CS 130 Software Engineering

Professor Miryung Kim

Class Activity: Adaptor Design Pattern

```
package headfirst.adapter.ducks;

public class DuckTestDrive {
      public static void main(String[] args) {
            MallardDuck duck = new MallardDuck();

            WildTurkey turkey = new WildTurkey();
            Duck turkeyAdapter = new TurkeyAdapter(turkey);

            System.out.println("The Turkey says...");
            turkey.gobble();
            turkey.fly();

            System.out.println("\nThe Duck says...");
            testDuck(duck);

            System.out.println("\nThe TurkeyAdapter says...");
            testDuck(turkeyAdapter);
      }

      static void testDuck(Duck duck) {
            duck.quack();
            duck.fly();// calling the ideal interface of Duck that I
want to call.
      }
}
package headfirst.adapter.ducks;

public interface Turkey {
      public void gobble();
      public void fly();
}
package headfirst.adapter.ducks;

public class WildTurkey implements Turkey {
      public void gobble() {
            System.out.println("Gobble gobble");
      }

      public void fly() {
            System.out.println("I'm flying a short distance");
      }
```

```
}
package headfirst.adapter.ducks;

public class TurkeyAdapter implements Duck {
      Turkey turkey;

      public TurkeyAdapter(Turkey turkey) {
            this.turkey = turkey;
      }

      public void quack() {
            turkey.gobble();
      }

      public void fly() {
            for(int i=0; i < 5; i++) {
                  turkey.fly();
            }
      }
}
package headfirst.adapter.ducks;

public class TurkeyTestDrive {
      public static void main(String[] args) {
            MallardDuck duck = new MallardDuck();
            Turkey duckAdapter = new DuckAdapter(duck);

            for(int i=0;i<10;i++) {
                  System.out.println("The DuckAdapter says...");
                  duckAdapter.gobble();
                  duckAdapter.fly();
            }
      }
}
package headfirst.adapter.ducks;

public interface Duck {
      public void quack();
      public void fly();
}
package headfirst.adapter.ducks;

public class MallardDuck implements Duck {
      public void quack() {
            System.out.println("Quack");
      }

      public void fly() {
            System.out.println("I'm flying");
      }
}

package headfirst.adapter.ducks;
import java.util.Random;

public class DuckAdapter implements Turkey {
      Duck duck;
      Random rand;

      public DuckAdapter(Duck duck) {
            this.duck = duck;
            rand = new Random();
      }

      public void gobble() {
            duck.quack();
      }

      public void fly() {
            if (rand.nextInt(5)  == 0) {
                 duck.fly();
            }
      }
}
```

```java
package headfirst.adapter.iterenum;

import java.util.*;

public class EI {
    public static void main (String args[]) {
        Vector v = new Vector(Arrays.asList(args));
        Enumeration enumeration = v.elements();
        while (enumeration.hasMoreElements()) {
            System.out.println(enumeration.nextElement());
        }
        Iterator iterator = v.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
package headfirst.adapter.iterenum;

import java.util.*;

public class EnumerationIterator implements Iterator {
    Enumeration enumeration;

    public EnumerationIterator(Enumeration enumeration) {
        this.enumeration = enumeration;
    }

    public boolean hasNext() {
        return enumeration.hasMoreElements();
    }

    public Object next() {
        return enumeration.nextElement();
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
package headfirst.adapter.iterenum;

import java.util.*;

public class EnumerationIteratorTestDrive {
    public static void main (String args[]) {
        Vector v = new Vector(Arrays.asList(args));
        Iterator iterator = new EnumerationIterator(v.elements());
```

```
                  while (iterator.hasNext()) {
                        System.out.println(iterator.next());
                  }
            }
}
package headfirst.adapter.iterenum;

import java.util.*;

public class IteratorEnumeration implements Enumeration {
      Iterator iterator;

      public IteratorEnumeration(Iterator iterator) {
            this.iterator = iterator;
      }

      public boolean hasMoreElements() {
            return iterator.hasNext();
      }

      public Object nextElement() {
            return iterator.next();
      }
}
package headfirst.adapter.iterenum;

import java.util.*;

public class IteratorEnumerationTestDrive {
      public static void main (String args[]) {
            ArrayList l = new ArrayList(Arrays.asList(args));
            Enumeration enumeration = new
IteratorEnumeration(l.iterator());
            while (enumeration.hasMoreElements()) {
                  System.out.println(enumeration.nextElement());
            }
      }
}
```

```
class LegacyLine { // legacy code / adaptee that needs to be adapted.

    public void draw(int x1, int y1, int x2, int y2) {
        System.out.println("line from (" + x1 + ',' + y1 + ") to ("
            + x2 + ',' + y2 + ')');
    }
}
class LegacyRectangle {// legacy code, or adaptee that needs to adapted .
    public void draw(int x, int y, int w, int h) { // width and height //
not ideal interface.
        System.out.println("rectangle at (" + x + ',' + y + ") with width "
            + w + " and height " + h);
    }
}
public class OldDemo {
    public static void main(String[] args) {
        Object[] shapes = { new LegacyLine(), new LegacyRectangle() };
        int x1 = 10, y1 = 20, x2 = 30, y2 = 60;
        for (int i = 0; i < shapes.length; ++i) {
          if (shapes[i].getClass().getName().equals("LegacyLine"))
            (LegacyLine)shapes[i].draw(x1, y1, x2, y2);
          else if
(shapes[i].getClass().getName().equals("LegacyRectangle"))
            (LegacyRectangle)shapes[i].draw(Math.min(x1, x2),
                   Math.min(y1, y2), Math.abs(x2 - x1), Math.abs(y2 - y1));
        }
    }
}


interface Shape {

  void draw(int x1, int y1, int x2, int y2);// ideal interface that New
Demo (client code) wants to use.
}
class Line implements Shape {
    private LegacyLine ll = new LegacyLine();
    public void draw(int x1, int y1, int x2, int y2) {
        ll.draw(x1, y1, x2, y2);
    }
}
class Rectangle implements Shape {
    private LegacyRectangle lr = new LegacyRectangle();
    public void draw(int x1, int y1, int x2, int y2) {
        lr.draw(Math.min(x1, x2), Math.min(y1, y2),
             Math.abs(x2 - x1), Math.abs(y2 - y1));
    }
}
public class NewDemo { // client
    public static void main(String[] args) {
        ArrayList<Shape> shapes = new ArrayList<Shape>();
    shapes.add(new Line());
    shapes.add(new Rectangle());
    int x1 = 10, y1 = 20, x2 = 30, y2 = 60;
    for (Shape s : shapes)
     s.draw(x1, y1, x2, y2); // passing the start and end point here.
  }
}
```

