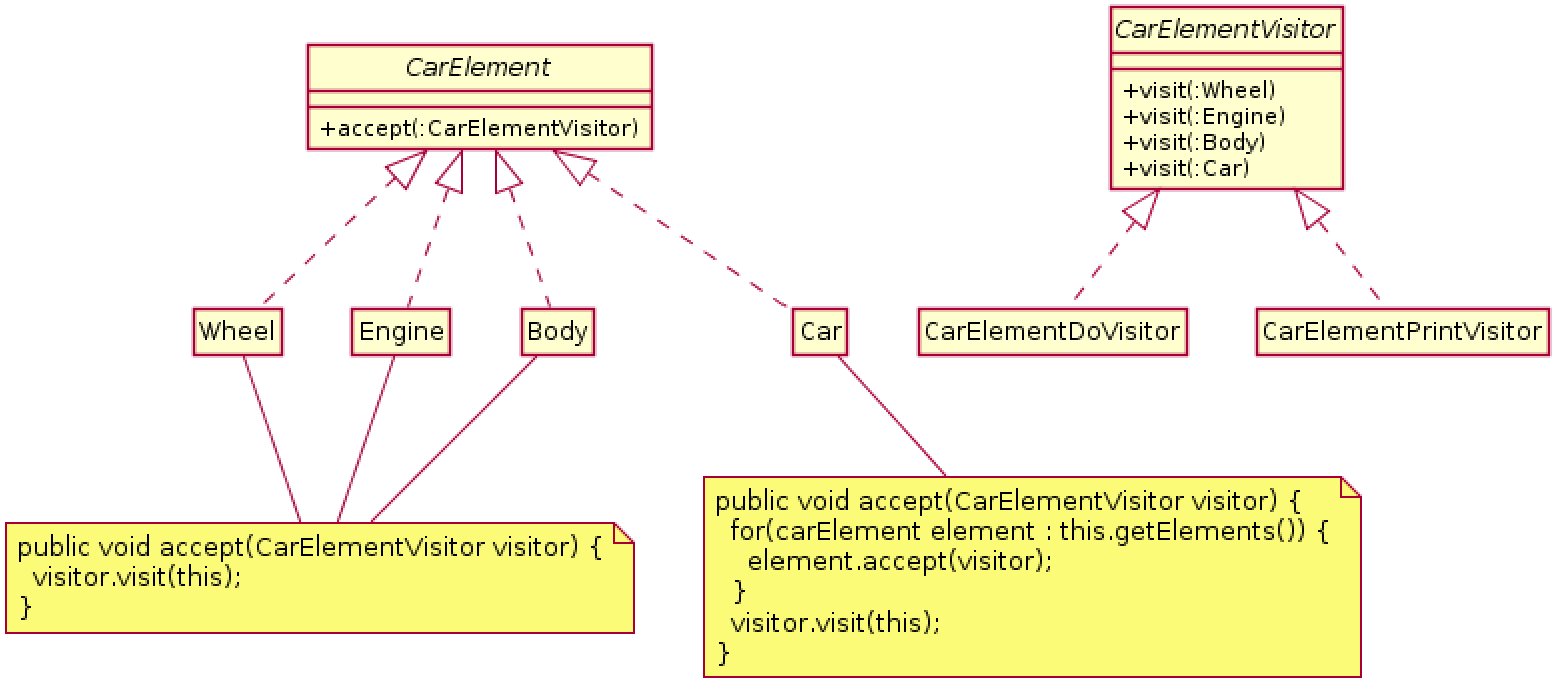
- Participants
  - Visitor
    - declares a Visit operation for each class of ConcreteElement in the object structure. The operation's name and signature identifies the class that sends the Visit request to the visitor. That lets the visitor determine the concrete class of the element being visited. Then the visitor can access the element directly through its particular interface.
  - ConcreteVisitor
    - implements each operation declared by Visitor.
  - Element
    - Defines an Accept operation that takes a visitor as an argument
  - ConcreteElement
    - Implements said Accept operation
  - ObjectStructure
    - Enumerate its elements.
    - May provide a high-level interface to allow the visitor to visit its elements.
    - May either be a composite or a collection.

Visitor (Object behavioral pattern)

- Collaborations



Visitor Example (from wiki)



```
public void accept(CarElementVisitor visitor) {
  visitor.visit(this);
}
```

```
public void accept(CarElementVisitor visitor) {
  for(carElement element : this.getElements()) {
    element.accept(visitor);
  }
  visitor.visit(this);
}
```

Visitor (Object behavioral pattern)

- Applicability
  - When to use it?
  - an object structure contains many classes of objects with differing interfaces, and you want to perform operations on these objects that depend on their concrete classes.
  - lets you keep related operations together by defining them in one class. When the object structure is shared by many applications, use Visitor to put operations in just those applications that need them.
  - the classes defining the object structure rarely change, but you often want to define new operations over the structure.

Visitor (Object behavioral pattern)

- Consequences
  - Visitor makes adding new operations easy
  - A visitor gathers related operations and separates unrelated ones.
  - Adding new ConcreteElement is hard
  - Visiting across class hierarchies.
  - Accumulating state
  - Breaking encapsulation

Bonus Content

- https://cppcrypt.tumblr.com/post/168134225802
- Click “Next” to see the whole thing!