Contents

1. Template:

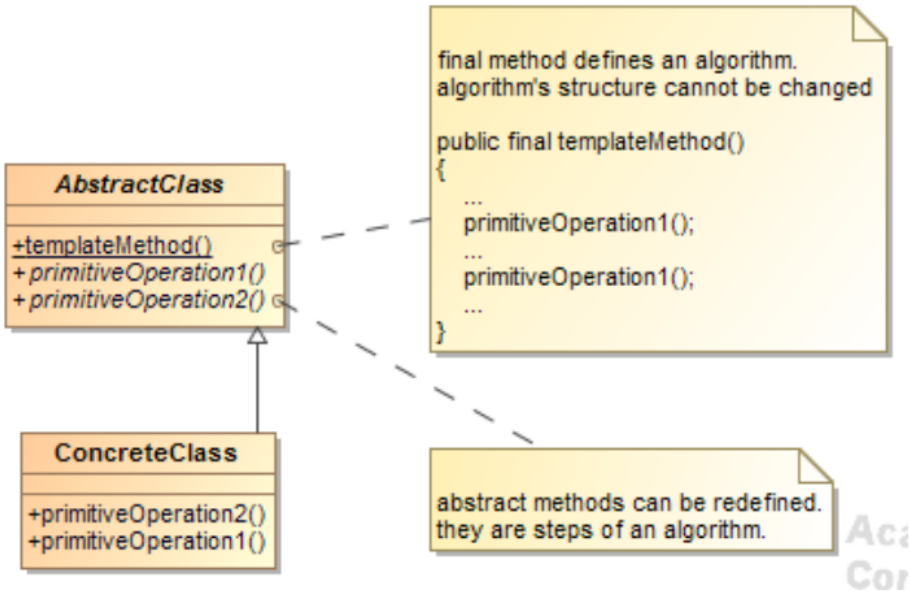
When to use “Template method”, motivation

- when you’ve got an **algorithm of several steps** and you want to allow customization by subclasses

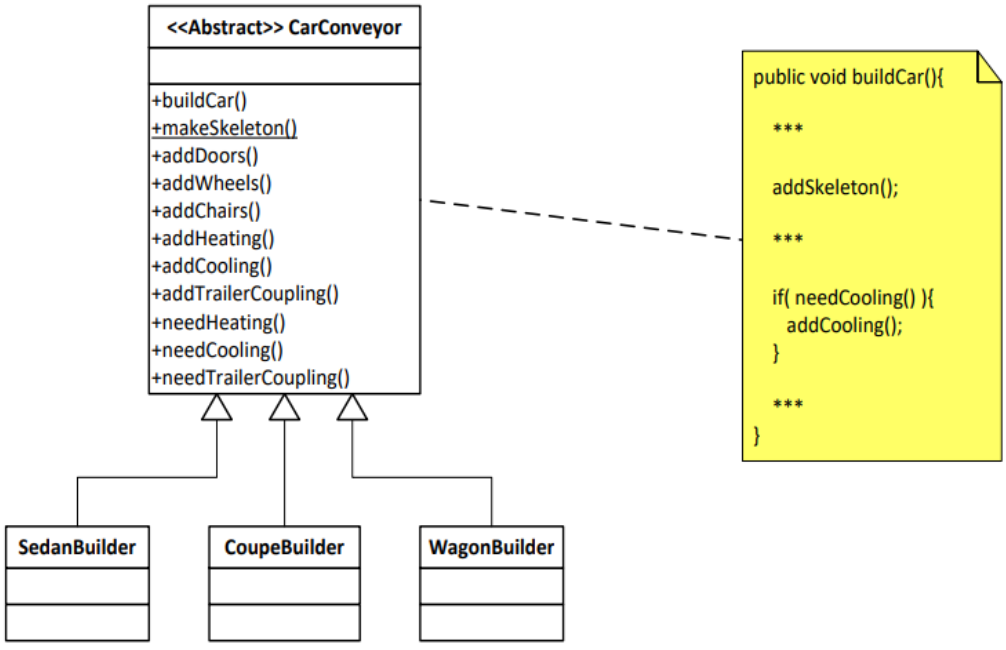
- when common behavior among subclasses should be factored and localized in a common class to avoid code duplication “***refactoring to generalize***”

- Fundamental technique for code reuse (**libraries**) because they are the means for factoring out common behavior

Template method's structure (GoF)



Template method example



CarConveoyr:

```
public final void buildCar(){
    addDoors();

    if(needParking()){
        addParking();
    }
    if(needHeating() ){
        addHeating();
    }
    if(needConditioning()){
        addConditioning();
    }
    if(needTrailerCoupling()){
        addTrailerCoupling();
    }
}

public abstract boolean needTrailerCoupling() ;

public abstract boolean needConditioning() ;

public abstract boolean needHeating() ;

public abstract boolean needParking() ;

public abstract void addDoors();

public abstract void addParking();

public abstract void addHeating();

public abstract void addConditioning();

public abstract void addTrailerCoupling();
```

CoupeConveyor:

```
public class CoupeConveyor extends CarConveoyr {

    @Override
    public boolean needTrailerCoupling() {      return false;   }

    @Override
    public boolean needConditioning() {     return false;   }

    @Override
    public boolean needHeating() {      return false;   }

    @Override
    public boolean needParking() {      return true;    }

    @Override
    public void addDoors() {        System.out.println("adding 2 doors");

    }

    @Override
    public void addParking() {      System.out.println("adding parking to coupe");

    }

    @Override
    public void addHeating() { //some code  }

    @Override
    public void addConditioning() { //some code }

    @Override
    public void addTrailerCoupling() { //some code }

}
```

SedanConveyor:

```
public class SedanConveyor extends CarConveoyr {

    @Override
    public boolean needTrailerCoupling() {      return false;   }

    @Override
    public boolean needConditioning() {     return false;   }

    @Override
    public boolean needHeating() {      return true;    }

    @Override
    public boolean needParking() {  return true;    }

    @Override
    public void addDoors() {
        System.out.println("adding 4 doors");
    }

    @Override
    public void addParking() {
        System.out.println("adding parking");
    }

    @Override
    public void addHeating() {
        System.out.println("add heating");
    }

    @Override
    public void addConditioning() { //do nothing    }

    @Override
    public void addTrailerCoupling() {  }
}
```

WagonConveyor:

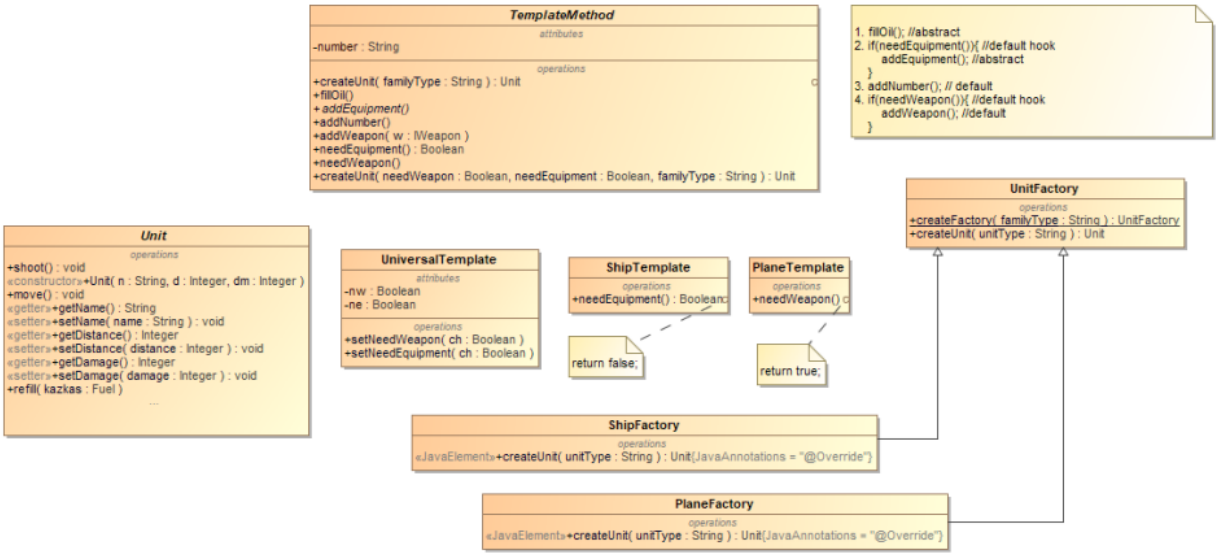
```
public class WagonConveyor extends CarConveoyr {

    @Override
    public boolean needTrailerCoupling() {      return true;    }

    @Override
    public boolean needConditioning() {     return true;    }

    @Override
    public boolean needHeating() {      return true;    }

    @Override
    public boolean needParking() {      return false;   }

    @Override
    public void addDoors() {
        System.out.println("adding 5 doors");
    }

    @Override
    public void addParking() {  }

    @Override
    public void addHeating() {
        System.out.println("adding heating to wagon");
    }

    @Override
    public void addConditioning() {
        System.out.println("adding conditioning to wagon");
    }

    @Override
    public void addTrailerCoupling() {
        System.out.println("adding coupling to wagon");
    }
}
```

Main:

```
public class CarsFactory {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        CarConveoyr sedanConveyor = new SedanConveyor();
        CarConveoyr wagonConveyor = new WagonConveyor();
        CarConveoyr coupeConveyor = new CoupeConveyor();

        System.out.println("Building sedan ============");
        sedanConveyor.buildCar();
        System.out.println("Building wagon ============");
        wagonConveyor.buildCar();
        System.out.println("Building coupe ============");
        coupeConveyor.buildCar();

    }

}
```

Template method : example to code

- Added template method `CreateEnemyTemplate()` to class



Template method: created during lecture

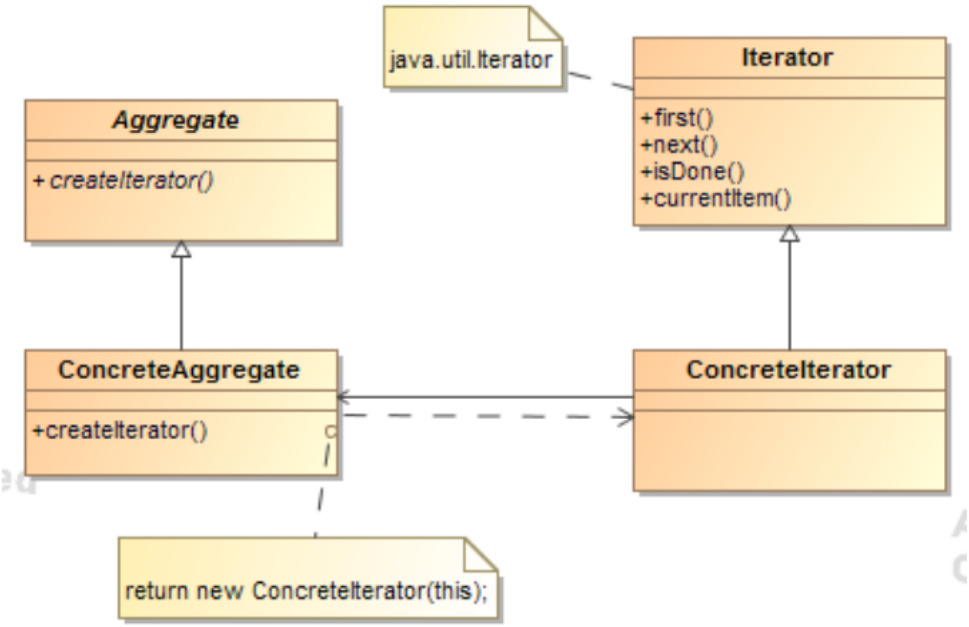

```
public final Enemy createEnemyTemplate(){
    setAlgorithm();
    setName();
    if(!isCivil()){
        setDamage();
    }
    setDistance();
    return this;
}
```

2. Iterator

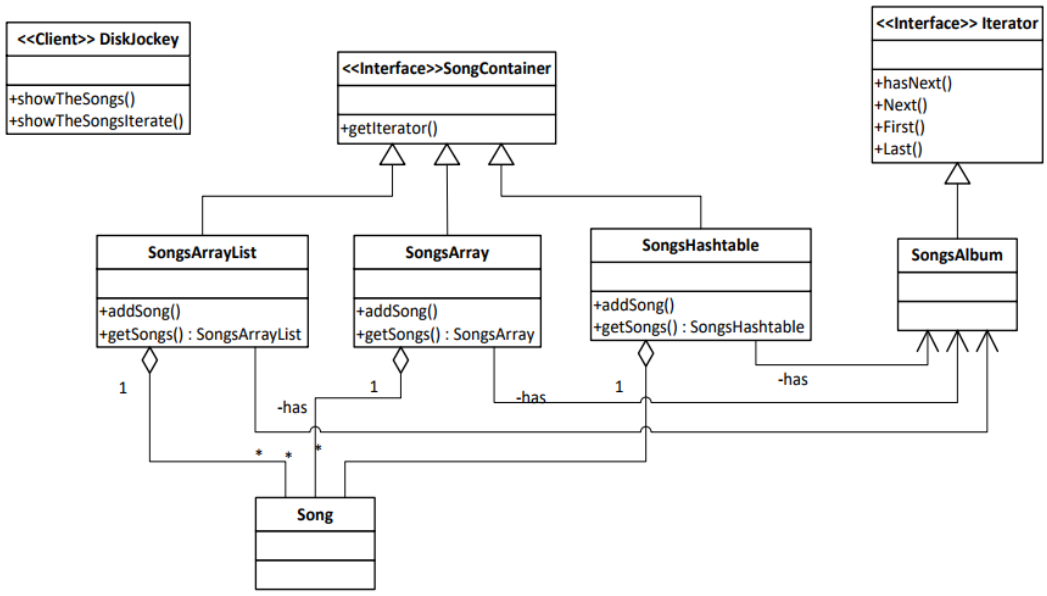
When to use “Iterator”

- if you want to access different collections of objects (trees, binary trees, arrays, ring buffers, hashes, hash maps, array lists, etc.) in the same way
- you might want to access the same collection’s data **in** different ways
- when the collection you’re creating is made up internally of separate subcollections

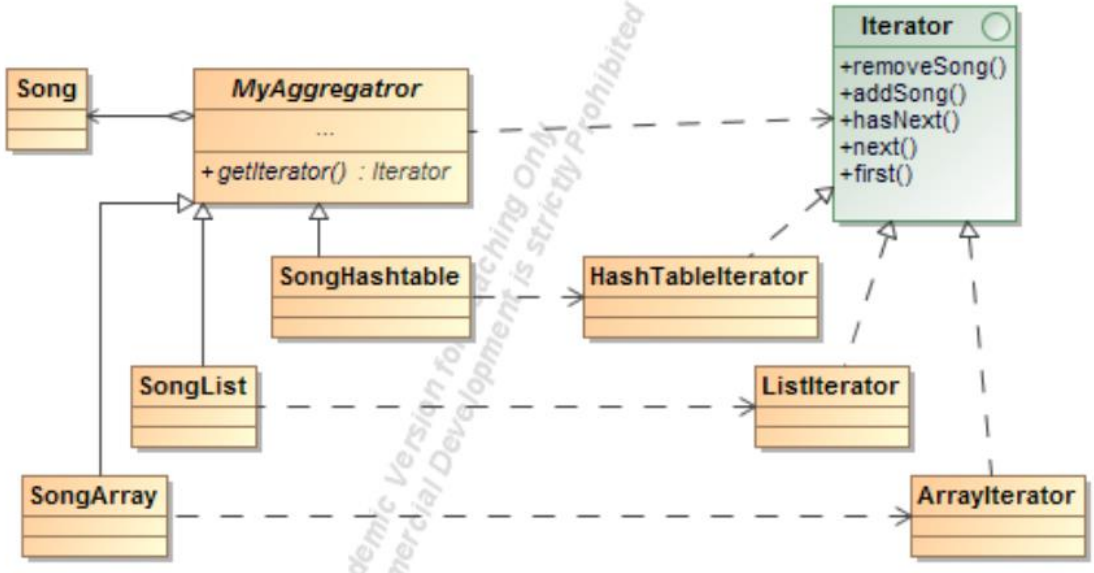
Iterator’s structure (GoF)



Iterator example



Iterator example



Song:

```
public class Song {

    private String name;
    private String band;
    private String releaseYear;

    public Song(String newName, String newBand, String newReleaseYear){
        name = newName;
        band = newBand;
        releaseYear = newReleaseYear;
    }

    public String toString(){
        return "Name: " + name + ", Band: " + band + ", Release year: " + releaseYear;
    }

}
```

SongsArray:

```
import java.util.ArrayList;
import java.util.*; //.Iterator;


public class SongsArray {

    private Song[] songs;
    int index = 0;
    int arraySize = 3;

    public SongsArray(){
        songs = new Song[arraySize];
        songs[index++] = new Song("daina1", "grupe1", "1971");
        songs[index++] = new Song("daina2", "grupe2", "1971");
        songs[index++] = new Song("daina3", "grupe3", "1971");
    }

    public void addSong(Song newSong){
        songs[index++] = newSong;
    }

    public int Size(){
        return index;
    }

    public Song getSong(int i){
        return songs[i];
    }

    public Song[] getSongs(){
        return songs;
    }

    public Iterator getIterator(){
        List list = Arrays.asList(songs);
        return list.iterator();
    }
}
```

SongsHash:

```
import java.util.Hashtable;
import java.util.Iterator;


public class SongsHash {

    private int key = 1;
    private Hashtable<Integer, Song> hash = new Hashtable<Integer, Song>();

    public SongsHash(){
        hash.put(key++, new Song("daina7", "grupe7", "1979"));
        hash.put(key++, new Song("daina8", "grupe8", "1979"));
        hash.put(key++, new Song("daina9", "grupe9", "1979"));
    }

    public void addSong(Song s){
        hash.put(key++, s);
    }

    public void removeSong(Song s){
        hash.remove(s);
    }

    public Hashtable<Integer, Song> getSongs(){
        return hash;
    }

    public Iterator getIterator(){
        return hash.values().iterator();
    }
}
```

SongsList:

```
import java.util.ArrayList;
import java.util.Iterator;


public class SongsList{

    public SongsList(){
            list.add( new Song("daina4", "grupe4", "1975") );
            list.add( new Song("daina5", "grupe5", "1975") );
            list.add( new Song("daina6", "grupe6", "1975") );
    }

    private ArrayList<Song> list = new ArrayList<Song>();

    public void addSong(Song s){
        list.add(s);
    }

    public void removeSong(Song s){
        list.remove(s);
    }

    public ArrayList<Song> getSongs(){
        return list;
    }



    public Iterator getIterator(){
        return list.iterator();
    }

}
```

RadioHits:

```
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;

public class RadioHits {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        SongsArray array = new SongsArray();
        SongsList list = new SongsList();
        SongsHash hash = new SongsHash();

        Song[] songsArray = array.getSongs();
        ArrayList<Song> songsList = list.getSongs();
        Hashtable<Integer, Song> songsHash =  hash.getSongs();

        for(int i = 0; i < array.Size(); i++){
            System.out.println(songsArray[i].toString());
        }

        for(Song s : songsList){
            System.out.println(s.toString());
        }

        for(Song s : songsHash.values()){
            System.out.println(s.toString());
        }

        Iterator i1 = array.getIterator();
        Iterator i2 = list.getIterator();
        Iterator i3 = hash.getIterator();

        showSongs(i1);
        showSongs(i3);
        showSongs(i2);
    }

    public static void showSongs(Iterator i){
        System.out.println("new list ==============");
        while(i.hasNext()){
            Song s = (Song) i.next();
            System.out.println(s.toString());
        }
    }
}
```

TestIterator:

```java
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.TreeSet;
import java.util.Arrays;


public class TestIterator {

    public static void main(String[] args) {
        String[] songArray = new String[] {"first", "second", "third"};
        System.out.println("songArray.length: " + songArray.length);

        ArrayList<String> songList = new ArrayList<String>();
        songList.add("fourth");
        songList.add("fifth");
        songList.add("sixth");
        System.out.println("songList.size: " + songList.size() );

        Hashtable<Integer, String> songHash = new Hashtable<Integer, String>();
        int index = 1;
        songHash.put(index++, "seventh");
        songHash.put(index++, "eighth");
        songHash.put(index++, "ninth");
        System.out.println("songHash: " + songHash.values().size() );

        // Creating and initializing an TreeSet for iteration
        TreeSet<String> treeSetOfSongs = new TreeSet<>();
        treeSetOfSongs.add("ANNA");
        treeSetOfSongs.add("INFY");
        treeSetOfSongs.add("BABA");
        treeSetOfSongs.add("GOOG");
        treeSetOfSongs.add("AMZN");
        System.out.println("TreeSet of songs: " + treeSetOfSongs.size() );

        /*****************************************************************/

    /*********************************************************************/
    System.out.print("\nsongArray: "); // + songArray);
    for(int i = 0; i < songArray.length; i++){
        System.out.print(songArray[i] + "/ ");
    }
    System.out.println();

    System.out.print("\nsongList: "); // + songList);
    for(String item : songList){
        System.out.print(item + "/ ");
    }
    System.out.println();

    System.out.print("\nsongHash: "); // + songHash);
    for(String item : songHash.values()){
        System.out.print(item + "/ ");
    }
    System.out.println();

    System.out.print("\nTreeSet: "); // + treeSetOfSongs);
    for(String song : treeSetOfSongs) {
        System.out.print(song + "/ ");
    }
    System.out.println();


    /*********************************************************************/
    // Obtaining the Iterator
```

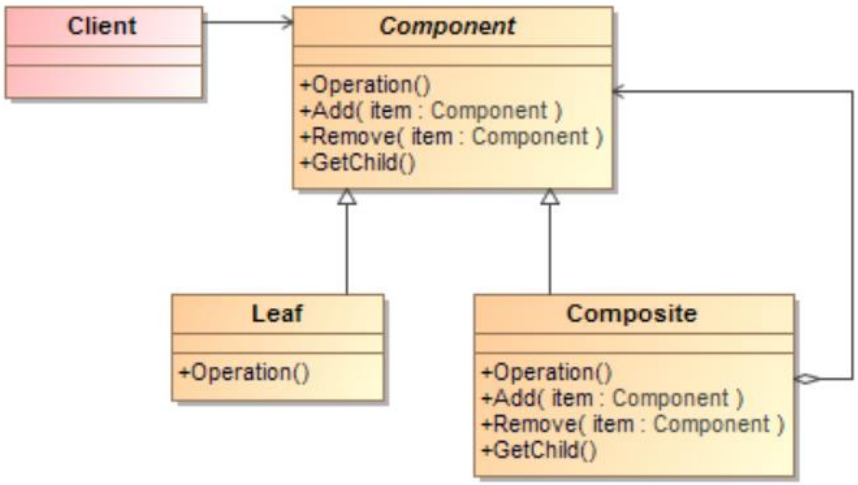
```
        // Obtaining the Iterator

        Iterator<String> iter1 = Arrays.asList(songArray).iterator();
        Iterator<String> iter2 = songList.iterator();
        Iterator<String> iter3 = songHash.values().iterator();
        Iterator<String> iter4 = treeSetOfSongs.iterator();

/*      System.out.println("\n1st album");
        while(iter1.hasNext()){
            System.out.print(iter1.next() + "/ ");
        }

        System.out.println("\n2nd album");
        while(iter2.hasNext()){
            System.out.print(iter2.next() + "/ ");
        }

        System.out.println("\n3rd album");
        while(iter3.hasNext()){
            System.out.print(iter3.next() + "/ ");
        }

        System.out.println("\n4th album");
        while(iter4.hasNext()){
            System.out.print(iter4.next() + "/ ");
        }   */

        printIterator("iter1", iter1);
        printIterator("iter2", iter2);
        printIterator("iter3", iter3);
        printIterator("iter4", iter4);
    }

    public static void printIterator(String name, Iterator<String> iter) {
        System.out.println("\n" + name);
        while(iter.hasNext()){
            System.out.print(iter.next() + "/ ");
        }
        System.out.println();
    }

}
```

3. Composite

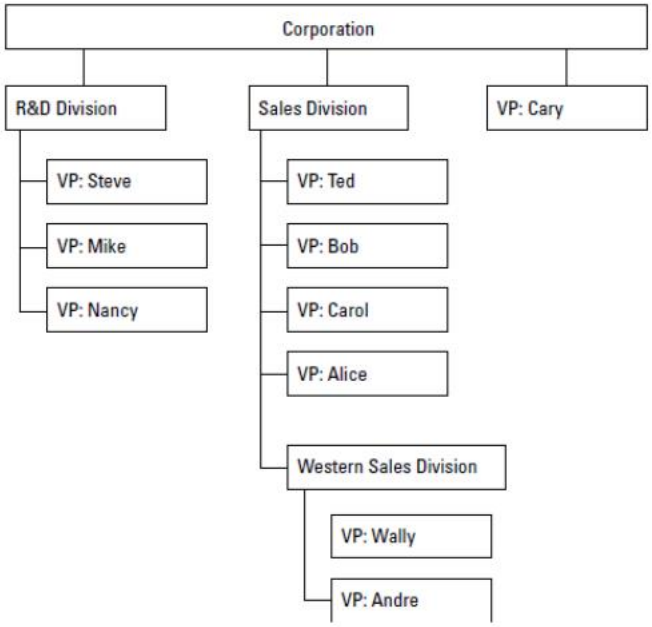
When to use “Composite”, motivation

- When you want clients to be able **to ignore the difference** between compositions of objects and individual objects. Clients will **treat all objects in the composite structure uniformly**.

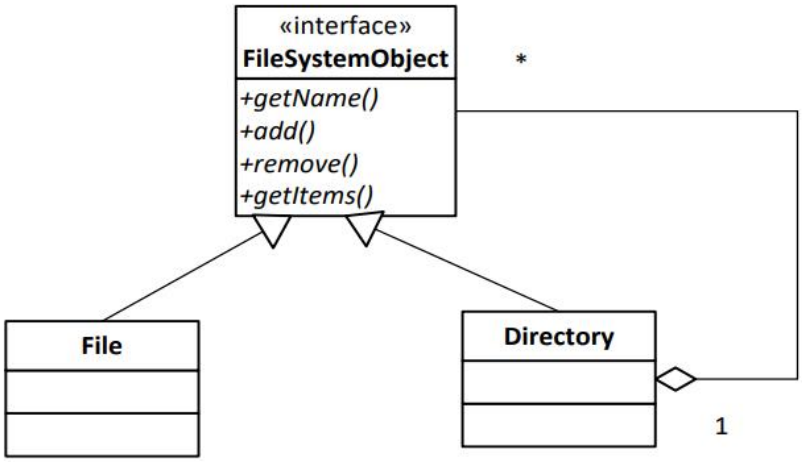
Composite’s structure (GoF)



Composite: application example



Composite : example



FileSystemObject:

```
import java.io.File;
import java.util.ArrayList;
import java.util.Iterator;



public abstract class FileSystemObject {

    private String name;
    protected FileSystemObject parent = null;
    protected ArrayList children;

    public FileSystemObject(String newName) {
        name = newName;
        children = new ArrayList();
    }

    public final String getPath(){
        //throw new UnsupportedOperationException();
        FileSystemObject item = this;
        String path = "";
        while(item.getParent() != null){
            path = item.getParent().getName() + "/" + path;
            item = item.getParent();
        }
        if(path.isEmpty()){
            path = "../";
        }
        path += name;
        System.out.println(path);
        return path;
    }

    public void getDir(){
        throw new UnsupportedOperationException();
    }

    public void getTree(){
        throw new UnsupportedOperationException();
    }
    public String getName(){
        return name;
    }

    public void setParent(FileSystemObject newParent){
        parent = newParent;
    }

    public FileSystemObject getParent(){
        return parent;
    }


    public Iterator getChildren(){
        throw new UnsupportedOperationException();
    }

    public void remove(FileSystemObject newObject){
        throw new UnsupportedOperationException();
    }

    public void add(FileSystemObject newObject){
        throw new UnsupportedOperationException();
    }


}
```

FileObject:

```
public class FileObject extends FileSystemObject{

    public FileObject(String newName) {
        super(newName);
        // TODO Auto-generated constructor stub
    }

}
```

FileSystem:

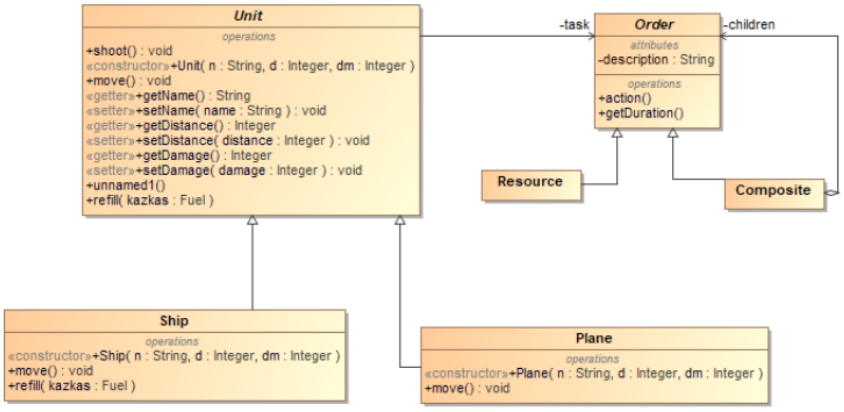
```
import java.util.Iterator;
public class FileSystem {
    public static void main(String[] args){
        FileSystemObject root = new DirectoryObject("root");
        FileSystemObject file1 = new FileObject("file1.txt");
        FileSystemObject file2 = new FileObject("file2.txt");

        root.add(file1);        root.add(file2);
        root.getPath();

        FileSystemObject dir1 = new DirectoryObject("dir1");
        root.add(dir1);
        FileSystemObject file3 = new FileObject("file3.txt");
        FileSystemObject file4 = new FileObject("file4.txt");
        dir1.add(file3);        dir1.add(file4);

        FileSystemObject dir2 = new DirectoryObject("dir2");
        dir1.add(dir2);
        FileSystemObject file5 = new FileObject("file5.txt");
        FileSystemObject file6 = new FileObject("file6.txt");
        dir2.add(file5);        dir2.add(file6);

        System.out.println("--- dir ---");
        root.getDir();
        System.out.println("--- tree ---");
        root.getTree();

        System.out.println("=== Iteration ===");
        Iterator ch = root.getChildren();
        while(ch.hasNext()){
            FileSystemObject obj = (FileSystemObject) ch.next();
            System.out.println(obj.getName());
            if (obj instanceof DirectoryObject) {
                DirectoryObject tempDir = (DirectoryObject) obj;
                //tempDir.getDir();
                tempDir.getTree();
            }
        }
        //file6.getPath();
        //file6.getDir();
    }
}
```

DirectoryObject:

```
import java.util.Iterator;


public class DirectoryObject extends FileSystemObject{

    public DirectoryObject(String newName) {
        super(newName);
    }

    public void add(FileSystemObject item){
        this.children.add(item);
        item.setParent(this);
    }

    public void remove(FileSystemObject item){
        this.children.remove(item);
    }

    public Iterator getChildren(){
        return this.children.iterator();
    }

    public void getDir(){
        Iterator iterator = getChildren();
        while( iterator.hasNext() ){
            FileSystemObject item = (FileSystemObject) iterator.next();
            item.getPath();
        }
    }

    public void getTree(){
        Iterator iterator = getChildren();
        while(iterator.hasNext()){
            FileSystemObject item = (FileSystemObject) iterator.next();
            item.getPath();
            if (item instanceof DirectoryObject) {
                DirectoryObject subdir = (DirectoryObject) item;
                subdir.getTree();
            }
        }
    }

}
```

Example to code

- Allow Unit to manage sequences of Tasks and Deliverable Results (ProjectItem\



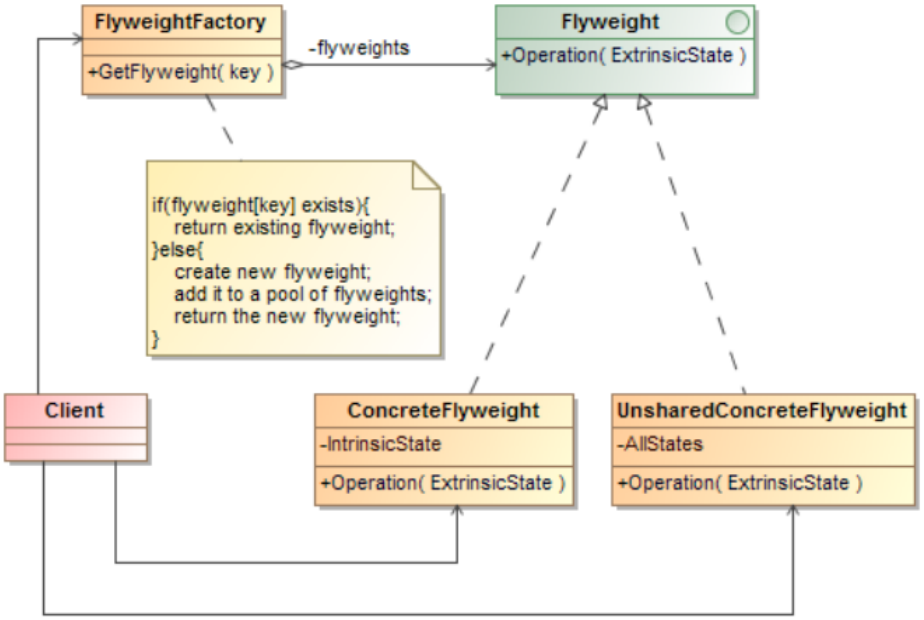
4.

Flyweight

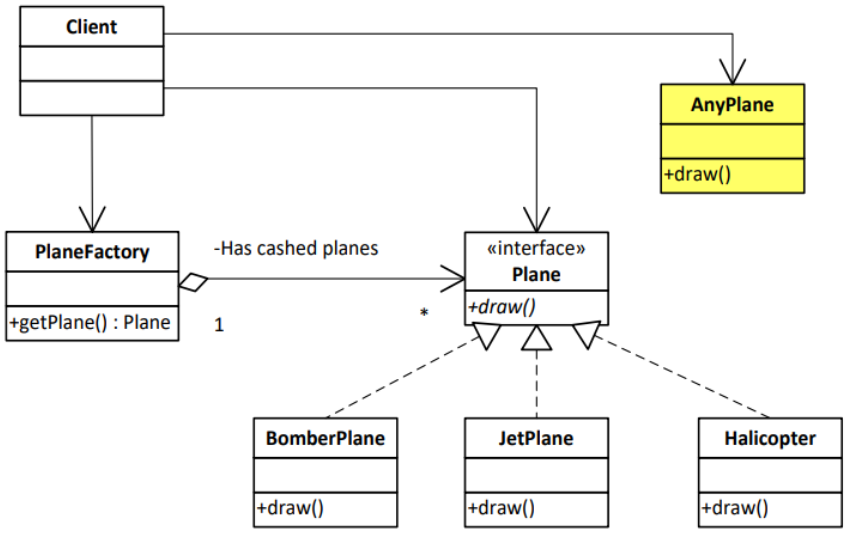
- Type: Structural
- Also Known As:
- Intent: “Use sharing to support large numbers of fine-grained objects efficiently”

- The key idea is to minimize objects quantity by sharing them.
- Objects have part of their internal state in common where the other part of state can vary

Flyweight's structure (GoF)



Flyweight: example



PlaneFactory:

```java
import java.util.Hashtable;

public class PlaneFactory {

    private final static Hashtable<String, Plane> hash = new Hashtable<String, Plane>();

    public static Plane getPlane(String planeType){

        Plane p = hash.get(planeType);

        if(p == null){
            if(planeType.equals("src/plane.jpg")){
                p = new JetPlane(planeType);
            }else
            if(planeType.equals("src/bomber.jpg")){
                p = new BomberPlane(planeType);
            }else
            if(planeType.equals("src/helicopter.jpg")){
                p = new Helicopter(planeType);
            }

            hash.put(planeType, p);
        }

        return p;

    }

}
```

Plane:

```java
import java.awt.Graphics;


public interface Plane {

    public void move(Graphics g, int positionX, int positionY);
}
```

JetPlane:

```java
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

import javax.imageio.ImageIO;

public class JetPlane implements Plane{

    BufferedImage image;
    Graphics g;

    public JetPlane(String fileName){
        //BufferedImage image;
        try {
            //String fileName =  getRandomType();  //"src/plane.jpg";
            File imageFile = new File(fileName);
            //System.out.println(imageFile.getAbsolutePath() );
            image = ImageIO.read(imageFile );


        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }

    }

    public void move(Graphics g, int positionX, int positionY){
        g.drawImage(image, positionX, positionY, null);
    }

}
```

Helicopter:

```java
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

import javax.imageio.ImageIO;

public class Helicopter implements Plane{

    BufferedImage image;
    Graphics g;

    public Helicopter(String fileName){
        //BufferedImage image;
        try {
            //String fileName =  getRandomType();  //"src/plane.jpg";
            File imageFile = new File(fileName);
            //System.out.println(imageFile.getAbsolutePath() );
            image = ImageIO.read(imageFile );


        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }

    }

    public void move(Graphics g, int positionX, int positionY){
        g.drawImage(image, positionX, positionY, null);
    }

}
```

BomberPlane:

```java
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

import javax.imageio.ImageIO;

public class BomberPlane implements Plane{

    BufferedImage image;
    Graphics g;

    public BomberPlane(String fileName){
        //BufferedImage image;
        try {
            //String fileName =  getRandomType();  //"src/plane.jpg";
            File imageFile = new File(fileName);
            //System.out.println(imageFile.getAbsolutePath() );
            image = ImageIO.read(imageFile );


        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }

    public void move(Graphics g, int positionX, int positionY){
        g.drawImage(image, positionX, positionY, null);
    }



}
```

AnyPlane:

```java
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;


public class AnyPlane {

    BufferedImage image;

    public AnyPlane(String fileName){
        BufferedImage image;
        try {
            //String fileName =  getRandomType();  //"src/plane.jpg";
            File imageFile = new File(fileName);
            //System.out.println(imageFile.getAbsolutePath() );
            image = ImageIO.read(imageFile );


        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }

    public void draw(Graphics g, int positionX, int positionY){
        g.drawImage(image, positionX, positionY, null);
    }

}
```

FlyweightExample:

```java
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;
import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class FlyweightExample extends JFrame implements ActionListener {
    JPanel drawingPanel;
    int windowWidth = 700;
    int windowHeight = 500;
    final int counter = 5000;
    String[] planeTypes = {"src/plane.jpg", "src/bomber.jpg", "src/helicopter.jpg"};

    public FlyweightExample(){
        setSize(windowWidth, windowHeight);
        setTitle("Flyweight design pattern");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        drawingPanel = new JPanel();
        drawingPanel.setBackground(Color.WHITE);
        add(drawingPanel, BorderLayout.CENTER);
        JButton button = new JButton("Press on me");
        button.addActionListener(this);
        add(button, BorderLayout.SOUTH);
    }
    public static void main(String[] args){
        FlyweightExample w = new FlyweightExample();
        w.setVisible(true);
    }
    public int getRandomX(){
        return (int)(Math.random()*windowWidth);
    }
    public int getRandomY(){
        return (int)(Math.random()*windowHeight);
    }
```

```
public String getRandomType(){
    Random random = new Random();
    int index = random.nextInt(planeTypes.length);
    return planeTypes[index];
}

@Override
public void actionPerformed(ActionEvent e) {
    //drawSinglePlane();

    drawManyPlanes();
    Runtime.getRuntime().gc(); //recycling unused objects
    drawManyClassPlanes();
    Runtime.getRuntime().gc();
    drawManyFactoryPlanes();
}

public void drawSinglePlane(){
    Graphics g = drawingPanel.getGraphics();
    BufferedImage image;
    try {
        String fileName =  getRandomType();  //"src/plane.jpg";
        File imageFile = new File(fileName);
        //System.out.println(imageFile.getAbsolutePath() );
        image = ImageIO.read(imageFile );
        g.drawImage(image, getRandomX(), getRandomY(), null);
    } catch (IOException e1) {
        e1.printStackTrace();
    }
}

public void drawManyPlanes(){
    long startTime = System.currentTimeMillis();
    long beforeUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
    for(int i = 0; i < counter; i++){
        drawSinglePlane();
    }
    long endTime = System.currentTimeMillis();
    long afterUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
    long KB = (afterUsedMem - beforeUsedMem) >> 10;
    long MB = KB >> 10;
    System.out.println("Simple plane draw took: " + (endTime - startTime) + "ms");
```

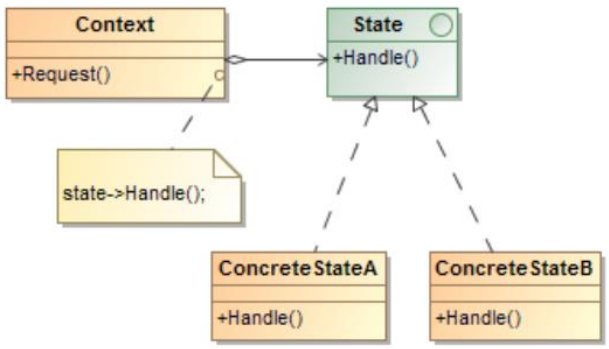
```
    System.out.println("Simple plane memory utilized: " + (afterUsedMem - beforeUsedMem) + " bytes | " + MB + " MB");

}

public void drawManyClassPlanes(){
    long startTime = System.currentTimeMillis();
    long beforeUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();

    Graphics g = drawingPanel.getGraphics();
    for(int i = 0; i < counter; i++){
        AnyPlane p = new AnyPlane( getRandomType() );
        p.draw(g, getRandomX(), getRandomY());
    }
    long endTime = System.currentTimeMillis();
    long afterUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
    long KB = (afterUsedMem - beforeUsedMem) >> 10;
    long MB = KB >> 10;
    System.out.println("Class planes draw took: " + (endTime - startTime) + "ms");
    System.out.println("Class plane memory utilized: " + (afterUsedMem - beforeUsedMem) + " bytes | " + MB + " MB");
}

public void drawManyFactoryPlanes(){
    long startTime = System.currentTimeMillis();
    long beforeUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();

    Graphics g = drawingPanel.getGraphics();
    for(int i = 0; i < counter; i++){
        Plane p = PlaneFactory.getPlane( getRandomType() );
        p.move(g, getRandomX(), getRandomY());
    }
    long endTime = System.currentTimeMillis();
    long afterUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
    long KB = (afterUsedMem - beforeUsedMem) >> 10;
    long MB = KB >> 10;
    System.out.println("Factory planes took: " + (endTime - startTime) + "ms");
    System.out.println("Factory plane memory utilized: " + (afterUsedMem - beforeUsedMem) + " bytes | " + MB + " MB");
}
```

5. State

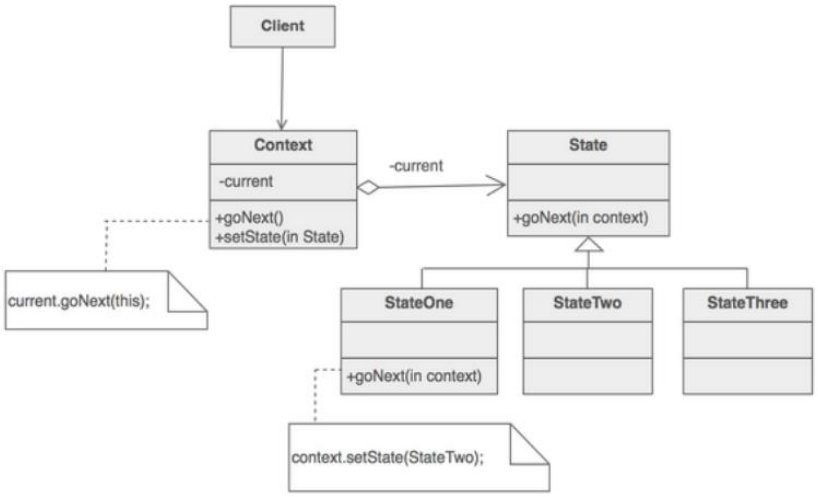
State" when to use

- when an Object change it's behavior based on it's internal state.
- an object keeps track of its internal state, and can change its behavior based on that state
- object's behavior is a function of its state, and it must **change its behavior at run-time depending on that state**.

State: structure (GoF)



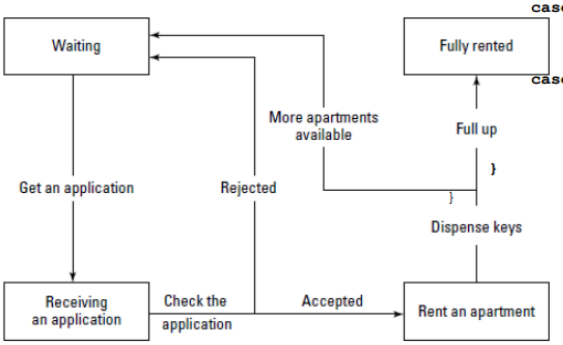
State: demystified



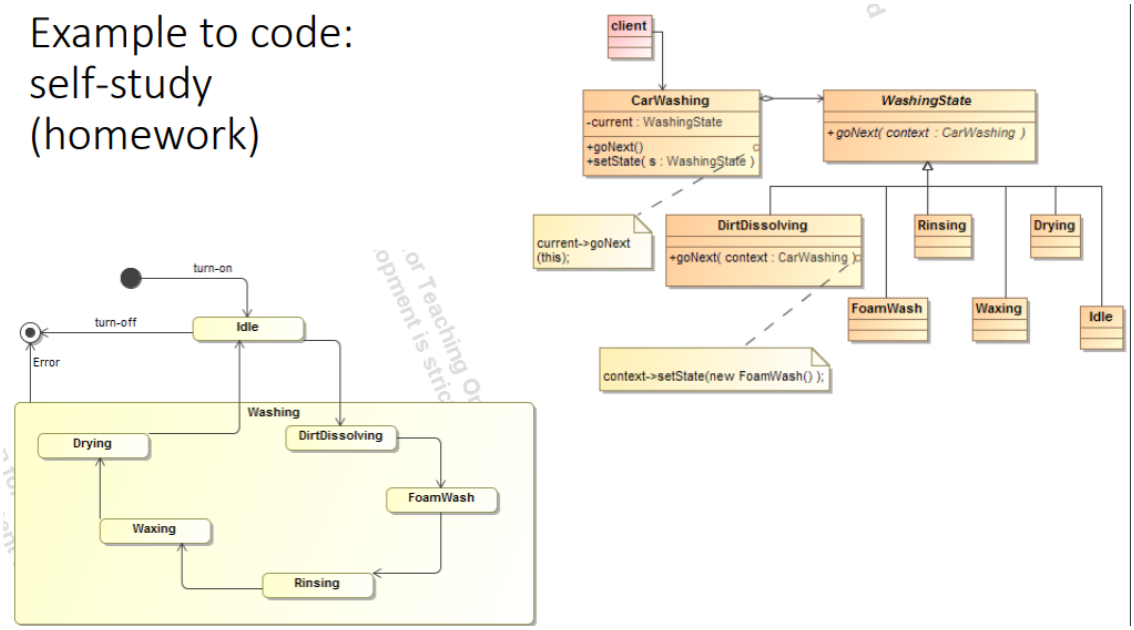
- Strategies are passed to the context object as parameters,
- States are created by the context object itself

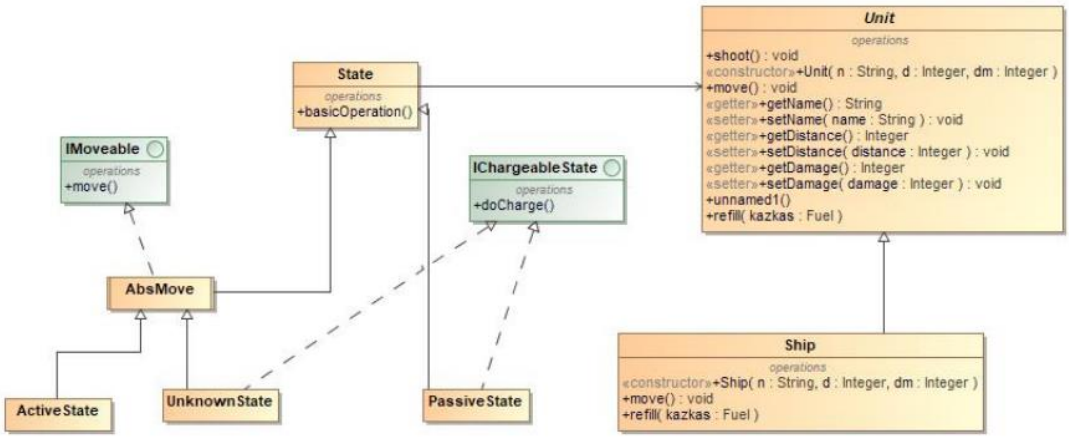
State: Implementation based on StateChart diagram

```
public void getApplication(){
    switch (state)
    {
        case FULLY_RENTED:
            System.out.println("Sorry, we're fully rented.");
            break;
        case WAITING:
            state = GOT_APPLICATION;
            System.out.println("Thanks for the application.");
            break;
        case GOT_APPLICATION:
            System.out.println("We already got your application.");
            break;
        case APARTMENT_RENTED:
            System.out.println("Hang on, we're renting you an
            apartment.");
            break;
```



Example to code: self-study (homework)





TVState:

```
public interface TVState{

    public void doAction();
}
```

TVStopState:

```
public class TVStopState implements TVState {

    @Override
    public void doAction() {
        // TODO Auto-generated method stub
        System.out.println("TV is turning OFF");
    }


}
```

TVStartState:

```
public class TVStartState implements TVState {

    @Override
    public void doAction() {
        // TODO Auto-generated method stub
        System.out.println("TV is turning ON");
    }


}
```

TVContext:

```
public class TVContext implements TVState{

    TVState state;

    public void setState(TVState newState){
        state = newState;
    }

    @Override
    public void doAction() {
        // TODO Auto-generated method stub
        state.doAction();
    }


}
```

TVRemote:

```
public class TVRemote {
    //with State pattern
    public static void main(String[] args) {

        TVContext tv = new TVContext();

        tv.setState(new TVStartState() );
        tv.doAction();

        tv.setState(new TVStopState());
        tv.doAction();

    }

}
```

TVRemoteBasic:

```
public class TVRemoteBasic {
    //without State pattern

    String state = "";

    public void setState(String newState){
        state = newState;
    }

    public void doAction(){
        if(state.equalsIgnoreCase("on")){
            System.out.println("turning it OFF");
        }else
        if(state.equalsIgnoreCase("off")){
            System.out.println("Turning it ON");
        }
    }

    public static void main(String[] args){

        TVRemoteBasic tvset = new TVRemoteBasic();

        tvset.setState("On");
        tvset.doAction();

        tvset.setState("Off");
        tvset.doAction();

    }

}
```

6. Proxy

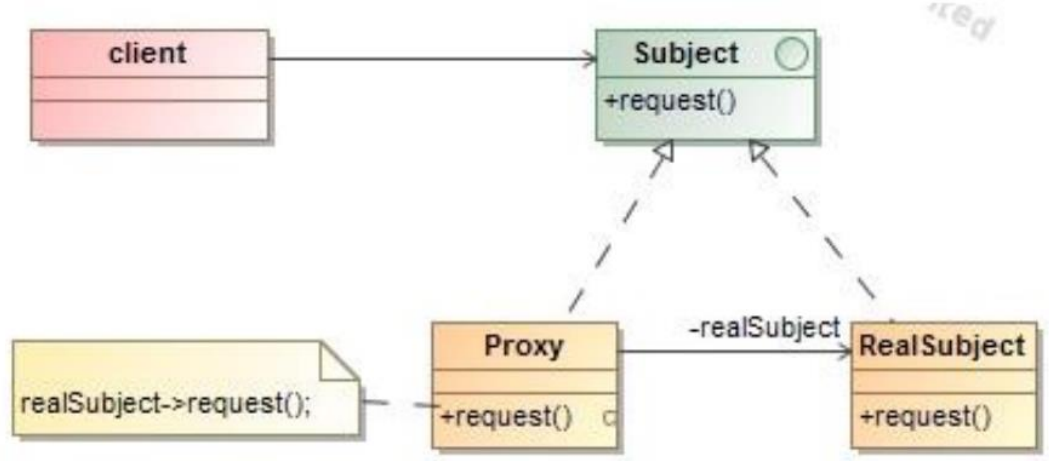
When to use “Proxy”, motivation

To provide controlled access to Real Subject and it’s functionality:

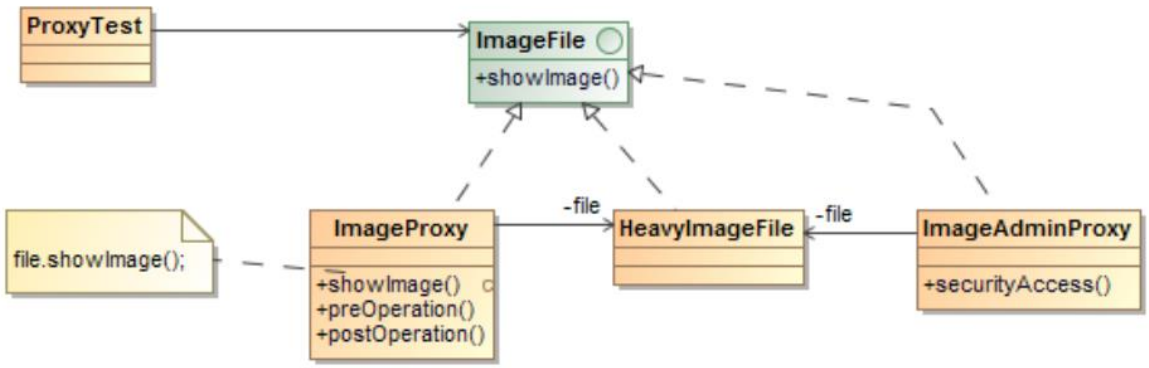
(1) to defer the full cost of its creation and initialization until we actually need to use it (controlling when a costly object needs to be instantiated and initialized)

(2) giving different access rights to an object.

Proxy’s structure (GoF)



Proxy : from last year



ImageFile:

```
public interface ImageFile {

    public void showImage();

}
```

HeavyImageFile:

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;


public class HeavyImageFile implements ImageFile{

    String path;

    public HeavyImageFile(String newPath){
        path = newPath;
        try {
            File imageFile = new File(path);
            BufferedImage image = ImageIO.read(imageFile );
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }

    @Override
    public void showImage() {

        System.out.println("Image " + path + "loaded");
    }

}
```

ImageAdminProxy:

```
public class ImageAdminProxy implements ImageFile{

    ImageFile file; //heavy image file
    //HeavyImageFile file;

    String path;

    boolean isAdmin = false;

    public ImageAdminProxy(String user, String pwd, String newPath){
        if(user.equals("admin")){
            isAdmin = true;
        }
        path = newPath;
    }

    @Override
    public void showImage() {
        if(isAdmin){
            file = new HeavyImageFile(path);
            file.showImage();
        }else{
            System.out.println("Access denied!");
        }

    }

}
```

ImageProxy:

```
public class ImageProxy implements ImageFile{

    ImageFile file; //heavy image file
    //HeavyImageFile file;

    String path;
    ImageFile parentProxy;

    public ImageProxy(String newPath){
        path = newPath;
    }

    public ImageProxy(ImageFile anotherProxy){
        parentProxy = anotherProxy;
    }

    @Override
    public void showImage() {
        if(parentProxy == null) {
            file = new HeavyImageFile(path);
            file.showImage();
        }else {
            parentProxy.showImage();
        }
    }
}
```

ProxyTest:

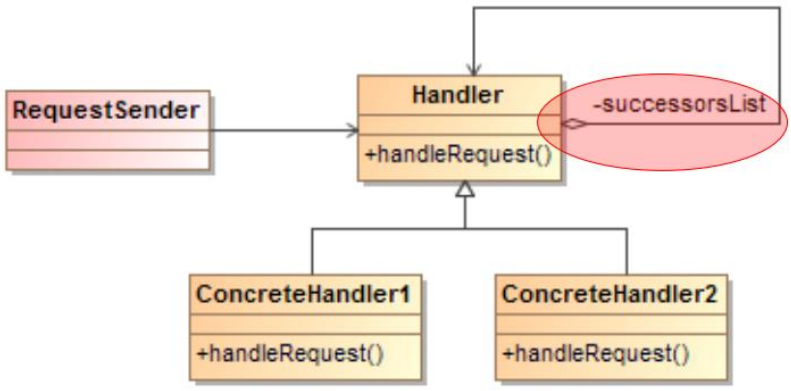
```java
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;


public class ProxyTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        long startP = System.currentTimeMillis();
        String path = "src/heavyImage.png";

        try {
            File imageFile = new File(path);
            BufferedImage image = ImageIO.read(imageFile );
        } catch (IOException e1) {
            e1.printStackTrace();
        }

        System.out.println("showImage");

        long stopP = System.currentTimeMillis();
        System.out.println("One image loaded in: " + (stopP - startP) + "ms");

        long startProxy = System.currentTimeMillis();
        ImageProxy p1 = new ImageProxy(path);
        ImageProxy p2 = new ImageProxy(path);
        ImageProxy p3 = new ImageProxy(path);
        p3.showImage();
        long stopProxy = System.currentTimeMillis();
        System.out.println("Proxy loaded in " + (stopProxy - startProxy) + "ms");


        long startReal = System.currentTimeMillis();
        HeavyImageFile h1 = new HeavyImageFile(path);
        HeavyImageFile h2 = new HeavyImageFile(path);
        HeavyImageFile h3 = new HeavyImageFile(path);
        h2.showImage();
        long stopReal = System.currentTimeMillis();
        System.out.println("Real loaded in " + (stopReal - startReal) + "ms");

        ImageAdminProxy guest = new ImageAdminProxy("guest", "123", path);
        guest.showImage();

        ImageAdminProxy admin = new ImageAdminProxy("admin", "123", path);
        admin.showImage();


        System.out.println("=== bandom kelis proxy sujungti ===");
        long proxiesStart = System.currentTimeMillis();
        ImageProxy proxiesAdmin = new ImageProxy(admin);
        proxiesAdmin.showImage();
        ImageProxy proxiesGuest = new ImageProxy(guest);
        proxiesGuest.showImage();
        long proxiesStop = System.currentTimeMillis();
        System.out.println("Real loaded in " + (proxiesStop - proxiesStart) + "ms");

    }

}
```

7. Chain of responsibility

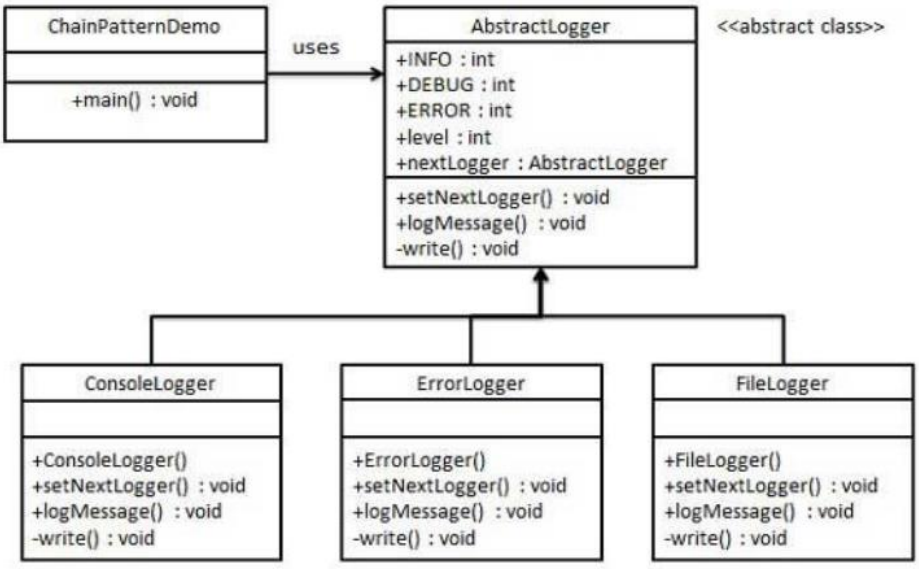
Chain of responsibility

- Type: behavioral
- Also Known As:
- Intent: “Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it”.

Chain of responsibility’s structure (GoF)

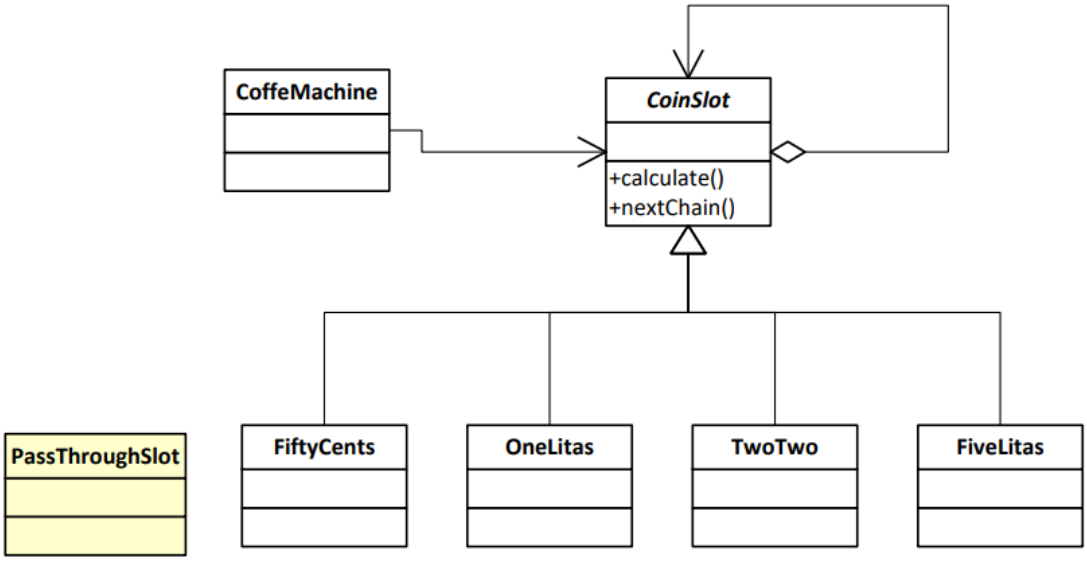


Chain of responsibility: application example



Chain of responsibility: code example

- The vending machine coin slot: rather than having a slot for each type of coin, the machine has only one slot for all of them.



CoinsSlot:

```java
public abstract class CoinsSlot {

    protected static double coinsSum = 0;
    protected CoinsSlot next;

    public abstract void calculate(double coins);

    public abstract void setNextChain(CoinsSlot next);

}
```

FiftyCentsSlot:

```java
public class FiftyCentsSlot extends CoinsSlot {

    private double value = 0.50;


    @Override
    public void calculate(double coins) {

        if(coins == value){
            this.coinsSum += value;
            System.out.println("Suma: " + coinsSum);
        }else{
            next.calculate(coins);
        }

    }

    @Override
    public void setNextChain(CoinsSlot next) {
        // TODO Auto-generated method stub
        this.next = next;
    }

}
```

OneLitasSlot:

```java
public class OneLitasSlot extends CoinsSlot {

    private double value = 1;

    @Override
    public void calculate(double coins) {

        if(coins == value){
            this.coinsSum += value;
            System.out.println("Suma: " + coinsSum);
        }else{
            next.calculate(coins);
        }

    }

    @Override
    public void setNextChain(CoinsSlot next) {
        // TODO Auto-generated method stub
        this.next = next;
    }

}
```

TwoLitasSlot:

```java
public class TwoLitasSlot extends CoinsSlot{

    private double value = 2;

    @Override
    public void calculate(double coins) {

        //next.calculate(coins);
        if(coins == value){
            this.coinsSum += value;
            System.out.println("Suma: " + coinsSum);
        }else{
            next.calculate(coins);
        }

    }

    @Override
    public void setNextChain(CoinsSlot next) {
        // TODO Auto-generated method stub
        this.next = next;
    }

}
```

FiveLitasSlot:

```java
public class FiveLitasSlot extends CoinsSlot{
    private double value = 5;


    @Override
    public void calculate(double coins) {

        if(coins == value){
            this.coinsSum += value;
            System.out.println("Suma: " + coinsSum);
        }else{
            next.calculate(coins);
        }

    }

    @Override
    public void setNextChain(CoinsSlot next) {
        // TODO Auto-generated method stub
        this.next = next;
    }
}
```

PassThroughSlot:

```java
public class PassThroughSlot extends CoinsSlot{

    @Override
    public void calculate(double coins) {
        // TODO Auto-generated method stub
        System.out.println("Netinkama moneta");
    }

    @Override
    public void setNextChain(CoinsSlot next) {
        // TODO Auto-generated method stub
        this.next = next;
    }

}
```

CoffeeMachine (Main):

```java
public class CoffeeMachine {

    public static void main(String[] args) {
        CoinsSlot s1 = new FiftyCentsSlot();
        CoinsSlot s2 = new OneLitasSlot();
        CoinsSlot s3 = new TwoLitasSlot();
        CoinsSlot s4 = new FiveLitasSlot();
        CoinsSlot s5 = new PassThroughSlot();

        s4.setNextChain(s3);
        s3.setNextChain(s2);
        s2.setNextChain(s1);
        s1.setNextChain(s5);
        s5.setNextChain(s1);

        CoinsSlot mainSlot = s4;

        mainSlot.calculate(0.5);
        mainSlot.calculate(1);
        mainSlot.calculate(444);
        mainSlot.calculate(0.5);
        mainSlot.calculate(5);
        mainSlot.calculate(0.2);


    }

}
```

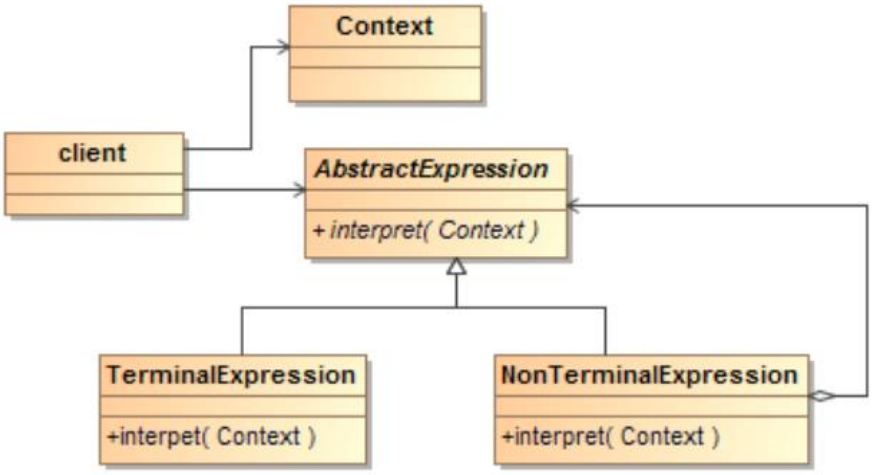
8. Interpreter

Interpreter: when to use

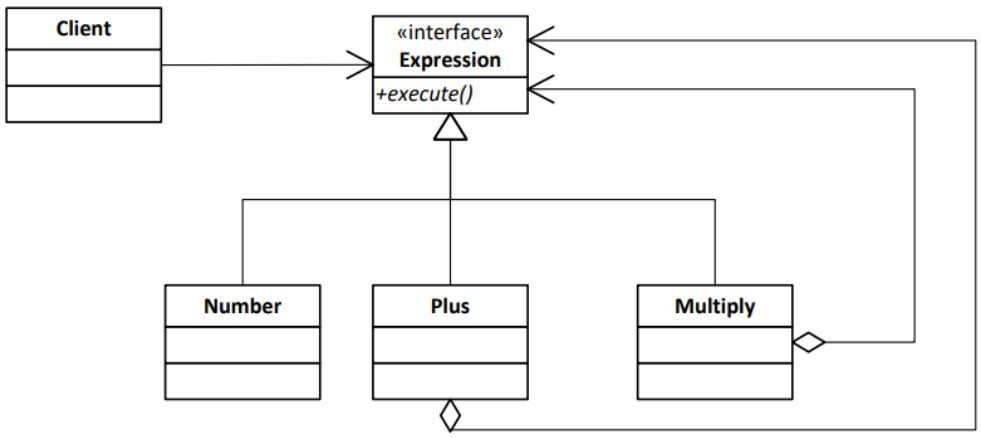
- this is one of those patterns that doesn't see a lot of everyday use because creating your own language is not something many people do ☺

- Each item might be represented as an object with an interpret method to translate your new language into something you can run in Java

Interpreter: structure



Interpreter: code sample



Expression:

```
public interface Expression {

    public int execute();

}
```

NumberExpression:

```
public class NumberExpression implements Expression{

    private int value;

    public NumberExpression(int newValue) {
        value = newValue;
    }

    @Override
    public int execute() {
        return value;
    }

}
```

PlusExpression:

```
public class PlusExpression implements Expression{

    Expression left, right;

    public PlusExpression(Expression exp1, Expression exp2) {
        left = exp1;
        right = exp2;
    }

    @Override
    public int execute() {
        // TODO Auto-generated method stub
        return left.execute() + right.execute();
    }


}
```

MultiplyExpression:

```java
public class MultiplyExpression implements Expression{

Expression left, right;

    public MultiplyExpression(Expression exp1, Expression exp2) {
        left = exp1;
        right = exp2;
    }

    @Override
    public int execute() {
        // TODO Auto-generated method stub
        return left.execute() * right.execute();
    }

}
```

ClientApp (main):

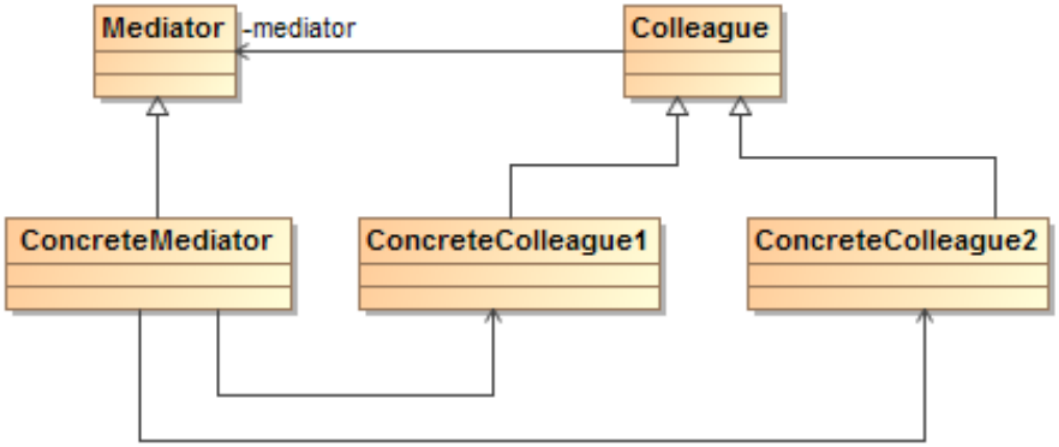
```java
import java.util.StringTokenizer;


public class ClientApp {

    public static void main(String[] args) {

        //StringTokenizer str = new StringTokenizer("4 + 5 * 2");

        Expression e1 = new NumberExpression(4);
        Expression e2 = new NumberExpression(5);
        Expression e3 = new NumberExpression(2);
        // 4 + 5
        Expression e4 = new PlusExpression(e1, e2);
        // 9 * 2
        Expression e5 = new MultiplyExpression(e4, e3);


        System.out.println("4 + 5 * 2 = " + e5.execute());
    }

}
```

9. Mediator

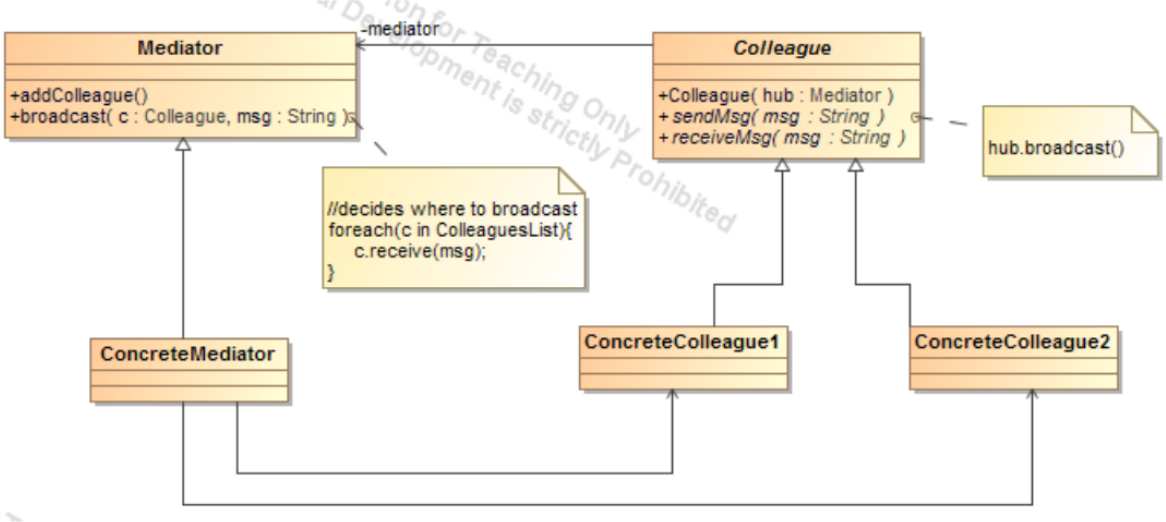
When to use “Mediator”, motivation

- You have many interdependent objects
- Every object ends up knowing about every other.
- Objects communicate in well-defined but complex ways
- Reusing an object is difficult because it refers to and communicates with many other objects

Mediator's structure (GoF)

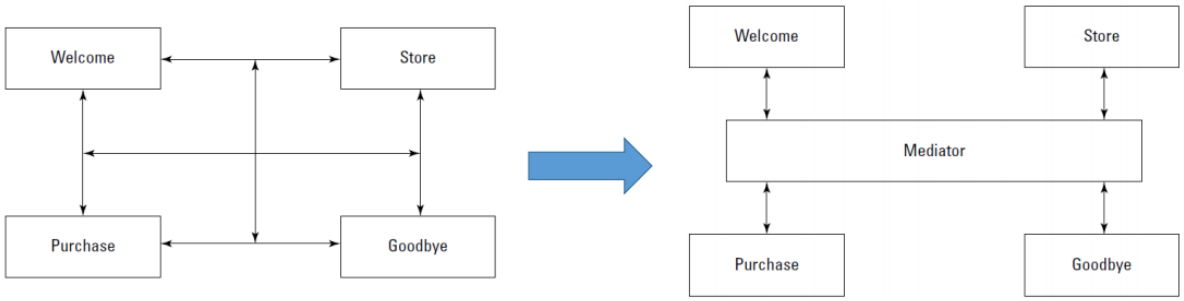


Mediator's structure extended

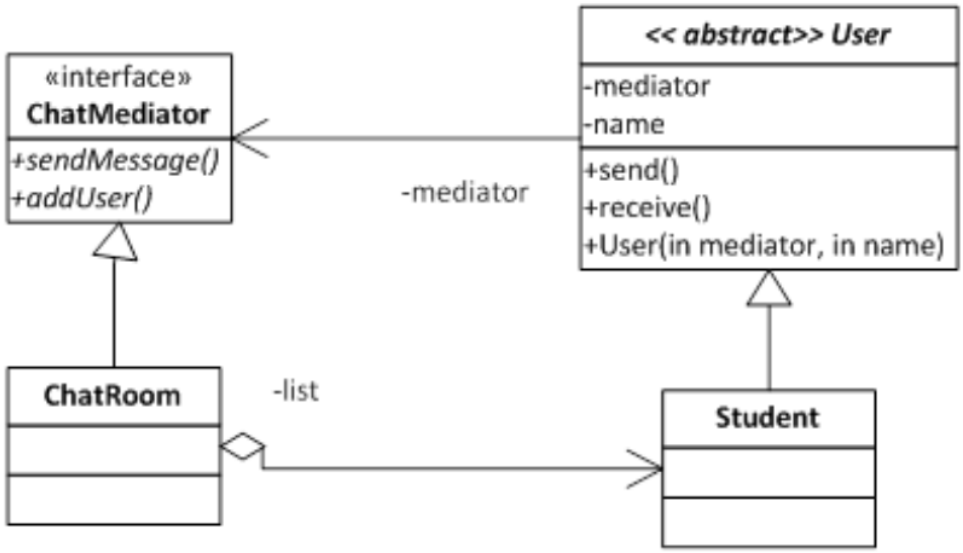


Mediator: implementation

- The Mediator design pattern brings a central processing hub into the picture.
- All the pages now have to **interact with the mediator only**. When a page's internal state changes, it just reports that state change to the mediator, which **decides where to transfer control next**, acting something like a controller in Model/View/Controller architecture.



Mediator : sample code



Mediator:

```
public interface Mediator {

    public void addUser(User user);

    public void broadcastMessage(User sender, String msg);

}
```

MediatorImpl:

```java
import java.util.ArrayList;


public class MediatorImpl implements Mediator{

    ArrayList<User> list = new ArrayList<User>();

    @Override
    public void addUser(User user) {
        list.add(user);

    }

    @Override
    public void broadcastMessage(User sender, String msg) {
        for(User user : list){
            if(user != sender){
                user.receiveMessage(msg);
            }
        }

    }

}
```

User:

```java
public abstract class User {

    protected Mediator m;
    protected String name;

    public User(Mediator mediator, String newName){
        m = mediator;
        name = newName;
    }

    public abstract void sendMessage(String msg);

    public abstract void receiveMessage(String msg);

}
```

Student:

```java
public class Student extends User{

    public Student(Mediator mediator, String newName) {
        super(mediator, newName);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void sendMessage(String msg) {
        System.out.println(name + " sent: " + msg);
        m.broadcastMessage(this, msg);

    }

    @Override
    public void receiveMessage(String msg) {
        System.out.println(name + " received: " + msg);

    }

}
```

ChatRoom (main):

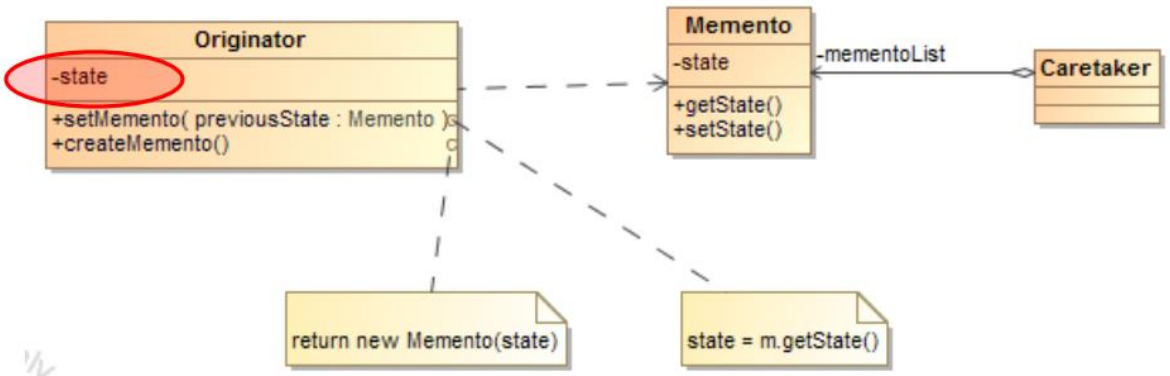
```java
public class ChatRoom {

    public static void main(String[] args){

        Mediator mediator = new MediatorImpl();

        User u1 = new Student(mediator, "Jonas");
        User u2 = new Student(mediator, "Paulius");
        User u3 = new Student(mediator, "Bronius");
        User u4 = new Student(mediator, "Sigitas");

        mediator.addUser(u1);
        mediator.addUser(u2);
        mediator.addUser(u3);
        mediator.addUser(u4);


        u1.sendMessage("labas rytas");
        u2.sendMessage("kas bus per koli?");
    }

}
```

10. Memento

Memento: when to use

- When implementing checkpoints and undo mechanisms that let users back out of tentative operations or recover from errors.
- You must save state information somewhere so that you can restore objects to their previous states

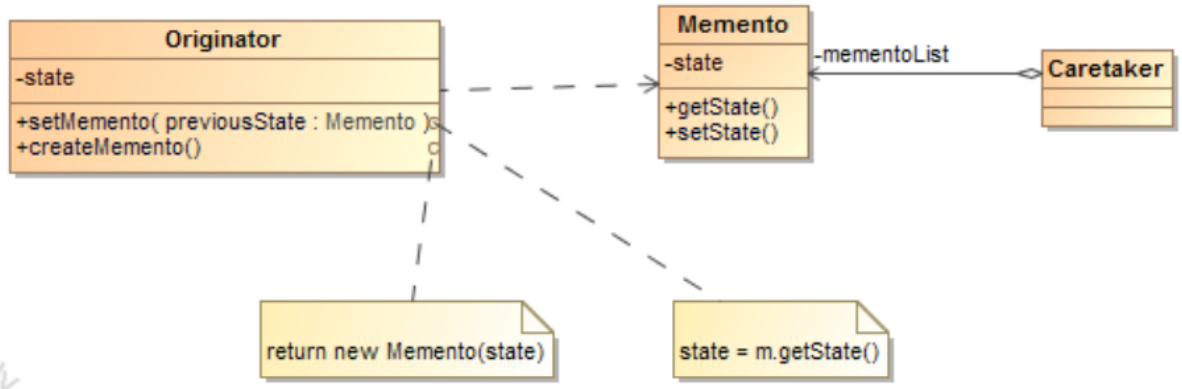
Memento's structure (GoF)



- Encapsulation – Originator's State (private) is closed from others
- Caretaker can get Originator's state only as a Memento object
- Caretaker manages states of Originator as a List<Memento> and restores state whenevet it is needed
- Originator restores it's state from Memento and when Memento is received

Memento: sample code

- ???



Originator:

```
public class Originator {

    String state;

    public Originator(String newState){
        System.out.println("Org created: " + newState);
        state = newState;
    }

    public String getState(){
        return state;
    }

    public void setState(String newState){
        System.out.println("set state: " + newState);
        state = newState;
    }

    public Memento saveState(){
        System.out.println("seved state: " + state);
        return new Memento(state);
    }

    public void restoreState(Memento restoreState){
        //System.out.println("restore state: " + restoreState.getState());
        //state = restoreState.getState();
        restoreState.getState(this);
        System.out.println("restore state: " + this.getState());
    }


}
```

Memento:

```
public class Memento {

    String state;

    public Memento(String newState){
        state = newState;
    }

    //public String getState(){
    //  return state;
    //}

    public void getState(Originator org) {
        org.setState(this.state);
    }

}
```

Caretaker:

```
public class Caretaker {

    List<Memento> statesList;

    public Caretaker(){
        statesList = new ArrayList<Memento>();
    }

    public void add(Memento state){
        //statesList = statesList.subList(0, currentState);
        statesList.add(state);
    }

    public Memento get(int index){
        Memento restoreState = statesList.get(index);
        statesList.remove(index);
        return restoreState;
    }
    /*
    public Memento undo(){
        Memento restoreState = statesList.get(index);
        //statesList.remove(index);
        return restoreState;
    }

    public Memento redo(){
        Memento restoreState = statesList.get(index);
        statesList.remove(index);
        return restoreState;
    }*/

    public int size(){
        return statesList.size();
    }
}
```

MementoApp:

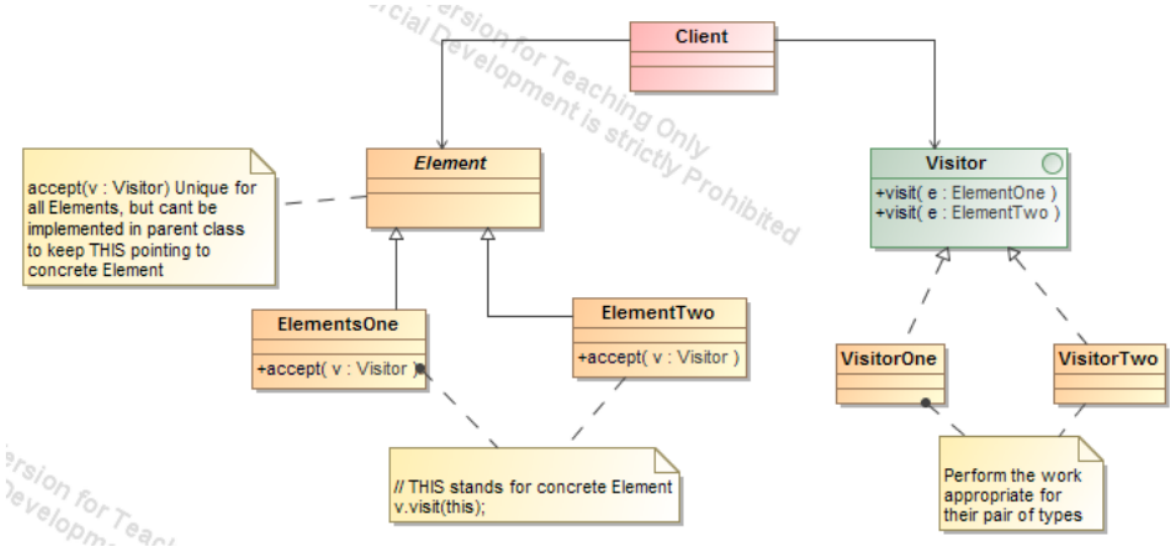
```java
public class MementoApp {

    public static void main(String[] args){

        Caretaker ct = new Caretaker();

        Originator org = new Originator("initial");

        Memento state1 = org.saveState();
        ct.add(state1);

        org.setState("123");
        Memento state2 = org.saveState();
        ct.add(state2);

        org.setState("456");
        Memento state3 = org.saveState();
        ct.add(state3);

        org.setState("789");

        Memento restoreState = ct.get( ct.size() - 1 );
        org.restoreState(restoreState);

        //klientas neturi priejimo prie Originator privaciu atributu
        //restoreState.getState(org);

        Memento restoreState1 = ct.get( ct.size() - 1 );
        org.restoreState(restoreState1);
    }

}
```

11. Visitor

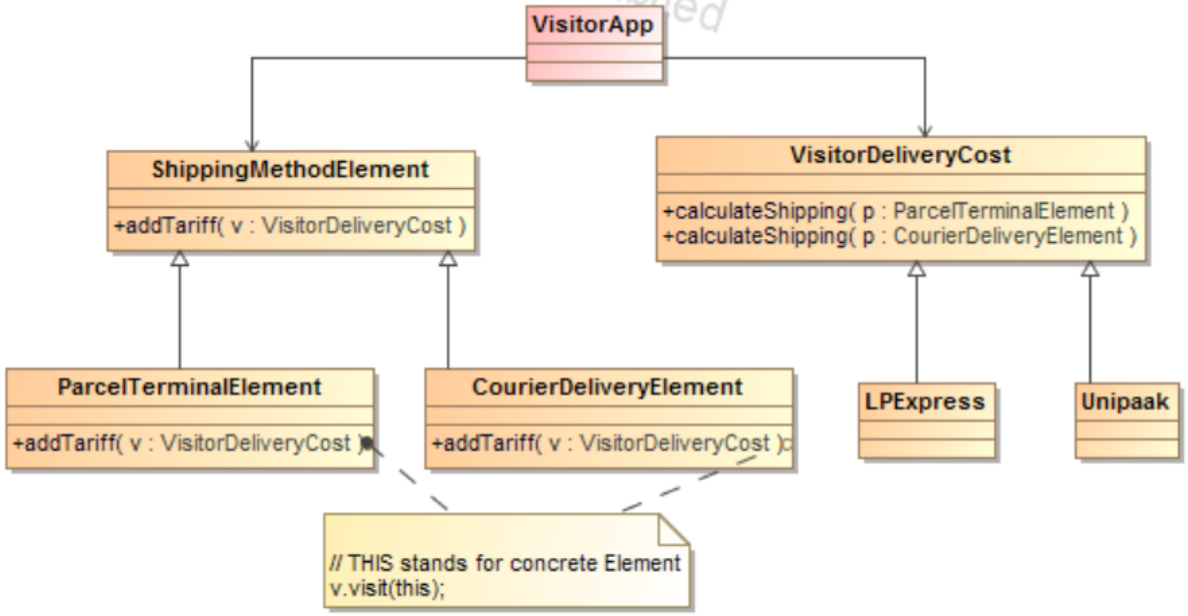
Visitor: when to use

- When we want to add additional functions to "Element" classes without changing them thus adding functionality to "Visitor" classes.

Visitor GoF structure



Visitor Sample code



ShippingMethod:

```
public interface ShippingMethod {

     public double addTariff(DeliveryCost tariff);

}
```

ParcelTerminal:

```
public class ParcelTerminal implements ShippingMethod{

    double baseCost = 0;

    public ParcelTerminal(double price){
        baseCost = price;
        System.out.println("ParcelTerminal base cost: " + baseCost);
    }

    public double getCost(){
        return baseCost;
    }

    @Override
    public double addTariff(DeliveryCost tariff) {
        return tariff.calculateShipping(this);
    }


}
```

SpecialDelivery:

```
public class SpecialDelivery implements ShippingMethod{


    double baseCost = 0;

    public SpecialDelivery(double price){
        baseCost = price;
        System.out.println("SpecialDelivery base cost: " + baseCost);
    }

    public double payCash(){
        return baseCost;
    }

    @Override
    public double addTariff(DeliveryCost tariff) {
        return tariff.calculateShipping(this);
    }

    public void callClient() {
        System.out.println("Call client new method added");
    }

}
```

Post:

```java
public class Post implements ShippingMethod{

    double baseCost = 0;

    public Post(double price){
        baseCost = price;
        System.out.println("Post base cost: " + baseCost);

    }

    public double payProformaInvoice(){
        return baseCost;
    }

    @Override
    public double addTariff(DeliveryCost tariff) {
        return tariff.calculateShipping(this);
    }

}
```

DeliveryCost:

```java
public interface DeliveryCost {

    public double calculateShipping(ParcelTerminal shipping);
    public double calculateShipping(Post shipping);
    public double calculateShipping(SpecialDelivery shipping);

}
```

LPExpress:

```java
public class LPExpress implements DeliveryCost {

    @Override
    public double calculateShipping(ParcelTerminal shipping) {
        //shipping.payProformaInvoice() + 6;
        //shipping.payCash() + 7;
        return shipping.getCost() + 5;

    }

    @Override
    public double calculateShipping(Post shipping) {
        return shipping.payProformaInvoice() + 6;
    }

    @Override
    public double calculateShipping(SpecialDelivery shipping) {
        System.out.println("reikia kazkur ateiti...");
        shipping.callClient();
        return shipping.payCash() + 7;
    }

}
```

Unipaak:

```
public class Unipaak implements DeliveryCost{

    @Override
    public double calculateShipping(ParcelTerminal shipping) {
        return shipping.getCost() + 8.2;

    }

    @Override
    public double calculateShipping(Post shipping) {
        return shipping.payProformaInvoice() + 9.5;
    }

    @Override
    public double calculateShipping(SpecialDelivery shipping) {
        System.out.println("kurjeris skambina klientui...");
        return shipping.payCash() + 11.37;
    }
}
```

VisitorApp(main):

```
public class VisitorApp {

    public static void main(String[] args){

        ParcelTerminal method1 = new ParcelTerminal(10);
        Post method2 = new Post(8);
        SpecialDelivery method3 = new SpecialDelivery(4.5);

        System.out.println("==========================================");

        LPExpress a1_vezejas = new LPExpress();

        System.out.println("LP Parcel terminal: " + method1.addTariff(a1_vezejas) );
        System.out.println("LP Post: " + method2.addTariff(a1_vezejas) );
        System.out.println("LP Special delivery: " + method3.addTariff(a1_vezejas) );

        System.out.println("==========================================");

        Unipaak a2_vezejas = new Unipaak();

        System.out.println("UNI Parcel terminal: " + method1.addTariff(a2_vezejas) );
        System.out.println("UNI Post: " + method2.addTariff(a2_vezejas) );
        System.out.println("UNI Special delivery: " + method3.addTariff(a2_vezejas) );

        System.out.println("==========================================");

        Post a3_vezejas = new Post(3);

        System.out.println("UNI Parcel terminal: " + method1.addTariff(a2_vezejas) );
        System.out.println("UNI Post: " + method2.addTariff(a2_vezejas) );
        System.out.println("UNI Special delivery: " + method3.addTariff(a2_vezejas) );

    }

}
```