CS 130 Software Engineering

Professor Miryung Kim

Class Activity: Template Method, State, Proxy

(A) The following is a code snippet using Swing frame. Where can you find the use of a template method pattern?

```
package headfirst.templatemethod.frame;

import java.awt.*;
import javax.swing.*;

public class MyFrame extends JFrame {

      public MyFrame(String title) {
            super(title);
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setSize(300,300);
            this.setVisible(true);
      }

      public void paint(Graphics graphics) {
            super.paint(graphics);
            String msg = "I rule!!";
            graphics.drawString(msg, 100, 100);
      }

      public static void main(String[] args) {
            MyFrame myFrame = new MyFrame("Head First Design 
Patterns");
      }
}
```

(B) The following program is a sorting program for Ducks. Where can you find the use of a template method pattern?

```
public class Duck implements Comparable {
      String name;
      int weight;

      public Duck(String name, int weight) {
            this.name = name;
            this.weight = weight;
```

```
    }

    public String toString() {
        return name + " weighs " + weight;
    }



    public int compareTo(Object object) {

        Duck otherDuck = (Duck)object;

        if (this.weight < otherDuck.weight) {
            return -1;
        } else if (this.weight == otherDuck.weight) {
            return 0;
        } else { // this.weight > otherDuck.weight
            return 1;
        }
    }
}
public class DuckSortTestDrive {

    public static void main(String[] args) {
        Duck[] ducks = {
                                new Duck("Daffy", 8),
                                new Duck("Dewey", 2),
                                new Duck("Howard", 7),
                                new Duck("Louie", 2),
                                new Duck("Donald", 10),
                                new Duck("Huey", 2)
         };

        System.out.println("Before sorting:");
        display(ducks);

        Arrays.sort(ducks);

        System.out.println("\nAfter sorting:");
        display(ducks);
    }

    public static void display(Duck[] ducks) {
        for (int i = 0; i < ducks.length; i++) {
            System.out.println(ducks[i]);
        }
    }
```

State Pattern

```
package headfirst.state.gumballstate;

public class GumballMachine {

	State soldOutState;
	State noQuarterState;
	State hasQuarterState;
	State soldState;

	State state = soldOutState;
	int count = 0;

	public GumballMachine(int numberGumballs) {
		soldOutState = new SoldOutState(this);
		noQuarterState = new NoQuarterState(this);
		hasQuarterState = new HasQuarterState(this);
		soldState = new SoldState(this);

		this.count = numberGumballs;
		if (numberGumballs > 0) {
			state = noQuarterState;
		}
	}

	public void insertQuarter() {
		state.insertQuarter();
	}

	public void ejectQuarter() {
		state.ejectQuarter();
	}

	public void turnCrank() {
		state.turnCrank();
		state.dispense();
	}

	void setState(State state) {
		this.state = state;
	}

	void releaseBall() {
		System.out.println("A gumball comes rolling out the slot...");
		if (count != 0) {
			count = count - 1;
		}
	}
```

```
          int getCount() {
                    return count;
          }

          void refill(int count) {
                    this.count = count;
                    state = noQuarterState;
          }

   public State getState() {
      return state;
   }

   public State getSoldOutState() {
      return soldOutState;
   }

   public State getNoQuarterState() {
      return noQuarterState;
   }

   public State getHasQuarterState() {
      return hasQuarterState;
   }

   public State getSoldState() {
      return soldState;
   }

          public String toString() {
                    StringBuffer result = new StringBuffer();
                    result.append("\nMighty Gumball, Inc.");
                    result.append("\nJava-enabled Standing Gumball Model #2004");
                    result.append("\nInventory: " + count + " gumball");
                    if (count != 1) {
                              result.append("s");
                    }
                    result.append("\n");
                    result.append("Machine is " + state + "\n");
                    return result.toString();
          }
}

package headfirst.state.gumballstate;

public class SoldState implements State {

   GumballMachine gumballMachine;
```

```
  public SoldState(GumballMachine gumballMachine) {
    this.gumballMachine = gumballMachine;
  }

        public void insertQuarter() {
                System.out.println("Please wait, we're already giving you a gumball");
        }

        public void ejectQuarter() {
                System.out.println("Sorry, you already turned the crank");
        }

        public void turnCrank() {
                System.out.println("Turning twice doesn't get you another gumball!");
        }

        public void dispense() {
                gumballMachine.releaseBall();
                if (gumballMachine.getCount() > 0) {
                        gumballMachine.setState(gumballMachine.getNoQuarterState());
                } else {
                        System.out.println("Oops, out of gumballs!");
                        gumballMachine.setState(gumballMachine.getSoldOutState());
                }
        }

        public String toString() {
                return "dispensing a gumball";
        }
}
```

Activity 1: Transform the TV Remote Application

```
public class TVRemoteBasic {
        private String state = "";

        public void setState(String state) {
                this.state = state;
        }

        public void doAction() {
                if (state.equalsIgnoreCase("ON")) {
                    System.out.println("TV is turned ON");
                } else if (state.equalsIgnoreCase("OFF")) {
                    System.out.println("TV is turned OFF");
                }
        }

        public static void main(String args[]) {
                TVRemoteBasic remote = new TVRemoteBasic();
                remote.setState("ON");
                remote.doAction();
                remote.setState("OFF");
                remote.doAction();

        }

}
```

Transform the above code by applying State Design Pattern

```
public interface State {
 (1) -----------------------------------------------------------------
}


public class TVOnState implements State {
   @Override

(2)--------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------------

}


public class TVOffState implements State {

(3)--------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------------

}
```

```
public class TVContext implements State {

          private State tvState;

          public void setState(State state) {
                    this.tvState = state;
          }

          public State getState() {
                    return this.tvState;
          }

          (4)----------------------------------------------------------

          ---------------------------------------------------------------

          ---------------------------------------------------------------

}

public class TVRemote {

   public static void main(String[] args) {

      TVContext context = new TVContext();
      State tvStartState = new TVStartState();
      State tvStopState = new TVStopState();
      context.setState(tvStartState);
      context.doAction();
      context.setState(tvStopState);
      context.doAction();
   }

}
----------------------------------------------------------------------------------------------------------------------
-------
----------------------------------------------------------------------------------------------------------------------
-------
```

Activity 2: Identify the class diagram components

```
class CeilingFan {
          private State currentState;
```

```
    public CeilingFan() {
        currentState = new Off();
    }

    public void setState(State s) {
        currentState = s;
    }

    public void pull() {
        currentState.pull(this);
    }
}
------------------------------------------------------------------------------------------------------------
------------------

interface State {
    void pull(CeilingFan wrapper);
}
------------------------------------------------------------------------------------------------------------
------------------
class OffState implements State {
    public void pull(CeilingFan wrapper) {
        wrapper.setState(new Low());
        System.out.println("   low speed");
    }
}

------------------------------------------------------------------------------------------------------------
------------------
class LowState implements State {
    public void pull(CeilingFan wrapper) {
        wrapper.setState(new Medium());
        System.out.println("   medium speed");
    }
}

------------------------------------------------------------------------------------------------------------
------------------
class MediumState implements State {
    public void pull(CeilingFan wrapper) {
        wrapper.setState(new High());
        System.out.println("   high speed");
    }
}
```

```
class HighState implements State {
        public void pull(CeilingFan wrapper) {
                wrapper.setState(new Off());
                System.out.println("   turning off");
        }
}


public class Demo {
        public static void main(String[] args) {
                CeilingFan chain = new CeilingFan();
                while (true) {
                        System.out.print("Press ");
                        getLine();
                        chain.pull();
                }
        }

        static String getLine() {
                BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
                String line = null;
                try {
                        line = in.readLine();
                } catch (IOException ex) {
                        ex.printStackTrace();
                }
                return line;
        }
}
```

1.Identify the class the Context class

2.Identify the State Interface

3.Identify the ConcreteState classes

Activity 3: Apply State Design Pattern to create an application

The following are the requirements of the application.

Requirements:

"Bob is trying to implement his own Editor, he has thought to create the Editor to operate in different modes. The ***modes*** *that his editor would support are* ***insert, commandKey*** *and* ***commandLine*** *mode. If you were the architect for bob's Editor application. How would you design the Editor by applying* ***State Pattern*** *that you learned in the class ?"*

Answer: