**Strategy Pattern**

**1)DecoyDuck.java**

**package headfirst.strategy;**

**public class DecoyDuck extends Duck {**

**public DecoyDuck() {**

**setFlyBehavior(new FlyNoWay());**

**setQuackBehavior(new MuteQuack());**

**}**

**public void display() {**

**System.out.println("I'm a duck Decoy");**

**}**

**}**

**2)Duck.java**

**package headfirst.strategy;**

**public abstract class Duck {**

**FlyBehavior flyBehavior;**

**QuackBehavior quackBehavior;**

**public Duck() {**

**}**

**public void setFlyBehavior (FlyBehavior fb) {**

**flyBehavior = fb;**

**}**

**public void setQuackBehavior(QuackBehavior qb) {**

**quackBehavior = qb;**

**}**

**abstract void display();**

**public void performFly() {**

**flyBehavior.fly();**

**}**

**public void performQuack() {**

**quackBehavior.quack();**

**}**

**public void swim() {**

**System.out.println("All ducks float, even decoys!");**

**}**

**}**

**3)FakeQuack.java**

**package headfirst.strategy;**

**public class FakeQuack implements QuackBehavior {**

**public void quack() {**

**System.out.println("Qwak");**

**}**

**}**

**4)FlyBehaviour.java**

**package headfirst.strategy;**

**public interface FlyBehavior {**

**public void fly();**

**}**

**5)FlyNoWay.java**

**package headfirst.strategy;**

**public class FlyNoWay implements FlyBehavior {**

**public void fly() {**

**System.out.println("I can't fly");**

**}**

**}**

**6)FlyRocketPowered.java**

**package headfirst.strategy;**

**public class FlyRocketPowered implements FlyBehavior {**

**public void fly() {**

**System.out.println("I'm flying with a rocket");**

**} }**

**7)FlyWithWings.java**

**package headfirst.strategy;**

**public class FlyWithWings implements FlyBehavior {**

**public void fly() {**

**System.out.println("I'm flying!!");**

**}**

**}**

**8)MallardDuck.java**

**package headfirst.strategy;**

**public class MallardDuck extends Duck {**

**public MallardDuck() {**

**quackBehavior = new Quack();**

**flyBehavior = new FlyWithWings();**

**}**

**public void display() {**

**System.out.println("I'm a real Mallard duck");**

**}**

**}**

**9)MiniDuckSimulator.java**

**package headfirst.strategy;**

**public class MiniDuckSimulator {**

**public static void main(String[] args) {**

**MallardDuck mallard = new MallardDuck();**

**RubberDuck rubberDuckie = new RubberDuck();**

**DecoyDuck decoy = new DecoyDuck();**

**ModelDuck model = new ModelDuck();**

**mallard.performQuack();**

**rubberDuckie.performQuack();**

**decoy.performQuack();**

**model.performFly();**

**model.setFlyBehavior(new FlyRocketPowered());**

**model.performFly();**

**}**

**}**

**10)MiniDuckSimulator1.java**

**package headfirst.strategy;**

**public class MiniDuckSimulator1 {**

**public static void main(String[] args) {**

**Duck mallard = new MallardDuck();**

**mallard.performQuack();**

**mallard.performFly();**

**Duck model = new ModelDuck();**

**model.performFly();**

**model.setFlyBehavior(new FlyRocketPowered());**

**model.performFly();**

**}**

**}**

**11)ModelDuck.java**

**package headfirst.strategy;**

**public class ModelDuck extends Duck {**

**public ModelDuck() {**

**flyBehavior = new FlyNoWay();**

**quackBehavior = new Quack();**

**}**

**public void display() {**

**System.out.println("I'm a model duck");**

**}**

**}**

**12)MuteQuack.java**

**package headfirst.strategy;**

**public class MuteQuack implements QuackBehavior {**

**public void quack() {**

**System.out.println("<< Silence >>");**

**}**

**}**

**13)Quack.java**

**package headfirst.strategy;**

**public class Quack implements QuackBehavior {**

**public void quack() {**

**System.out.println("Quack");**

**}**

**}**

**14)QuackBehaviour.java**

**package headfirst.strategy;**

**public interface QuackBehavior {**

**public void quack();**

**}**

**15)RedHeadDuck.java**

**package headfirst.strategy;**

**public class RedHeadDuck extends Duck {**

**public RedHeadDuck() {**

**flyBehavior = new FlyWithWings();**

**quackBehavior = new Quack();**

**}**

**public void display() {**

**System.out.println("I'm a real Red Headed duck");**

**}**

**}**

**16)RubberDuck.java**

**package headfirst.strategy;**

**public class RubberDuck extends Duck {**

**public RubberDuck() {**

**flyBehavior = new FlyNoWay();**

**quackBehavior = new Squeak();**

**}**

**public void display() {**

**System.out.println("I'm a rubber duckie");**

**}**

**}**

**17)Squeak.java**

**package headfirst.strategy;**

**public class Squeak implements QuackBehavior {**

**public void quack() {**

**System.out.println("Squeak");**

**}**

**}**

**State Pattern**

* **GumBallState**

**1)GumBallMachine.java**

**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();**

**}**

**}**

**2)GumBallMachineStateDrive.java**

**package headfirst.state.gumballstate;**

**public class GumballMachineTestDrive {**

**public static void main(String[] args) {**

**GumballMachine gumballMachine = new GumballMachine(5);**

**System.out.println(gumballMachine);**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**System.out.println(gumballMachine);**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**System.out.println(gumballMachine);**

**}**

**}**

**3)HasQuarterState.java**

**package headfirst.state.gumballstate;**

**import java.util.Random;**

**public class HasQuarterState implements State {**

**GumballMachine gumballMachine;**

**public HasQuarterState(GumballMachine gumballMachine) {**

**this.gumballMachine = gumballMachine;**

**}**

**public void insertQuarter() {**

**System.out.println("You can't insert another quarter");**

**}**

**public void ejectQuarter() {**

**System.out.println("Quarter returned");**

**gumballMachine.setState(gumballMachine.getNoQuarterState());**

**}**

**public void turnCrank() {**

**System.out.println("You turned...");**

**gumballMachine.setState(gumballMachine.getSoldState());**

**}**

**public void dispense() {**

**System.out.println("No gumball dispensed");**

**}**

**public String toString() {**

**return "waiting for turn of crank";**

**}**

**}**

**4)NoQuarterState.java**

**package headfirst.state.gumballstate;**

**public class NoQuarterState implements State {**

**GumballMachine gumballMachine;**

**public NoQuarterState(GumballMachine gumballMachine) {**

**this.gumballMachine = gumballMachine;**

**}**

**public void insertQuarter() {**

**System.out.println("You inserted a quarter");**

**gumballMachine.setState(gumballMachine.getHasQuarterState());**

**}**

**public void ejectQuarter() {**

**System.out.println("You haven't inserted a quarter");**

**}**

**public void turnCrank() {**

**System.out.println("You turned, but there's no quarter");**

**}**

**public void dispense() {**

**System.out.println("You need to pay first");**

**}**

**public String toString() {**

**return "waiting for quarter";**

**}**

**}**

**5)SoldOutState.java**

**package headfirst.state.gumballstate;**

**public class SoldOutState implements State {**

**GumballMachine gumballMachine;**

**public SoldOutState(GumballMachine gumballMachine) {**

**this.gumballMachine = gumballMachine;**

**}**

**public void insertQuarter() {**

**System.out.println("You can't insert a quarter, the machine is sold out");**

**}**

**public void ejectQuarter() {**

**System.out.println("You can't eject, you haven't inserted a quarter yet");**

**}**

**public void turnCrank() {**

**System.out.println("You turned, but there are no gumballs");**

**}**

**public void dispense() {**

**System.out.println("No gumball dispensed");**

**}**

**public String toString() {**

**return "sold out";**

**}**

**}**

**6)SoldState.java**

**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";**

**}**

**}**

**7)State.java**

**package headfirst.state.gumballstate;**

**public interface State {**

**public void insertQuarter();**

**public void ejectQuarter();**

**public void turnCrank();**

**public void dispense();**

**}**

**—---------------------------------------------------------------------**

* **GumBallStateWinner**

**1)GumBallMachine.java**

**package headfirst.state.gumballstatewinner;**

**public class GumballMachine {**

**State soldOutState;**

**State noQuarterState;**

**State hasQuarterState;**

**State soldState;**

**State winnerState;**

**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);**

**winnerState = new WinnerState(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 State getWinnerState() {**

**return winnerState;**

**}**

**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();**

**}**

**}**

**2)GumBallMachineTestDrive.java**

**package headfirst.state.gumballstatewinner;**

**public class GumballMachineTestDrive {**

**public static void main(String[] args) {**

**GumballMachine gumballMachine =**

**new GumballMachine(10);**

**System.out.println(gumballMachine);**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**System.out.println(gumballMachine);**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**System.out.println(gumballMachine);**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**System.out.println(gumballMachine);**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**System.out.println(gumballMachine);**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**gumballMachine.insertQuarter();**

**gumballMachine.turnCrank();**

**System.out.println(gumballMachine);**

**}**

**}**

**3)HasQuarterState.java**

**package headfirst.state.gumballstatewinner;**

**import java.util.Random;**

**public class HasQuarterState implements State {**

**Random randomWinner = new Random(System.currentTimeMillis());**

**GumballMachine gumballMachine;**

**public HasQuarterState(GumballMachine gumballMachine) {**

**this.gumballMachine = gumballMachine;**

**}**

**public void insertQuarter() {**

**System.out.println("You can't insert another quarter");**

**}**

**public void ejectQuarter() {**

**System.out.println("Quarter returned");**

**gumballMachine.setState(gumballMachine.getNoQuarterState());**

**}**

**public void turnCrank() {**

**System.out.println("You turned...");**

**int winner = randomWinner.nextInt(10);**

**if ((winner == 0) && (gumballMachine.getCount() > 1)) {**

**gumballMachine.setState(gumballMachine.getWinnerState());**

**} else {**

**gumballMachine.setState(gumballMachine.getSoldState());**

**}**

**}**

**public void dispense() {**

**System.out.println("No gumball dispensed");**

**}**

**public String toString() {**

**return "waiting for turn of crank";**

**}**

**}**

**4)NoQuarterState.java**

**package headfirst.state.gumballstatewinner;**

**public class NoQuarterState implements State {**

**GumballMachine gumballMachine;**

**public NoQuarterState(GumballMachine gumballMachine) {**

**this.gumballMachine = gumballMachine;**

**}**

**public void insertQuarter() {**

**System.out.println("You inserted a quarter");**

**gumballMachine.setState(gumballMachine.getHasQuarterState());**

**}**

**public void ejectQuarter() {**

**System.out.println("You haven't inserted a quarter");**

**}**

**public void turnCrank() {**

**System.out.println("You turned, but there's no quarter");**

**}**

**public void dispense() {**

**System.out.println("You need to pay first");**

**}**

**public String toString() {**

**return "waiting for quarter";**

**}**

**}**

**5)SoldOutState.java**

**package headfirst.state.gumballstatewinner;**

**public class SoldOutState implements State {**

**GumballMachine gumballMachine;**

**public SoldOutState(GumballMachine gumballMachine) {**

**this.gumballMachine = gumballMachine;**

**}**

**public void insertQuarter() {**

**System.out.println("You can't insert a quarter, the machine is sold out");**

**}**

**public void ejectQuarter() {**

**System.out.println("You can't eject, you haven't inserted a quarter yet");**

**}**

**public void turnCrank() {**

**System.out.println("You turned, but there are no gumballs");**

**}**

**public void dispense() {**

**System.out.println("No gumball dispensed");**

**}**

**public String toString() {**

**return "sold out";**

**}**

**}**

**6)SoldState.java**

**package headfirst.state.gumballstatewinner;**

**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";**

**}**

**}**

**7)State.java**

**package headfirst.state.gumballstatewinner;**

**public interface State {**

**public void insertQuarter();**

**public void ejectQuarter();**

**public void turnCrank();**

**public void dispense();**

**}**

**8)WinnerState.java**

**package headfirst.state.gumballstatewinner;**

**public class WinnerState implements State {**

**GumballMachine gumballMachine;**

**public WinnerState(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("Please wait, we're already giving you a Gumball");**

**}**

**public void turnCrank() {**

**System.out.println("Turning again doesn't get you another gumball!");**

**}**

**public void dispense() {**

**System.out.println("YOU'RE A WINNER! You get two gumballs for your quarter");**

**gumballMachine.releaseBall();**

**if (gumballMachine.getCount() == 0) {**

**gumballMachine.setState(gumballMachine.getSoldOutState());**

**} else {**

**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 "despensing two gumballs for your quarter, because YOU'RE A WINNER!";**

**}**

**}**

**Singleton Pattern**

* **Chocolate**

**1)ChocolateBoiler.java**

**package headfirst.singleton.chocolate;**

**public class ChocolateBoiler {**

**private boolean empty;**

**private boolean boiled;**

**private static ChocolateBoiler uniqueInstance;**

**private ChocolateBoiler() {**

**empty = true;**

**boiled = false;**

**}**

**public static ChocolateBoiler getInstance() {**

**if (uniqueInstance == null) {**

**System.out.println("Creating unique instance of Chocolate Boiler");**

**uniqueInstance = new ChocolateBoiler();**

**}**

**System.out.println("Returning instance of Chocolate Boiler");**

**return uniqueInstance;**

**}**

**public void fill() {**

**if (isEmpty()) {**

**empty = false;**

**boiled = false;**

**// fill the boiler with a milk/chocolate mixture**

**}**

**}**

**public void drain() {**

**if (!isEmpty() && isBoiled()) {**

**// drain the boiled milk and chocolate**

**empty = true;**

**}**

**}**

**public void boil() {**

**if (!isEmpty() && !isBoiled()) {**

**// bring the contents to a boil**

**boiled = true;**

**}**

**}**

**public boolean isEmpty() {**

**return empty;**

**}**

**public boolean isBoiled() {**

**return boiled;**

**}**

**}**

**2)ChocolateController.java**

**package headfirst.singleton.chocolate;**

**public class ChocolateController {**

**public static void main(String args[]) {**

**ChocolateBoiler boiler = ChocolateBoiler.getInstance();**

**boiler.fill();**

**boiler.boil();**

**boiler.drain();**

**// will return the existing instance**

**ChocolateBoiler boiler2 = ChocolateBoiler.getInstance();**

**}**

**}**

**—---------------------------------------------------------------------**

* **DCL**

**1)Singleton.java**

**package headfirst.singleton.dcl;**

**//**

**// Danger! This implementation of Singleton not**

**// guaranteed to work prior to Java 5**

**//**

**public class Singleton {**

**private volatile static Singleton uniqueInstance;**

**private Singleton() {}**

**public static Singleton getInstance() {**

**if (uniqueInstance == null) {**

**synchronized (Singleton.class) {**

**if (uniqueInstance == null) {**

**uniqueInstance = new Singleton();**

**}**

**}**

**}**

**return uniqueInstance;**

**}**

**}**

**2)SingletonClient.java**

**package headfirst.singleton.dcl;**

**public class SingletonClient {**

**public static void main(String[] args) {**

**Singleton singleton = Singleton.getInstance();**

**}**

**}**

**—-------------------------------------------------------------------**

* **ThreadSafe**

**1)Singleton.java**

**package headfirst.singleton.threadsafe;**

**public class Singleton {**

**private static Singleton uniqueInstance;**

**// other useful instance variables here**

**private Singleton() {}**

**public static synchronized Singleton getInstance() {**

**if (uniqueInstance == null) {**

**uniqueInstance = new Singleton();**

**}**

**return uniqueInstance;**

**}// other useful methods here }**

**—--------------------------------------------------------------------**

* **Classic**

**1)Singleton.java**

**package headfirst.singleton.classic;**

**// NOTE: This is not thread safe!**

**public class Singleton {**

**private static Singleton uniqueInstance;**

**// other useful instance variables here**

**private Singleton() {}**

**public static Singleton getInstance() {**

**if (uniqueInstance == null) {**

**uniqueInstance = new Singleton();**

**}**

**return uniqueInstance;**

**}**

**// other useful methods here**

**}**

**—----------------------------------------------------------------------**

* **Stat**

**1)Singleton.java**

**package headfirst.singleton.stat;**

**public class Singleton {**

**private static Singleton uniqueInstance = new Singleton();**

**private Singleton() {}**

**public static Singleton getInstance() {**

**return uniqueInstance;**

**}**

**}**

**2)SingletonClient.java**

**package headfirst.singleton.stat;**

**public class SingletonClient {**

**public static void main(String[] args) {**

**Singleton singleton = Singleton.getInstance();**

**}**

**}**

**Iterator Pattern**

* **DinerMergerCafe**

**1)AlternatingDinerMenuIterator.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.Iterator;**

**import java.util.Calendar;**

**public class AlternatingDinerMenuIterator implements Iterator {**

**MenuItem[] items;**

**int position;**

**public AlternatingDinerMenuIterator(MenuItem[] items) {**

**this.items = items;**

**Calendar rightNow = Calendar.getInstance();**

**position = rightNow.DAY\_OF\_WEEK % 2;**

**}**

**public Object next() {**

**MenuItem menuItem = items[position];**

**position = position + 2;**

**return menuItem;**

**}**

**public boolean hasNext() {**

**if (position >= items.length || items[position] == null) {**

**return false;**

**} else {**

**return true;**

**}**

**}**

**public void remove() {**

**throw new UnsupportedOperationException(**

**"Alternating Diner Menu Iterator does not support remove()");**

**}**

**}**

**2)CafeMenu.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.\*;**

**public class CafeMenu implements Menu {**

**Hashtable menuItems = new Hashtable();**

**public CafeMenu() {**

**addItem("Veggie Burger and Air Fries","Veggie burger on a whole wheat bun, lettuce, tomato, and fries",true, 3.99);**

**addItem("Soup of the day","A cup of the soup of the day, with a side salad",false, 3.69);**

**addItem("Burrito","A large burrito, with whole pinto beans, salsa, guacamole",true, 4.29);**

**}**

**public void addItem(String name, String description,**

**boolean vegetarian, double price)**

**{**

**MenuItem menuItem = new MenuItem(name, description, vegetarian, price);**

**menuItems.put(menuItem.getName(), menuItem);**

**}**

**public Hashtable getItems() {**

**return menuItems;**

**}**

**public Iterator createIterator() {**

**return menuItems.values().iterator();**

**}**

**}**

**3)DinerMenu.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.Iterator;**

**public class DinerMenu implements Menu {**

**static final int MAX\_ITEMS = 6;**

**int numberOfItems = 0;**

**MenuItem[] menuItems;**

**public DinerMenu() {**

**menuItems = new MenuItem[MAX\_ITEMS];**

**addItem("Vegetarian BLT","(Fakin') Bacon with lettuce & tomato on whole wheat", true, 2.99);**

**addItem("BLT","Bacon with lettuce & tomato on whole wheat", false, 2.99);**

**addItem("Soup of the day","Soup of the day, with a side of potato salad", false, 3.29);**

**addItem("Hotdog","A hot dog, with saurkraut, relish, onions, topped with cheese",false, 3.05);**

**addItem("Steamed Veggies and Brown Rice","A medly of steamed vegetables over brown rice", true, 3.99);**

**addItem("Pasta","Spaghetti with Marinara Sauce, and a slice of sourdough bread",true, 3.89);**

**}**

**public void addItem(String name, String description,**

**boolean vegetarian, double price)**

**{**

**MenuItem menuItem = new MenuItem(name, description, vegetarian, price);**

**if (numberOfItems >= MAX\_ITEMS) {**

**System.err.println("Sorry, menu is full! Can't add item to menu");**

**} else {**

**menuItems[numberOfItems] = menuItem;**

**numberOfItems = numberOfItems + 1;**

**}**

**}**

**public MenuItem[] getMenuItems() {**

**return menuItems;**

**}**

**public Iterator createIterator() {**

**return new DinerMenuIterator(menuItems);**

**//return new AlternatingDinerMenuIterator(menuItems);**

**}**

**// other menu methods here**

**}**

**4)DinerMenuIterator.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.Iterator;**

**public class DinerMenuIterator implements Iterator {**

**MenuItem[] list;**

**int position = 0;**

**public DinerMenuIterator(MenuItem[] list) {**

**this.list = list;**

**}**

**public Object next() {**

**MenuItem menuItem = list[position];**

**position = position + 1;**

**return menuItem;**

**}**

**public boolean hasNext() {**

**if (position >= list.length || list[position] == null) {**

**return false;**

**} else {**

**return true;**

**}**

**}**

**public void remove() {**

**if (position <= 0) {**

**throw new IllegalStateException**

**("You can't remove an item until you've done at least one next()");**

**}**

**if (list[position-1] != null) {**

**for (int i = position-1; i < (list.length-1); i++) {**

**list[i] = list[i+1];**

**}**

**list[list.length-1] = null;**

**}**

**}**

**}**

**5)Menu.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.Iterator;**

**public interface Menu {**

**public Iterator createIterator();**

**}**

**6)MenuItem.java**

**package headfirst.iterator.dinermergercafe;**

**public class MenuItem {**

**String name;**

**String description;**

**boolean vegetarian;**

**double price;**

**public MenuItem(String name, String description,**

**boolean vegetarian,double price)**

**{**

**this.name = name;**

**this.description = description;**

**this.vegetarian = vegetarian;**

**this.price = price;**

**}**

**public String getName() {**

**return name;**

**}**

**public String getDescription() {**

**return description;**

**}**

**public double getPrice() {**

**return price;**

**}**

**public boolean isVegetarian() {**

**return vegetarian;**

**} }**

**7)MenuTestDrive.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.\*;**

**public class MenuTestDrive {**

**public static void main(String args[]) {**

**PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();**

**DinerMenu dinerMenu = new DinerMenu();**

**CafeMenu cafeMenu = new CafeMenu();**

**Waitress waitress = new Waitress(pancakeHouseMenu, dinerMenu, cafeMenu);**

**waitress.printMenu();**

**waitress.printVegetarianMenu();**

**System.out.println("\nCustomer asks, is the Hotdog vegetarian?");**

**System.out.print("Waitress says: ");**

**if (waitress.isItemVegetarian("Hotdog")) {**

**System.out.println("Yes");**

**} else {**

**System.out.println("No");**

**}**

**System.out.println("\nCustomer asks, are the Waffles vegetarian?");**

**System.out.print("Waitress says: ");**

**if (waitress.isItemVegetarian("Waffles")) {**

**System.out.println("Yes");**

**} else {**

**System.out.println("No");**

**}**

**}**

**}**

**8)PancakehouseMenu.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.ArrayList;**

**import java.util.Iterator;**

**public class PancakeHouseMenu implements Menu {**

**ArrayList menuItems;**

**public PancakeHouseMenu() {**

**menuItems = new ArrayList();**

**addItem("K&B's Pancake Breakfast", "Pancakes with scrambled eggs, and toast",true,2.99);**

**addItem("Regular Pancake Breakfast","Pancakes with fried eggs, sausage",false,2.99);**

**addItem("Blueberry Pancakes","Pancakes made with fresh blueberries, and blueberry syrup",true, 3.49);**

**addItem("Waffles","Waffles, with your choice of blueberries or strawberries",true,3.59);**

**}**

**public void addItem(String name, String description,**

**boolean vegetarian, double price)**

**{**

**MenuItem menuItem = new MenuItem(name, description, vegetarian, price);**

**menuItems.add(menuItem);**

**}**

**public ArrayList getMenuItems() {**

**return menuItems;**

**}**

**public Iterator createIterator() {**

**return menuItems.iterator();**

**}**

**// other menu methods here**

**}**

**9)Waitress.java**

**package headfirst.iterator.dinermergercafe;**

**import java.util.Iterator;**

**public class Waitress {**

**Menu pancakeHouseMenu;**

**Menu dinerMenu;**

**Menu cafeMenu;**

**public Waitress(Menu pancakeHouseMenu, Menu dinerMenu, Menu cafeMenu) {**

**this.pancakeHouseMenu = pancakeHouseMenu;**

**this.dinerMenu = dinerMenu;**

**this.cafeMenu = cafeMenu;**

**}**

**public void printMenu() {**

**Iterator pancakeIterator = pancakeHouseMenu.createIterator();**

**Iterator dinerIterator = dinerMenu.createIterator();**

**Iterator cafeIterator = cafeMenu.createIterator();**

**System.out.println("MENU\n----\nBREAKFAST");**

**printMenu(pancakeIterator);**

**System.out.println("\nLUNCH");**

**printMenu(dinerIterator);**

**System.out.println("\nDINNER");**

**printMenu(cafeIterator);**

**}**

**private void printMenu(Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**System.out.print(menuItem.getName() + ", ");**

**System.out.print(menuItem.getPrice() + " -- ");**

**System.out.println(menuItem.getDescription());**

**}**

**}**

**public void printVegetarianMenu() {**

**System.out.println("\nVEGETARIAN MENU\n---------------");**

**printVegetarianMenu(pancakeHouseMenu.createIterator());**

**printVegetarianMenu(dinerMenu.createIterator());**

**printVegetarianMenu(cafeMenu.createIterator());**

**}**

**public boolean isItemVegetarian(String name) {**

**Iterator pancakeIterator = pancakeHouseMenu.createIterator();**

**if (isVegetarian(name, pancakeIterator)) {**

**return true;**

**}**

**Iterator dinerIterator = dinerMenu.createIterator();**

**if (isVegetarian(name, dinerIterator)) {**

**return true;**

**}**

**Iterator cafeIterator = cafeMenu.createIterator();**

**if (isVegetarian(name, cafeIterator)) {**

**return true;**

**}**

**return false;**

**}**

**private void printVegetarianMenu(Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**if (menuItem.isVegetarian()) {**

**System.out.print(menuItem.getName() + ", ");**

**System.out.print(menuItem.getPrice() + " -- ");**

**System.out.println(menuItem.getDescription());**

**}**

**}**

**}**

**private boolean isVegetarian(String name, Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**if (menuItem.getName().equals(name)) {**

**if (menuItem.isVegetarian()) {**

**return true;**

**}**

**}**

**}**

**return false;**

**}**

**}**

**//^^ WaitressCafeMain**

**//^^ WaitressCafe**

**—---------------------------------------------------------------------**

* **Transition**

**1)Menu.java**

**package headfirst.iterator.transition;**

**import java.util.Iterator;**

**public interface Menu {**

**public Iterator createIterator();**

**}**

**2)MenuItem.java**

**package headfirst.iterator.transition;**

**public class MenuItem {**

**String name;**

**String description;**

**boolean vegetarian;**

**double price;**

**public MenuItem(String name, String description,**

**boolean vegetarian,double price)**

**{**

**this.name = name;**

**this.description = description;**

**this.vegetarian = vegetarian;**

**this.price = price;**

**}**

**public String getName() {**

**return name;**

**}**

**public String getDescription() {**

**return description;**

**}**

**public double getPrice() {**

**return price;**

**}**

**public boolean isVegetarian() {**

**return vegetarian;**

**}**

**}**

**3)Waitress.java**

**package headfirst.iterator.transition;**

**import java.util.\*;**

**public class Waitress {**

**ArrayList menus;**

**public Waitress(ArrayList menus) {**

**this.menus = menus;**

**}**

**public void printMenu() {**

**Iterator menuIterator = menus.iterator();**

**while(menuIterator.hasNext()) {**

**Menu menu = (Menu)menuIterator.next();**

**printMenu(menu.createIterator());**

**}**

**}**

**void printMenu(Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**System.out.print(menuItem.getName() + ", ");**

**System.out.print(menuItem.getPrice() + " -- ");**

**System.out.println(menuItem.getDescription());**

**}**

**}**

**}**

**—----------------------------------------------------------------------**

* **DinerMerger1**

**1)AlternatingDinerMenuIterator.java**

**package headfirst.iterator.dinermergeri;**

**import java.util.Iterator;**

**import java.util.Calendar;**

**public class AlternatingDinerMenuIterator implements Iterator {**

**MenuItem[] items;**

**int position;**

**public AlternatingDinerMenuIterator(MenuItem[] items) {**

**this.items = items;**

**Calendar rightNow = Calendar.getInstance();**

**position = rightNow.DAY\_OF\_WEEK % 2;**

**}**

**public Object next() {**

**MenuItem menuItem = items[position];**

**position = position + 2;**

**return menuItem;**

**}**

**public boolean hasNext() {**

**if (position >= items.length || items[position] == null) {**

**return false;**

**} else {**

**return true;**

**}**

**}**

**public void remove() {**

**throw new UnsupportedOperationException(**

**"Alternating Diner Menu Iterator does not support remove()");**

**}**

**}**

**2)DinerMenu.java**

**package headfirst.iterator.dinermergeri;**

**import java.util.Iterator;**

**public class DinerMenu implements Menu {**

**static final int MAX\_ITEMS = 6;**

**int numberOfItems = 0;**

**MenuItem[] menuItems;**

**public DinerMenu() {**

**menuItems = new MenuItem[MAX\_ITEMS];**

**addItem("Vegetarian BLT","(Fakin') Bacon with lettuce & tomato on whole wheat", true, 2.99);**

**addItem("BLT","Bacon with lettuce & tomato on whole wheat", false, 2.99);**

**addItem("Soup of the day","Soup of the day, with a side of potato salad", false, 3.29);**

**addItem("Hotdog","A hot dog, with saurkraut, relish, onions, topped with cheese",false, 3.05);**

**addItem("Steamed Veggies and Brown Rice","Steamed vegetables over brown rice", true, 3.99);**

**addItem("Pasta","Spaghetti with Marinara Sauce, and a slice of sourdough bread",true, 3.89);**

**}**

**public void addItem(String name, String description,**

**boolean vegetarian, double price)**

**{**

**MenuItem menuItem = new MenuItem(name, description, vegetarian, price);**

**if (numberOfItems >= MAX\_ITEMS) {**

**System.err.println("Sorry, menu is full! Can't add item to menu");**

**} else {**

**menuItems[numberOfItems] = menuItem;**

**numberOfItems = numberOfItems + 1;**

**}**

**}**

**public MenuItem[] getMenuItems() {**

**return menuItems;**

**}**

**public Iterator createIterator() {**

**return new DinerMenuIterator(menuItems);**

**//return new AlternatingDinerMenuIterator(menuItems);**

**}**

**// other menu methods here**

**}**

**3)DinerMenuIterator.java**

**package headfirst.iterator.dinermergeri;**

**import java.util.Iterator;**

**public class DinerMenuIterator implements Iterator {**

**MenuItem[] list;**

**int position = 0;**

**public DinerMenuIterator(MenuItem[] list) {**

**this.list = list;**

**}**

**public Object next() {**

**MenuItem menuItem = list[position];**

**position = position + 1;**

**return menuItem;**

**}**

**public boolean hasNext() {**

**if (position >= list.length || list[position] == null) {**

**return false;**

**} else {**

**return true;**

**}**

**}**

**public void remove() {**

**if (position <= 0) {**

**throw new IllegalStateException**

**("You can't remove an item until you've done at least one next()");**

**}**

**if (list[position-1] != null) {**

**for (int i = position-1; i < (list.length-1); i++) {**

**list[i] = list[i+1];**

**}**

**list[list.length-1] = null;**

**}**

**}**

**}**

**4)Menu.java**

**package headfirst.iterator.dinermergeri;**

**import java.util.Iterator;**

**public interface Menu {**

**public Iterator createIterator();**

**}**

**5)MenuItem.java**

**package headfirst.iterator.dinermergeri;**

**public class MenuItem {**

**String name;**

**String description;**

**boolean vegetarian;**

**double price;**

**public MenuItem(String name,String description,**

**boolean vegetarian,double price)**

**{**

**this.name = name;**

**this.description = description;**

**this.vegetarian = vegetarian;**

**this.price = price;**

**}**

**public String getName() {**

**return name;**

**}**

**public String getDescription() {**

**return description;**

**}**

**public double getPrice() {**

**return price;**

**}**

**public boolean isVegetarian() {**

**return vegetarian;**

**}**

**}**

**6)MenuTestDrive.java**

**package headfirst.iterator.dinermergeri;**

**import java.util.\*;**

**public class MenuTestDrive {**

**public static void main(String args[]) {**

**PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();**

**DinerMenu dinerMenu = new DinerMenu();**

**Waitress waitress = new Waitress(pancakeHouseMenu, dinerMenu);**

**waitress.printMenu();**

**waitress.printVegetarianMenu();**

**System.out.println("\nCustomer asks, is the Hotdog vegetarian?");**

**System.out.print("Waitress says: ");**

**if (waitress.isItemVegetarian("Hotdog")) {**

**System.out.println("Yes");**

**} else {**

**System.out.println("No");**

**}**

**System.out.println("\nCustomer asks, are the Waffles vegetarian?");**

**System.out.print("Waitress says: ");**

**if (waitress.isItemVegetarian("Waffles")) {**

**System.out.println("Yes");**

**} else {**

**System.out.println("No");**

**}**

**}**

**}**

**7)PancakeHouseMenu.java**

**package headfirst.iterator.dinermergeri;**

**import java.util.ArrayList;**

**import java.util.Iterator;**

**public class PancakeHouseMenu implements Menu {**

**ArrayList menuItems;**

**public PancakeHouseMenu() {**

**menuItems = new ArrayList();**

**addItem("K&B's Pancake Breakfast", "Pancakes with scrambled eggs, and toast",true,2.99);**

**addItem("Regular Pancake Breakfast","Pancakes with fried eggs, sausage",false,2.99);**

**addItem("Blueberry Pancakes","Pancakes made with fresh blueberries, and blueberry syrup",true, 3.49);**

**addItem("Waffles","Waffles, with your choice of blueberries or strawberries",true,3.59);**

**}**

**public void addItem(String name, String description,**

**boolean vegetarian, double price)**

**{**

**MenuItem menuItem = new MenuItem(name, description, vegetarian, price);**

**menuItems.add(menuItem);**

**}**

**public ArrayList getMenuItems() {**

**return menuItems;**

**}**

**public Iterator createIterator() {**

**return menuItems.iterator();**

**}**

**// other menu methods here**

**}**

**8)Waitress.java**

**package headfirst.iterator.dinermergeri;**

**import java.util.Iterator;**

**public class Waitress {**

**Menu pancakeHouseMenu;**

**Menu dinerMenu;**

**public Waitress(Menu pancakeHouseMenu, Menu dinerMenu) {**

**this.pancakeHouseMenu = pancakeHouseMenu;**

**this.dinerMenu = dinerMenu;**

**}**

**public void printMenu() {**

**Iterator pancakeIterator = pancakeHouseMenu.createIterator();**

**Iterator dinerIterator = dinerMenu.createIterator();**

**System.out.println("MENU\n----\nBREAKFAST");**

**printMenu(pancakeIterator);**

**System.out.println("\nLUNCH");**

**printMenu(dinerIterator);**

**}**

**private void printMenu(Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**System.out.print(menuItem.getName() + ", ");**

**System.out.print(menuItem.getPrice() + " -- ");**

**System.out.println(menuItem.getDescription());**

**}**

**}**

**public void printVegetarianMenu() {**

**System.out.println("\nVEGETARIAN MENU\n----\nBREAKFAST");**

**printVegetarianMenu(pancakeHouseMenu.createIterator());**

**System.out.println("\nLUNCH");**

**printVegetarianMenu(dinerMenu.createIterator());**

**}**

**public boolean isItemVegetarian(String name) {**

**Iterator pancakeIterator = pancakeHouseMenu.createIterator();**

**if (isVegetarian(name, pancakeIterator)) {**

**return true;**

**}**

**Iterator dinerIterator = dinerMenu.createIterator();**

**if (isVegetarian(name, dinerIterator)) {**

**return true;**

**}**

**return false;**

**}**

**private void printVegetarianMenu(Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**if (menuItem.isVegetarian()) {**

**System.out.print(menuItem.getName());**

**System.out.println("\t\t" + menuItem.getPrice());**

**System.out.println("\t" + menuItem.getDescription());**

**}**

**}**

**}**

**private boolean isVegetarian(String name, Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**if (menuItem.getName().equals(name)) {**

**if (menuItem.isVegetarian()) {**

**return true;**

**}**

**}**

**}**

**return false;**

**}**

**}**

**—-----------------------------------------------------------------------**

* **Transition1**

**1)Menu.java**

**//package headfirst.iterator.transition;**

**import java.util.Iterator;**

**public interface Menu {**

**public Iterator createIterator(); }**

**2)MenuItem.java**

**//package headfirst.iterator.transition;**

**public class MenuItem {**

**String name;**

**String description;**

**boolean vegetarian;**

**double price;**

**public MenuItem(String name,String description,**

**boolean vegetarian,double price)**

**{**

**this.name = name;**

**this.description = description;**

**this.vegetarian = vegetarian;**

**this.price = price;**

**}**

**public String getName() {**

**return name;**

**}**

**public String getDescription() {**

**return description;**

**}**

**public double getPrice() {**

**return price;**

**}**

**public boolean isVegetarian() {**

**return vegetarian;**

**}**

**}**

**3)Waitress.java**

**//package headfirst.iterator.transition;**

**import java.util.\*;**

**public class Waitress {**

**ArrayList menus;**

**public Waitress(ArrayList menus) {**

**this.menus = menus;**

**}**

**public void printMenu() {**

**Iterator menuIterator = menus.iterator();**

**while(menuIterator.hasNext()) {**

**Menu menu = (Menu)menuIterator.next();**

**printMenu(menu.createIterator());**

**}**

**}**

**void printMenu(Iterator iterator) {**

**while (iterator.hasNext()) {**

**MenuItem menuItem = (MenuItem)iterator.next();**

**System.out.print(menuItem.getName() + ", ");**

**System.out.print(menuItem.getPrice() + " -- ");**

**System.out.println(menuItem.getDescription());**

**}**

**}**

**}**

**Observer Pattern**

* **Weather**

**1)CurrentConditionsDisplay.java**

**package headfirst.observer.weather;**

**public class CurrentConditionsDisplay implements Observer,DisplayElement {**

**private float temperature;**

**private float humidity;**

**private Subject weatherData;**

**public CurrentConditionsDisplay(Subject weatherData) {**

**this.weatherData = weatherData;**

**weatherData.registerObserver(this);**

**}**

**public void update(float temperature, float humidity, float pressure) {**

**this.temperature = temperature;**

**this.humidity = humidity;**

**display();**

**}**

**public void display() {**

**System.out.println("Current conditions: " + temperature**

**+ "F degrees and " + humidity + "% humidity");**

**}**

**}**

**2)DisplayElement.java**

**package headfirst.observer.weather;**

**public interface DisplayElement {**

**public void display();**

**}**

**3)ForcastDisplay.java**

**package headfirst.observer.weather;**

**import java.util.\*;**

**public class ForecastDisplay implements Observer, DisplayElement {**

**private float currentPressure = 29.92f;**

**private float lastPressure;**

**private WeatherData weatherData;**

**public ForecastDisplay(WeatherData weatherData) {**

**this.weatherData = weatherData;**

**weatherData.registerObserver(this);**

**}**

**public void update(float temp, float humidity, float pressure) {**

**lastPressure = currentPressure;**

**currentPressure = pressure;**

**display();**

**}**

**public void display() {**

**System.out.print("Forecast: ");**

**if (currentPressure > lastPressure) {**

**System.out.println("Improving weather on the way!");**

**} else if (currentPressure == lastPressure) {**

**System.out.println("More of the same");**

**} else if (currentPressure < lastPressure) {**

**System.out.println("Watch out for cooler, rainy weather");**

**}**

**}**

**}**

**4)HeatIndexDisplay.java**

**package headfirst.observer.weather;**

**public class HeatIndexDisplay implements Observer, DisplayElement {**

**float heatIndex = 0.0f;**

**private WeatherData weatherData;**

**public HeatIndexDisplay(WeatherData weatherData) {**

**this.weatherData = weatherData;**

**weatherData.registerObserver(this);**

**}**

**public void update(float t, float rh, float pressure) {**

**heatIndex = computeHeatIndex(t, rh);**

**display();**

**}**

**private float computeHeatIndex(float t, float rh) {**

**float index = (float)((16.923 + (0.185212 \* t) + (5.37941 \* rh) - (0.100254 \* t \* rh)**

**+ (0.00941695 \* (t \* t)) + (0.00728898 \* (rh \* rh))**

**+ (0.000345372 \* (t \* t \* rh)) - (0.000814971 \* (t \* rh \* rh)) +**

**(0.0000102102 \* (t \* t \* rh \* rh)) - (0.000038646 \* (t \* t \* t)) + (0.0000291583 \***

**(rh \* rh \* rh)) + (0.00000142721 \* (t \* t \* t \* rh)) +**

**(0.000000197483 \* (t \* rh \* rh \* rh)) - (0.0000000218429 \* (t \* t \* t \* rh \* rh)) +**

**0.000000000843296 \* (t \* t \* rh \* rh \* rh)) -**

**(0.0000000000481975 \* (t \* t \* t \* rh \* rh \* rh)));**

**return index;**

**}**

**public void display() {**

**System.out.println("Heat index is " + heatIndex);**

**}**

**}**

**5)Observer.java**

**package headfirst.observer.weather;**

**public interface Observer {**

**public void update(float temp, float humidity, float pressure);**

**}**

**6)StatisticsDisplay.java**

**package headfirst.observer.weather;**

**import java.util.\*;**

**public class StatisticsDisplay implements Observer, DisplayElement {**

**private float maxTemp = 0.0f;**

**private float minTemp = 200;**

**private float tempSum= 0.0f;**

**private int numReadings;**

**private WeatherData weatherData;**

**public StatisticsDisplay(WeatherData weatherData) {**

**this.weatherData = weatherData;**

**weatherData.registerObserver(this);**

**}**

**public void update(float temp, float humidity, float pressure) {**

**tempSum += temp;**

**numReadings++;**

**if (temp > maxTemp) {**

**maxTemp = temp;**

**}**

**if (temp < minTemp) {**

**minTemp = temp;**

**}**

**display();**

**}**

**public void display() {**

**System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)**

**+ "/" + maxTemp + "/" + minTemp);**

**}**

**}**

**7)Subject.java**

**package headfirst.observer.weather;**

**public interface Subject {**

**public void registerObserver(Observer o);**

**public void removeObserver(Observer o);**

**public void notifyObservers();**

**}**

**8)WeatherData.java**

**package headfirst.observer.weather;**

**import java.util.\*;**

**public class WeatherData implements Subject {**

**private ArrayList observers;**

**private float temperature;**

**private float humidity;**

**private float pressure;**

**public WeatherData() {**

**observers = new ArrayList();**

**}**

**public void registerObserver(Observer o) {**

**observers.add(o);**

**}**

**public void removeObserver(Observer o) {**

**int i = observers.indexOf(o);**

**if (i >= 0) {**

**observers.remove(i);**

**}**

**}**

**public void notifyObservers() {**

**for (int i = 0; i < observers.size(); i++) {**

**Observer observer = (Observer)observers.get(i);**

**observer.update(temperature, humidity, pressure);**

**}**

**}**

**public void measurementsChanged() {**

**notifyObservers();**

**}**

**public void setMeasurements(float temperature, float humidity, float pressure) {**

**this.temperature = temperature;**

**this.humidity = humidity;**

**this.pressure = pressure;**

**measurementsChanged();**

**}**

**public float getTemperature() {**

**return temperature;**

**}**

**public float getHumidity() {**

**return humidity;**

**}**

**public float getPressure() {**

**return pressure;**

**}**

**}**

**9)WeatherStation.java**

**package headfirst.observer.weather;**

**import java.util.\*;**

**public class WeatherStation {**

**public static void main(String[] args) {**

**WeatherData weatherData = new WeatherData();**

**CurrentConditionsDisplay currentDisplay =**

**new CurrentConditionsDisplay(weatherData);**

**StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);**

**ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);**

**weatherData.setMeasurements(80, 65, 30.4f);**

**weatherData.setMeasurements(82, 70, 29.2f);**

**weatherData.setMeasurements(78, 90, 29.2f);**

**}**

**}**

**10)WeatherStationHeatIndex.java**

**package headfirst.observer.weather;**

**import java.util.\*;**

**public class WeatherStationHeatIndex {**

**public static void main(String[] args) {**

**WeatherData weatherData = new WeatherData();**

**CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(weatherData);**

**StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);**

**ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);**

**HeatIndexDisplay heatIndexDisplay = new HeatIndexDisplay(weatherData);**

**weatherData.setMeasurements(80, 65, 30.4f);**

**weatherData.setMeasurements(82, 70, 29.2f);**

**weatherData.setMeasurements(78, 90, 29.2f);**

**}**

**}**

**—-----------------------------------------------------------------------**

* **WeatherObservable**

**1)CurrentConditionsDisplay.java**

**package headfirst.observer.weatherobservable;**

**import java.util.Observable;**

**import java.util.Observer;**

**public class CurrentConditionsDisplay implements Observer, DisplayElement {**

**Observable observable;**

**private float temperature;**

**private float humidity;**

**public CurrentConditionsDisplay(Observable observable) {**

**this.observable = observable;**

**observable.addObserver(this);**

**}**

**public void update(Observable obs, Object arg) {**

**if (obs instanceof WeatherData) {**

**WeatherData weatherData = (WeatherData)obs;**

**this.temperature = weatherData.getTemperature();**

**this.humidity = weatherData.getHumidity();**

**display();**

**}**

**}**

**public void display() {**

**System.out.println("Current conditions: " + temperature**

**+ "F degrees and " + humidity + "% humidity");**

**}**

**}**

**2)DisplayElement.java**

**package headfirst.observer.weatherobservable;**

**public interface DisplayElement {**

**public void display();**

**}**

**3)ForcastDisplay.java**

**package headfirst.observer.weatherobservable;**

**import java.util.Observable;**

**import java.util.Observer;**

**public class ForecastDisplay implements Observer, DisplayElement {**

**private float currentPressure = 29.92f;**

**private float lastPressure;**

**public ForecastDisplay(Observable observable) {**

**observable.addObserver(this);**

**}**

**public void update(Observable observable, Object arg) {**

**if (observable instanceof WeatherData) {**

**WeatherData weatherData = (WeatherData)observable;**

**lastPressure = currentPressure;**

**currentPressure = weatherData.getPressure();**

**display();**

**}**

**}**

**public void display() {**

**System.out.print("Forecast: ");**

**if (currentPressure > lastPressure) {**

**System.out.println("Improving weather on the way!");**

**} else if (currentPressure == lastPressure) {**

**System.out.println("More of the same");**

**} else if (currentPressure < lastPressure) {**

**System.out.println("Watch out for cooler, rainy weather");**

**}**

**}**

**}**

**4)HeatIndexDisplay.java**

**package headfirst.observer.weatherobservable;**

**import java.util.Observable;**

**import java.util.Observer;**

**public class HeatIndexDisplay implements Observer, DisplayElement {**

**float heatIndex = 0.0f;**

**public HeatIndexDisplay(Observable observable) {**

**observable.addObserver(this);**

**}**

**public void update(Observable observable, Object arg) {**

**if (observable instanceof WeatherData) {**

**WeatherData weatherData = (WeatherData)observable;**

**float t = weatherData.getTemperature();**

**float rh = weatherData.getHumidity();**

**heatIndex = (float)**

**(**

**(16.923 + (0.185212 \* t)) +**

**(5.37941 \* rh) -**

**(0.100254 \* t \* rh) +**

**(0.00941695 \* (t \* t)) +**

**(0.00728898 \* (rh \* rh)) +**

**(0.000345372 \* (t \* t \* rh)) -**

**(0.000814971 \* (t \* rh \* rh)) +**

**(0.0000102102 \* (t \* t \* rh \* rh)) -**

**(0.000038646 \* (t \* t \* t)) +**

**(0.0000291583 \* (rh \* rh \* rh)) +**

**(0.00000142721 \* (t \* t \* t \* rh)) +**

**(0.000000197483 \* (t \* rh \* rh \* rh)) -**

**(0.0000000218429 \* (t \* t \* t \* rh \* rh)) +**

**(0.000000000843296 \* (t \* t \* rh \* rh \* rh)) -**

**(0.0000000000481975 \* (t \* t \* t \* rh \* rh \*rh)));**

**display();**

**}**

**}**

**public void display() {**

**System.out.println("Heat index is " + heatIndex);**

**}**

**}**

**5)StatisticsDisplay.java**

**package headfirst.observer.weatherobservable;**

**import java.util.Observable;**

**import java.util.Observer;**

**public class StatisticsDisplay implements Observer, DisplayElement {**

**private float maxTemp = 0.0f;**

**private float minTemp = 200;**

**private float tempSum= 0.0f;**

**private int numReadings;**

**public StatisticsDisplay(Observable observable) {**

**observable.addObserver(this);**

**}**

**public void update(Observable observable, Object arg) {**

**if (observable instanceof WeatherData) {**

**WeatherData weatherData = (WeatherData)observable;**

**float temp = weatherData.getTemperature();**

**tempSum += temp;**

**numReadings++;**

**if (temp > maxTemp) {**

**maxTemp = temp;**

**}**

**if (temp < minTemp) {**

**minTemp = temp;**

**}**

**display();**

**}**

**}**

**public void display() {**

**System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)**

**+ "/" + maxTemp + "/" + minTemp);**

**}**

**}**

**6)WeatherData.java**

**package headfirst.observer.weatherobservable;**

**import java.util.Observable;**

**import java.util.Observer;**

**public class WeatherData extends Observable {**

**private float temperature;**

**private float humidity;**

**private float pressure;**

**public WeatherData() { }**

**public void measurementsChanged() {**

**setChanged();**

**notifyObservers();**

**}**

**public void setMeasurements(float temperature, float humidity, float pressure) {**

**this.temperature = temperature;**

**this.humidity = humidity;**

**this.pressure = pressure;**

**measurementsChanged();**

**}**

**public float getTemperature() {**

**return temperature;**

**}**

**public float getHumidity() {**

**return humidity;**

**}**

**public float getPressure() {**

**return pressure;**

**}**

**}**

**7)WeatherStation.java**

**package headfirst.observer.weatherobservable;**

**public class WeatherStation {**

**public static void main(String[] args) {**

**WeatherData weatherData = new WeatherData();**

**CurrentConditionsDisplay currentConditions = new CurrentConditionsDisplay(weatherData);**

**StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);**

**ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);**

**weatherData.setMeasurements(80, 65, 30.4f);**

**weatherData.setMeasurements(82, 70, 29.2f);**

**weatherData.setMeasurements(78, 90, 29.2f);**

**}**

**}**

**8)WeatherStationHeatIndex.java**

**package headfirst.observer.weatherobservable;**

**public class WeatherStationHeatIndex {**

**public static void main(String[] args) {**

**WeatherData weatherData = new WeatherData();**

**CurrentConditionsDisplay currentConditions = new CurrentConditionsDisplay(weatherData);**

**StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);**

**ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);**

**HeatIndexDisplay heatIndexDisplay = new HeatIndexDisplay(weatherData);**

**weatherData.setMeasurements(80, 65, 30.4f);**

**weatherData.setMeasurements(82, 70, 29.2f);**

**weatherData.setMeasurements(78, 90, 29.2f);**

**}**

**}**

**Factory Pattern**

* **PizzaFm**

**1)ChicagoPizzaStore.java**

**package headfirst.factory.pizzafm;**

**public class ChicagoPizzaStore extends PizzaStore {**

**Pizza createPizza(String item) {**

**if (item.equals("cheese")) {**

**return new ChicagoStyleCheesePizza();**

**} else if (item.equals("veggie")) {**

**return new ChicagoStyleVeggiePizza();**

**} else if (item.equals("clam")) {**

**return new ChicagoStyleClamPizza();**

**} else if (item.equals("pepperoni")) {**

**return new ChicagoStylePepperoniPizza();**

**} else return null;**

**}**

**}**

**2)ChicagoStyleCheesePizza.java**

**package headfirst.factory.pizzafm;**

**public class ChicagoStyleCheesePizza extends Pizza {**

**public ChicagoStyleCheesePizza() {**

**name = "Chicago Style Deep Dish Cheese Pizza";**

**dough = "Extra Thick Crust Dough";**

**sauce = "Plum Tomato Sauce";**

**toppings.add("Shredded Mozzarella Cheese");**

**}**

**void cut() {**

**System.out.println("Cutting the pizza into square slices");**

**}**

**}**

**3)ChicagoStyleClamPizza.java**

**package headfirst.factory.pizzafm;**

**public class ChicagoStyleClamPizza extends Pizza {**

**public ChicagoStyleClamPizza() {**

**name = "Chicago Style Clam Pizza";**

**dough = "Extra Thick Crust Dough";**

**sauce = "Plum Tomato Sauce";**

**toppings.add("Shredded Mozzarella Cheese");**

**toppings.add("Frozen Clams from Chesapeake Bay");**

**}**

**void cut() {**

**System.out.println("Cutting the pizza into square slices");**

**}**

**}**

**4)ChicagoStylePepperoniPizza.java**

**package headfirst.factory.pizzafm;**

**public class ChicagoStylePepperoniPizza extends Pizza {**

**public ChicagoStylePepperoniPizza() {**

**name = "Chicago Style Pepperoni Pizza";**

**dough = "Extra Thick Crust Dough";**

**sauce = "Plum Tomato Sauce";**

**toppings.add("Shredded Mozzarella Cheese");**

**toppings.add("Black Olives");**

**toppings.add("Spinach");**

**toppings.add("Eggplant");**

**toppings.add("Sliced Pepperoni");**

**}**

**void cut() {**

**System.out.println("Cutting the pizza into square slices");**

**}**

**}**

**5)ChicagoStyleVeggiePizza.java**

**package headfirst.factory.pizzafm;**

**public class ChicagoStyleVeggiePizza extends Pizza {**

**public ChicagoStyleVeggiePizza() {**

**name = "Chicago Deep Dish Veggie Pizza";**

**dough = "Extra Thick Crust Dough";**

**sauce = "Plum Tomato Sauce";**

**toppings.add("Shredded Mozzarella Cheese");**

**toppings.add("Black Olives");**

**toppings.add("Spinach");**

**toppings.add("Eggplant");**

**}**

**void cut() {**

**System.out.println("Cutting the pizza into square slices");**

**}**

**}**

**6)DependentPizzaStore.java**

**package headfirst.factory.pizzafm;**

**public class DependentPizzaStore {**

**public Pizza createPizza(String style, String type) {**

**Pizza pizza = null;**

**if (style.equals("NY")) {**

**if (type.equals("cheese")) {**

**pizza = new NYStyleCheesePizza();**

**} else if (type.equals("veggie")) {**

**pizza = new NYStyleVeggiePizza();**

**} else if (type.equals("clam")) {**

**pizza = new NYStyleClamPizza();**

**} else if (type.equals("pepperoni")) {**

**pizza = new NYStylePepperoniPizza();**

**}**

**} else if (style.equals("Chicago")) {**

**if (type.equals("cheese")) {**

**pizza = new ChicagoStyleCheesePizza();**

**} else if (type.equals("veggie")) {**

**pizza = new ChicagoStyleVeggiePizza();**

**} else if (type.equals("clam")) {**

**pizza = new ChicagoStyleClamPizza();**

**} else if (type.equals("pepperoni")) {**

**pizza = new ChicagoStylePepperoniPizza();**

**}**

**} else {**

**System.out.println("Error: invalid type of pizza");**

**return null;**

**}**

**pizza.prepare();**

**pizza.bake();**

**pizza.cut();**

**pizza.box();**

**return pizza;**

**}**

**}**

**7)NYPizzaStore.java**

**package headfirst.factory.pizzafm;**

**public class NYPizzaStore extends PizzaStore {**

**Pizza createPizza(String item) {**

**if (item.equals("cheese")) {**

**return new NYStyleCheesePizza();**

**} else if (item.equals("veggie")) {**

**return new NYStyleVeggiePizza();**

**} else if (item.equals("clam")) {**

**return new NYStyleClamPizza();**

**} else if (item.equals("pepperoni")) {**

**return new NYStylePepperoniPizza();**

**} else return null;**

**}**

**}**

**8)NYStyleCheesePizza.java**

**package headfirst.factory.pizzafm;**

**public class NYStyleCheesePizza extends Pizza {**

**public NYStyleCheesePizza() {**

**name = "NY Style Sauce and Cheese Pizza";**

**dough = "Thin Crust Dough";**

**sauce = "Marinara Sauce";**

**toppings.add("Grated Reggiano Cheese");**

**}**

**}**

**9)NYStyleClamPizza.java**

**package headfirst.factory.pizzafm;**

**public class NYStyleClamPizza extends Pizza {**

**public NYStyleClamPizza() {**

**name = "NY Style Clam Pizza";**

**dough = "Thin Crust Dough";**

**sauce = "Marinara Sauce";**

**toppings.add("Grated Reggiano Cheese");**

**toppings.add("Fresh Clams from Long Island Sound");**

**}**

**}**

**10)NYStylePepperoniPizza.java**

**package headfirst.factory.pizzafm;**

**public class NYStylePepperoniPizza extends Pizza {**

**public NYStylePepperoniPizza() {**

**name = "NY Style Pepperoni Pizza";**

**dough = "Thin Crust Dough";**

**sauce = "Marinara Sauce";**

**toppings.add("Grated Reggiano Cheese");**

**toppings.add("Sliced Pepperoni");**

**toppings.add("Garlic");**

**toppings.add("Onion");**

**toppings.add("Mushrooms");**

**toppings.add("Red Pepper");**

**}**

**}**

**11)NYStyleVeggiePizza.java**

**package headfirst.factory.pizzafm;**

**public class NYStyleVeggiePizza extends Pizza {**

**public NYStyleVeggiePizza() {**

**name = "NY Style Veggie Pizza";**

**dough = "Thin Crust Dough";**

**sauce = "Marinara Sauce";**

**toppings.add("Grated Reggiano Cheese");**

**toppings.add("Garlic");**

**toppings.add("Onion");**

**toppings.add("Mushrooms");**

**toppings.add("Red Pepper");**

**}**

**}**

**12)Pizza.java**

**package headfirst.factory.pizzafm;**

**import java.util.ArrayList;**

**public abstract class Pizza {**

**String name;**

**String dough;**

**String sauce;**

**ArrayList toppings = new ArrayList();**

**void prepare() {**

**System.out.println("Preparing " + name);**

**System.out.println("Tossing dough...");**

**System.out.println("Adding sauce...");**

**System.out.println("Adding toppings: ");**

**for (int i = 0; i < toppings.size(); i++) {**

**System.out.println(" " + toppings.get(i));**

**}**

**}**

**void bake() {**

**System.out.println("Bake for 25 minutes at 350");**

**}**

**void cut() {**

**System.out.println("Cutting the pizza into diagonal slices");**

**}**

**void box() {**

**System.out.println("Place pizza in official PizzaStore box");**

**}**

**public String getName() {**

**return name;**

**}**

**public String toString() {**

**StringBuffer display = new StringBuffer();**

**display.append("---- " + name + " ----\n");**

**display.append(dough + "\n");**

**display.append(sauce + "\n");**

**for (int i = 0; i < toppings.size(); i++) {**

**display.append((String )toppings.get(i) + "\n");**

**}**

**return display.toString();**

**}**

**}**

**13)PizzaStore.java**

**package headfirst.factory.pizzafm;**

**public abstract class PizzaStore {**

**abstract Pizza createPizza(String item);**

**public Pizza orderPizza(String type) {**

**Pizza pizza = createPizza(type);**

**System.out.println("--- Making a " + pizza.getName() + " ---");**

**pizza.prepare();**

**pizza.bake();**

**pizza.cut();**

**pizza.box();**

**return pizza;**

**}**

**}**

**14)PizzaTestDrive.java**

**package headfirst.factory.pizzafm;**

**public class PizzaTestDrive {**

**public static void main(String[] args) {**

**PizzaStore nyStore = new NYPizzaStore();**

**PizzaStore chicagoStore = new ChicagoPizzaStore();**

**Pizza pizza = nyStore.orderPizza("cheese");**

**System.out.println("Ethan ordered a " + pizza.getName() + "\n");**

**pizza = chicagoStore.orderPizza("cheese");**

**System.out.println("Joel ordered a " + pizza.getName() + "\n");**

**pizza = nyStore.orderPizza("clam");**

**System.out.println("Ethan ordered a " + pizza.getName() + "\n");**

**pizza = chicagoStore.orderPizza("clam");**

**System.out.println("Joel ordered a " + pizza.getName() + "\n");**

**pizza = nyStore.orderPizza("pepperoni");**

**System.out.println("Ethan ordered a " + pizza.getName() + "\n");**

**pizza = chicagoStore.orderPizza("pepperoni");**

**System.out.println("Joel ordered a " + pizza.getName() + "\n");**

**pizza = nyStore.orderPizza("veggie");**

**System.out.println("Ethan ordered a " + pizza.getName() + "\n");**

**pizza = chicagoStore.orderPizza("veggie");**

**System.out.println("Joel ordered a " + pizza.getName() + "\n");**

**}**

**}**

**—---------------------------------------------------------------------**

* **Pizzas**

**1)CheesePizza.java**

**package headfirst.factory.pizzas;**

**public class CheesePizza extends Pizza {**

**public CheesePizza() {**

**name = "Cheese Pizza";**

**dough = "Regular Crust";**

**sauce = "Marinara Pizza Sauce";**

**toppings.add("Fresh Mozzarella");**

**toppings.add("Parmesan");**

**}**

**}**

**2)ClamPizza.java**

**package headfirst.factory.pizzas;**

**public class ClamPizza extends Pizza {**

**public ClamPizza() {**

**name = "Clam Pizza";**

**dough = "Thin crust";**

**sauce = "White garlic sauce";**

**toppings.add("Clams");**

**toppings.add("Grated parmesan cheese");**

**}**

**}**

**3)PepperoniPizza.java**

**package headfirst.factory.pizzas;**

**public class PepperoniPizza extends Pizza {**

**public PepperoniPizza() {**

**name = "Pepperoni Pizza";**

**dough = "Crust";**

**sauce = "Marinara sauce";**

**toppings.add("Sliced Pepperoni");**

**toppings.add("Sliced Onion");**

**toppings.add("Grated parmesan cheese");**

**}**

**}**

**4)Pizza.java**

**package headfirst.factory.pizzas;**

**import java.util.ArrayList;**

**abstract public class Pizza {**

**String name;**

**String dough;**

**String sauce;**

**ArrayList toppings = new ArrayList();**

**public String getName() {**

**return name;**

**}**

**public void prepare() {**

**System.out.println("Preparing " + name);**

**}**

**public void bake() {**

**System.out.println("Baking " + name);**

**}**

**public void cut() {**

**System.out.println("Cutting " + name);**

**}**

**public void box() {**

**System.out.println("Boxing " + name);**

**}**

**public String toString() {**

**// code to display pizza name and ingredients**

**StringBuffer display = new StringBuffer();**

**display.append("---- " + name + " ----\n");**

**display.append(dough + "\n");**

**display.append(sauce + "\n");**

**for (int i = 0; i < toppings.size(); i++) {**

**display.append((String )toppings.get(i) + "\n");**

**}**

**return display.toString();**

**}**

**}**

**5)PizzaStore.java**

**package headfirst.factory.pizzas;**

**public class PizzaStore {**

**SimplePizzaFactory factory;**

**public PizzaStore(SimplePizzaFactory factory) {**

**this.factory = factory;**

**}**

**public Pizza orderPizza(String type) {**

**Pizza pizza;**

**pizza = factory.createPizza(type);**

**pizza.prepare();**

**pizza.bake();**

**pizza.cut();**

**pizza.box();**

**return pizza;**

**}**

**}**

**6)PizzaTestDrive.java**

**package headfirst.factory.pizzas;**

**public class PizzaTestDrive {**

**public static void main(String[] args) {**

**SimplePizzaFactory factory = new SimplePizzaFactory();**

**PizzaStore store = new PizzaStore(factory);**

**Pizza pizza = store.orderPizza("cheese");**

**System.out.println("We ordered a " + pizza.getName() + "\n");**

**pizza = store.orderPizza("veggie");**

**System.out.println("We ordered a " + pizza.getName() + "\n");**

**}**

**}**

**7)SimplePizzaFactory.java**

**package headfirst.factory.pizzas;**

**public class SimplePizzaFactory {**

**public Pizza createPizza(String type) {**

**Pizza pizza = null;**

**if (type.equals("cheese")) {**

**pizza = new CheesePizza();**

**} else if (type.equals("pepperoni")) {**

**pizza = new PepperoniPizza();**

**} else if (type.equals("clam")) {**

**pizza = new ClamPizza();**

**} else if (type.equals("veggie")) {**

**pizza = new VeggiePizza();**

**}**

**return pizza;**

**}**

**}**

**8)VeggiePizza.java**

**package headfirst.factory.pizzas;**

**public class VeggiePizza extends Pizza {**

**public VeggiePizza() {**

**name = "Veggie Pizza";**

**dough = "Crust";**

**sauce = "Marinara sauce";**

**toppings.add("Shredded mozzarella");**

**toppings.add("Grated parmesan");**

**toppings.add("Diced onion");**

**toppings.add("Sliced mushrooms");**

**toppings.add("Sliced red pepper");**

**toppings.add("Sliced black olives");**

**}**

**}**

**Facade Pattern**

* **HomeTheatre**

**1)Amplifier.java**

**package headfirst.facade.hometheater;**

**public class Amplifier {**

**String description;**

**Tuner tuner;**

**DvdPlayer dvd;**

**CdPlayer cd;**

**public Amplifier(String description) {**

**this.description = description;**

**}**

**public void on() {**

**System.out.println(description + " on");**

**}**

**public void off() {**

**System.out.println(description + " off");**

**}**

**public void setStereoSound() {**

**System.out.println(description + " stereo mode on");**

**}**

**public void setSurroundSound() {**

**System.out.println(description + " surround sound on (5 speakers, 1 subwoofer)");**

**}**

**public void setVolume(int level) {**

**System.out.println(description + " setting volume to " + level);**

**}**

**public void setTuner(Tuner tuner) {**

**System.out.println(description + " setting tuner to " + dvd);**

**this.tuner = tuner;**

**}**

**public void setDvd(DvdPlayer dvd) {**

**System.out.println(description + " setting DVD player to " + dvd);**

**this.dvd = dvd;**

**}**

**public void setCd(CdPlayer cd) {**

**System.out.println(description + " setting CD player to " + cd);**

**this.cd = cd;**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**2)CdPlayer.java**

**package headfirst.facade.hometheater;**

**public class CdPlayer {**

**String description;**

**int currentTrack;**

**Amplifier amplifier;**

**String title;**

**public CdPlayer(String description, Amplifier amplifier) {**

**this.description = description;**

**this.amplifier = amplifier;**

**}**

**public void on() {**

**System.out.println(description + " on");**

**}**

**public void off() {**

**System.out.println(description + " off");**

**}**

**public void eject() {**

**title = null;**

**System.out.println(description + " eject");**

**}**

**public void play(String title) {**

**this.title = title;**

**currentTrack = 0;**

**System.out.println(description + " playing \"" + title + "\"");**

**}**

**public void play(int track) {**

**if (title == null) {**

**System.out.println(description + " can't play track " + currentTrack + ", no cd inserted");**

**} else {**

**currentTrack = track;**

**System.out.println(description + " playing track " + currentTrack);**

**}**

**}**

**public void stop() {**

**currentTrack = 0;**

**System.out.println(description + " stopped");**

**}**

**public void pause() {**

**System.out.println(description + " paused \"" + title + "\"");**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**3)DvdPlayer.java**

**package headfirst.facade.hometheater;**

**public class DvdPlayer {**

**String description;**

**int currentTrack;**

**Amplifier amplifier;**

**String movie;**

**public DvdPlayer(String description, Amplifier amplifier) {**

**this.description = description;**

**this.amplifier = amplifier;**

**}**

**public void on() {**

**System.out.println(description + " on");**

**}**

**public void off() {**

**System.out.println(description + " off");**

**}**

**public void eject() {**

**movie = null;**

**System.out.println(description + " eject");**

**}**

**public void play(String movie) {**

**this.movie = movie;**

**currentTrack = 0;**

**System.out.println(description + " playing \"" + movie + "\"");**

**}**

**public void play(int track) {**

**if (movie == null) {**

**System.out.println(description + " can't play track " + track + " no dvd inserted");**

**} else {**

**currentTrack = track;**

**System.out.println(description + " playing track " + currentTrack + " of \"" + movie + "\"");**

**}**

**}**

**public void stop() {**

**currentTrack = 0;**

**System.out.println(description + " stopped \"" + movie + "\"");**

**}**

**public void pause() {**

**System.out.println(description + " paused \"" + movie + "\"");**

**}**

**public void setTwoChannelAudio() {**

**System.out.println(description + " set two channel audio");**

**}**

**public void setSurroundAudio() {**

**System.out.println(description + " set surround audio");**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**4)HomeTheatreFacade.java**

**package headfirst.facade.hometheater;**

**public class HomeTheaterFacade {**

**Amplifier amp;**

**Tuner tuner;**

**DvdPlayer dvd;**

**CdPlayer cd;**

**Projector projector;**

**TheaterLights lights;**

**Screen screen;**

**PopcornPopper popper;**

**public HomeTheaterFacade(Amplifier amp,**

**Tuner tuner,**

**DvdPlayer dvd,**

**CdPlayer cd,**

**Projector projector,**

**Screen screen,**

**TheaterLights lights,**

**PopcornPopper popper) {**

**this.amp = amp;**

**this.tuner = tuner;**

**this.dvd = dvd;**

**this.cd = cd;**

**this.projector = projector;**

**this.screen = screen;**

**this.lights = lights;**

**this.popper = popper;**

**}**

**public void watchMovie(String movie) {**

**System.out.println("Get ready to watch a movie...");**

**popper.on();**

**popper.pop();**

**lights.dim(10);**

**screen.down();**

**projector.on();**

**projector.wideScreenMode();**

**amp.on();**

**amp.setDvd(dvd);**

**amp.setSurroundSound();**

**amp.setVolume(5);**

**dvd.on();**

**dvd.play(movie);**

**}**

**public void endMovie() {**

**System.out.println("Shutting movie theater down...");**

**popper.off();**

**lights.on();**

**screen.up();**

**projector.off();**

**amp.off();**

**dvd.stop();**

**dvd.eject();**

**dvd.off();**

**}**

**public void listenToCd(String cdTitle) {**

**System.out.println("Get ready for an audiopile experence...");**

**lights.on();**

**amp.on();**

**amp.setVolume(5);**

**amp.setCd(cd);**

**amp.setStereoSound();**

**cd.on();**

**cd.play(cdTitle);**

**}**

**public void endCd() {**

**System.out.println("Shutting down CD...");**

**amp.off();**

**amp.setCd(cd);**

**cd.eject();**

**cd.off();**

**}**

**public void listenToRadio(double frequency) {**

**System.out.println("Tuning in the airwaves...");**

**tuner.on();**

**tuner.setFrequency(frequency);**

**amp.on();**

**amp.setVolume(5);**

**amp.setTuner(tuner);**

**}**

**public void endRadio() {**

**System.out.println("Shutting down the tuner...");**

**tuner.off();**

**amp.off();**

**}**

**}**

**5)HomeTheatreTestDrive.java**

**package headfirst.facade.hometheater;**

**public class HomeTheaterTestDrive {**

**public static void main(String[] args) {**

**Amplifier amp = new Amplifier("Top-O-Line Amplifier");**

**Tuner tuner = new Tuner("Top-O-Line AM/FM Tuner", amp);**

**DvdPlayer dvd = new DvdPlayer("Top-O-Line DVD Player", amp);**

**CdPlayer cd = new CdPlayer("Top-O-Line CD Player", amp);**

**Projector projector = new Projector("Top-O-Line Projector", dvd);**

**TheaterLights lights = new TheaterLights("Theater Ceiling Lights");**

**Screen screen = new Screen("Theater Screen");**

**PopcornPopper popper = new PopcornPopper("Popcorn Popper");**

**HomeTheaterFacade homeTheater =**

**new HomeTheaterFacade(amp, tuner, dvd, cd,**

**projector, screen, lights, popper);**

**homeTheater.watchMovie("Raiders of the Lost Ark");**

**homeTheater.endMovie();**

**}**

**}**

**6)PopcornPopper.java**

**package headfirst.facade.hometheater;**

**public class PopcornPopper {**

**String description;**

**public PopcornPopper(String description) {**

**this.description = description;**

**}**

**public void on() {**

**System.out.println(description + " on");**

**}**

**public void off() {**

**System.out.println(description + " off");**

**}**

**public void pop() {**

**System.out.println(description + " popping popcorn!");**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**7)Projector.java**

**package headfirst.facade.hometheater;**

**public class Projector {**

**String description;**

**DvdPlayer dvdPlayer;**

**public Projector(String description, DvdPlayer dvdPlayer) {**

**this.description = description;**

**this.dvdPlayer = dvdPlayer;**

**}**

**public void on() {**

**System.out.println(description + " on");**

**}**

**public void off() {**

**System.out.println(description + " off");**

**}**

**public void wideScreenMode() {**

**System.out.println(description + " in widescreen mode (16x9 aspect ratio)");**

**}**

**public void tvMode() {**

**System.out.println(description + " in tv mode (4x3 aspect ratio)");**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**8)Screen.java**

**package headfirst.facade.hometheater;**

**public class Screen {**

**String description;**

**public Screen(String description) {**

**this.description = description;**

**}**

**public void up() {**

**System.out.println(description + " going up");**

**}**

**public void down() {**

**System.out.println(description + " going down");**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**9)TheatreLights.java**

**package headfirst.facade.hometheater;**

**public class TheaterLights {**

**String description;**

**public TheaterLights(String description) {**

**this.description = description;**

**}**

**public void on() {**

**System.out.println(description + " on");**

**}**

**public void off() {**

**System.out.println(description + " off");**

**}**

**public void dim(int level) {**

**System.out.println(description + " dimming to " + level + "%");**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**10)Tuner.java**

**package headfirst.facade.hometheater;**

**public class Tuner {**

**String description;**

**Amplifier amplifier;**

**double frequency;**

**public Tuner(String description, Amplifier amplifier) {**

**this.description = description;**

**}**

**public void on() {**

**System.out.println(description + " on");**

**}**

**public void off() {**

**System.out.println(description + " off");**

**}**

**public void setFrequency(double frequency) {**

**System.out.println(description + " setting frequency to " + frequency);**

**this.frequency = frequency;**

**}**

**public void setAm() {**

**System.out.println(description + " setting AM mode");**

**}**

**public void setFm() {**

**System.out.println(description + " setting FM mode");**

**}**

**public String toString() {**

**return description;**

**}**

**}**

**Decorator Pattern**

* **Io**

**1)InputTest.java**

**package headfirst.decorator.io;**

**import java.io.\*;**

**public class InputTest {**

**public static void main(String[] args) throws IOException {**

**int c;**

**try {**

**InputStream in =**

**new LowerCaseInputStream(**

**new BufferedInputStream(**

**new FileInputStream("test.txt")));**

**while((c = in.read()) >= 0) {**

**System.out.print((char)c);**

**}**

**in.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**2)LowerCaseInputStream.java**

**package headfirst.decorator.io;**

**import java.io.\*;**

**public class LowerCaseInputStream extends FilterInputStream {**

**public LowerCaseInputStream(InputStream in) {**

**super(in);**

**}**

**public int read() throws IOException {**

**int c = super.read();**

**return (c == -1 ? c : Character.toLowerCase((char)c));**

**}**

**public int read(byte[] b, int offset, int len) throws IOException {**

**int result = super.read(b, offset, len);**

**for (int i = offset; i < offset+result; i++) {**

**b[i] = (byte)Character.toLowerCase((char)b[i]);**

**}**

**return result;**

**}**

**}**

**—----------------------------------------------------------------------**

* **StarBuzz**

**1)Beverage.java**

**package headfirst.decorator.starbuzz;**

**public abstract class Beverage {**

**String description = "Unknown Beverage";**

**public String getDescription() {**

**return description;**

**}**

**public abstract double cost();**

**}**

**2)CondimentDecorator.java**

**package headfirst.decorator.starbuzz;**

**public abstract class CondimentDecorator extends Beverage {**

**public abstract String getDescription();**

**}**

**3)DarkRoast.java**

**package headfirst.decorator.starbuzz;**

**public class DarkRoast extends Beverage {**

**public DarkRoast() {**

**description = "Dark Roast Coffee";**

**}**

**public double cost() {**

**return .99;**

**}**

**}**

**4)Decaf.java**

**package headfirst.decorator.starbuzz;**

**public class Decaf extends Beverage {**

**public Decaf() {**

**description = "Decaf Coffee";**

**}**

**public double cost() {**

**return 1.05;**

**}**

**}**

**5)Espresso.java**

**package headfirst.decorator.starbuzz;**

**public class Espresso extends Beverage {**

**public Espresso() {**

**description = "Espresso";**

**}**

**public double cost() {**

**return 1.99;**

**}**

**}**

**6)HouseBlend.java**

**package headfirst.decorator.starbuzz;**

**public class HouseBlend extends Beverage {**

**public HouseBlend() {**

**description = "House Blend Coffee";**

**}**

**public double cost() {**

**return .89;**

**}**

**}**

**7)Milk.java**

**package headfirst.decorator.starbuzz;**

**public class Milk extends CondimentDecorator {**

**Beverage beverage;**

**public Milk(Beverage beverage) {**

**this.beverage = beverage;**

**}**

**public String getDescription() {**

**return beverage.getDescription() + ", Milk";**

**}**

**public double cost() {**

**return .10 + beverage.cost();**

**}**

**}**

**8)Mocha.java**

**package headfirst.decorator.starbuzz;**

**public class Mocha extends CondimentDecorator {**

**Beverage beverage;**

**public Mocha(Beverage beverage) {**

**this.beverage = beverage;**

**}**

**public String getDescription() {**

**return beverage.getDescription() + ", Mocha";**

**}**

**public double cost() {**

**return .20 + beverage.cost();**

**}**

**}**

**9)Soy.java**

**package headfirst.decorator.starbuzz;**

**public class Soy extends CondimentDecorator {**

**Beverage beverage;**

**public Soy(Beverage beverage) {**

**this.beverage = beverage;**

**}**

**public String getDescription() {**

**return beverage.getDescription() + ", Soy";**

**}**

**public double cost() {**

**return .15 + beverage.cost();**

**}**

**}**

**10)StarbuzzCoffee.java**

**package headfirst.decorator.starbuzz;**

**public class StarbuzzCoffee {**

**public static void main(String args[]) {**

**Beverage beverage = new Espresso();**

**System.out.println(beverage.getDescription() + " $" + beverage.cost());**

**Beverage beverage2 = new DarkRoast();**

**beverage2 = new Mocha(beverage2);**

**beverage2 = new Mocha(beverage2);**

**beverage2 = new Whip(beverage2);**

**System.out.println(beverage2.getDescription() + " $" +beverage2.cost());**

**Beverage beverage3 = new HouseBlend();**

**beverage3 = new Soy(beverage3);**

**beverage3 = new Mocha(beverage3);**

**beverage3 = new Whip(beverage3);**

**System.out.println(beverage3.getDescription() + " $" +beverage3.cost());**

**}**

**}**

**11)Whip.java**

**package headfirst.decorator.starbuzz;**

**public class Whip extends CondimentDecorator {**

**Beverage beverage;**

**public Whip(Beverage beverage) {**

**this.beverage = beverage;**

**}**

**public String getDescription() {**

**return beverage.getDescription() + ", Whip";**

**}**

**public double cost() {**

**return .10 + beverage.cost();**

**}**

**}**

**Command Pattern**

* **Remote**

**1)CeilingFan.java**

**package headfirst.command.remote;**

**public class CeilingFan {**

**String location = "";**

**int level;**

**public static final int HIGH = 2;**

**public static final int MEDIUM = 1;**

**public static final int LOW = 0;**

**public CeilingFan(String location) {**

**this.location = location;**

**}**

**public void high() {**

**// turns the ceiling fan on to high**

**level = HIGH;**

**System.out.println(location + " ceiling fan is on high");**

**}**

**public void medium() {**

**// turns the ceiling fan on to medium**

**level = MEDIUM;**

**System.out.println(location + " ceiling fan is on medium");**

**}**

**public void low() {**

**// turns the ceiling fan on to low**

**level = LOW;**

**System.out.println(location + " ceiling fan is on low");**

**}**

**public void off() {**

**// turns the ceiling fan off**

**level = 0;**

**System.out.println(location + " ceiling fan is off");**

**}**

**public int getSpeed() {**

**return level;**

**}**

**}**

**2)CeilingFanOffCommand.java**

**package headfirst.command.remote;**

**public class CeilingFanOffCommand implements Command {**

**CeilingFan ceilingFan;**

**public CeilingFanOffCommand(CeilingFan ceilingFan) {**

**this.ceilingFan = ceilingFan;**

**}**

**public void execute() {**

**ceilingFan.off();**

**}**

**}**

**3)CeilingFanOnCommand.java**

**package headfirst.command.remote;**

**public class CeilingFanOnCommand implements Command {**

**CeilingFan ceilingFan;**

**public CeilingFanOnCommand(CeilingFan ceilingFan) {**

**this.ceilingFan = ceilingFan;**

**}**

**public void execute() {**

**ceilingFan.high();**

**}**

**}**

**4)Command.java**

**package headfirst.command.remote;**

**public interface Command {**

**public void execute();**

**}**

**5)GarageDoor.java**

**package headfirst.command.remote;**

**public class GarageDoor {**

**String location;**

**public GarageDoor(String location) {**

**this.location = location;**

**}**

**public void up() {**

**System.out.println(location + " garage Door is Up");**

**}**

**public void down() {**

**System.out.println(location + " garage Door is Down");**

**}**

**public void stop() {**

**System.out.println(location + " garage Door is Stopped");**

**}**

**public void lightOn() {**

**System.out.println(location + " garage light is on");**

**}**

**public void lightOff() {**

**System.out.println(location + " garage light is off");**

**}**

**}**

**6)GarageDoorDownCommand.java**

**package headfirst.command.remote;**

**public class GarageDoorDownCommand implements Command {**

**GarageDoor garageDoor;**

**public GarageDoorDownCommand(GarageDoor garageDoor) {**

**this.garageDoor = garageDoor;**

**}**

**public void execute() {**

**garageDoor.up();**

**}**

**}**

**7)GarageDoorUpCommand.java**

**package headfirst.command.remote;**

**public class GarageDoorUpCommand implements Command {**

**GarageDoor garageDoor;**

**public GarageDoorUpCommand(GarageDoor garageDoor) {**

**this.garageDoor = garageDoor;**

**}**

**public void execute() {**

**garageDoor.up();**

**}**

**}**

**8)Hottub.java**

**package headfirst.command.remote;**

**public class Hottub {**

**boolean on;**

**int temperature;**

**public Hottub() {**

**}**

**public void on() {**

**on = true;**

**}**

**public void off() {**

**on = false;**

**}**

**public void bubblesOn() {**

**if (on) {**

**System.out.println("Hottub is bubbling!");**

**}**

**}**

**public void bubblesOff() {**

**if (on) {**

**System.out.println("Hottub is not bubbling");**

**}**

**}**

**public void jetsOn() {**

**if (on) {**

**System.out.println("Hottub jets are on");**

**}**

**}**

**public void jetsOff() {**

**if (on) {**

**System.out.println("Hottub jets are off");**

**}**

**}**

**public void setTemperature(int temperature) {**

**this.temperature = temperature;**

**}**

**public void heat() {**

**temperature = 105;**

**System.out.println("Hottub is heating to a steaming 105 degrees");**

**}**

**public void cool() {**

**temperature = 98;**

**System.out.println("Hottub is cooling to 98 degrees");**

**}**

**}**

**9)HottubOffCommand.java**

**package headfirst.command.remote;**

**public class HottubOffCommand implements Command {**

**Hottub hottub;**

**public HottubOffCommand(Hottub hottub) {**

**this.hottub = hottub;**

**}**

**public void execute() {**

**hottub.cool();**

**hottub.off();**

**}**

**}**

**10)HottubOnCommand.java**

**package headfirst.command.remote;**

**public class HottubOnCommand implements Command {**

**Hottub hottub;**

**public HottubOnCommand(Hottub hottub) {**

**this.hottub = hottub;**

**}**

**public void execute() {**

**hottub.on();**

**hottub.heat();**

**hottub.bubblesOn();**

**}**

**}**

**11)Light.java**

**package headfirst.command.remote;**

**public class Light {**

**String location = "";**

**public Light(String location) {**

**this.location = location;**

**}**

**public void on() {**

**System.out.println(location + " light is on");**

**}**

**public void off() {**

**System.out.println(location + " light is off");**

**}**

**}**

**12)LightOffCommand.java**

**package headfirst.command.remote;**

**public class LightOffCommand implements Command {**

**Light light;**

**public LightOffCommand(Light light) {**

**this.light = light;**

**}**

**public void execute() {**

**light.off();**

**}**

**}**

**13)LightOnCommand.java**

**package headfirst.command.remote;**

**public class LightOnCommand implements Command {**

**Light light;**

**public LightOnCommand(Light light) {**

**this.light = light;**

**}**

**public void execute() {**

**light.on();**

**}**

**}**

**14)LivingroomLightOffCommand.java**

**package headfirst.command.remote;**

**public class LivingroomLightOffCommand implements Command {**

**Light light;**

**public LivingroomLightOffCommand(Light light) {**

**this.light = light;**

**}**

**public void execute() {**

**light.off();**

**}**

**}**

**15)LivingroomLightOnCommand.java**

**package headfirst.command.remote;**

**public class LivingroomLightOnCommand implements Command {**

**Light light;**

**public LivingroomLightOnCommand(Light light) {**

**this.light = light;**

**}**

**public void execute() {**

**light.on();**

**}**

**}**

**16)NoCommand.java**

**package headfirst.command.remote;**

**public class NoCommand implements Command {**

**public void execute() { }**

**}**

**17)RemoteControl.java**

**package headfirst.command.remote;**

**import java.util.\*;**

**//**

**// This is the invoker**

**//**

**public class RemoteControl {**

**Command[] onCommands;**

**Command[] offCommands;**

**public RemoteControl() {**

**onCommands = new Command[7];**

**offCommands = new Command[7];**

**Command noCommand = new NoCommand();**

**for (int i = 0; i < 7; i++) {**

**onCommands[i] = noCommand;**

**offCommands[i] = noCommand;**

**}**

**}**

**public void setCommand(int slot, Command onCommand, Command offCommand) {**

**onCommands[slot] = onCommand;**

**offCommands[slot] = offCommand;**

**}**

**public void onButtonWasPushed(int slot) {**

**onCommands[slot].execute();**

**}**

**public void offButtonWasPushed(int slot) {**

**offCommands[slot].execute();**

**}**

**public String toString() {**

**StringBuffer stringBuff = new StringBuffer();**

**stringBuff.append("\n------ Remote Control -------\n");**

**for (int i = 0; i < onCommands.length; i++) {**

**stringBuff.append("[slot " + i + "] " + onCommands[i].getClass().getName()**

**+ " " + offCommands[i].getClass().getName() + "\n");**

**}**

**return stringBuff.toString();**

**}**

**}**

**18)RemoteLoader.java**

**package headfirst.command.remote;**

**public class RemoteLoader {**

**public static void main(String[] args) {**

**RemoteControl remoteControl = new RemoteControl();**

**Light livingRoomLight = new Light("Living Room");**

**Light kitchenLight = new Light("Kitchen");**

**CeilingFan ceilingFan= new CeilingFan("Living Room");**

**GarageDoor garageDoor = new GarageDoor("");**

**Stereo stereo = new Stereo("Living Room");**

**LightOnCommand livingRoomLightOn =new LightOnCommand(livingRoomLight);**

**LightOffCommand livingRoomLightOff =new LightOffCommand(livingRoomLight);**

**LightOnCommand kitchenLightOn =new LightOnCommand(kitchenLight);**

**LightOffCommand kitchenLightOff =new LightOffCommand(kitchenLight);**

**CeilingFanOnCommand ceilingFanOn =new CeilingFanOnCommand(ceilingFan);**

**CeilingFanOffCommand ceilingFanOff =new CeilingFanOffCommand(ceilingFan);**

**GarageDoorUpCommand garageDoorUp =new GarageDoorUpCommand(garageDoor);**

**GarageDoorDownCommand garageDoorDown =new GarageDoorDownCommand(garageDoor);**

**StereoOnWithCDCommand stereoOnWithCD =new StereoOnWithCDCommand(stereo);**

**StereoOffCommand stereoOff =new StereoOffCommand(stereo);**

**remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);**

**remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);**

**remoteControl.setCommand(2, ceilingFanOn, ceilingFanOff);**

**remoteControl.setCommand(3, stereoOnWithCD, stereoOff);**

**System.out.println(remoteControl);**

**remoteControl.onButtonWasPushed(0);**

**remoteControl.offButtonWasPushed(0);**

**remoteControl.onButtonWasPushed(1);**

**remoteControl.offButtonWasPushed(1);**

**remoteControl.onButtonWasPushed(2);**

**remoteControl.offButtonWasPushed(2);**

**remoteControl.onButtonWasPushed(3);**

**remoteControl.offButtonWasPushed(3);**

**}**

**}**

**19)Stereo.java**

**package headfirst.command.remote;**

**public class Stereo {**

**String location;**

**public Stereo(String location) {**

**this.location = location;**

**}**

**public void on() {**

**System.out.println(location + " stereo is on");**

**}**

**public void off() {**

**System.out.println(location + " stereo is off");**

**}**

**public void setCD() {**

**System.out.println(location + " stereo is set for CD input");**

**}**

**public void setDVD() {**

**System.out.println(location + " stereo is set for DVD input");**

**}**

**public void setRadio() {**

**System.out.println(location + " stereo is set for Radio");**

**}**

**public void setVolume(int volume) {**

**// code to set the volume**

**// valid range: 1-11 (after all 11 is better than 10, right?)**

**System.out.println(location + " Stereo volume set to " + volume);**

**}**

**}**

**20)StereoOffCommand.java**

**package headfirst.command.remote;**

**public class StereoOffCommand implements Command {**

**Stereo stereo;**

**public StereoOffCommand(Stereo stereo) {**

**this.stereo = stereo;**

**}**

**public void execute() {**

**stereo.off();**

**}**

**}**

**21)StereoOnWithCDCommand.java**

**package headfirst.command.remote;**

**public class StereoOnWithCDCommand implements Command {**

**Stereo stereo;**

**public StereoOnWithCDCommand(Stereo stereo) {**

**this.stereo = stereo;**

**}**

**public void execute() {**

**stereo.on();**

**stereo.setCD();**

**stereo.setVolume(11);**

**}**

**}**

**22)TV.java**

**package headfirst.command.remote;**

**public class TV {**

**String location;**

**int channel;**

**public TV(String location) {**

**this.location = location;**

**}**

**public void on() {**

**System.out.println("TV is on");**

**}**

**public void off() {**

**System.out.println("TV is off");**

**}**

**public void setInputChannel() {**

**this.channel = 3;**

**System.out.println("Channel is set for VCR");**

**}**

**}**

**—----------------------------------------------------------------------**

* **Undo**

**1)CeilingFan.java**

**package headfirst.command.undo;**

**public class CeilingFan {**

**public static final int HIGH = 3;**

**public static final int MEDIUM = 2;**

**public static final int LOW = 1;**

**public static final int OFF = 0;**

**String location;**

**int speed;**

**public CeilingFan(String location) {**

**this.location = location;**

**speed = OFF;**

**}**

**public void high() {**

**speed = HIGH;**

**System.out.println(location + " ceiling fan is on high");**

**}**

**public void medium() {**

**speed = MEDIUM;**

**System.out.println(location + " ceiling fan is on medium");**

**}**

**public void low() {**

**speed = LOW;**

**System.out.println(location + " ceiling fan is on low");**

**}**

**public void off() {**

**speed = OFF;**

**System.out.println(location + " ceiling fan is off");**

**}**

**public int getSpeed() {**

**return speed;**

**}**

**}**

**2)CeilingFanHighCommand.java**

**package headfirst.command.undo;**

**public class CeilingFanHighCommand implements Command {**

**CeilingFan ceilingFan;**

**int prevSpeed;**

**public CeilingFanHighCommand(CeilingFan ceilingFan) {**

**this.ceilingFan = ceilingFan;**

**}**

**public void execute() {**

**prevSpeed = ceilingFan.getSpeed();**

**ceilingFan.high();**

**}**

**public void undo() {**

**if (prevSpeed == CeilingFan.HIGH) {**

**ceilingFan.high();**

**} else if (prevSpeed == CeilingFan.MEDIUM) {**

**ceilingFan.medium();**

**} else if (prevSpeed == CeilingFan.LOW) {**

**ceilingFan.low();**

**} else if (prevSpeed == CeilingFan.OFF) {**

**ceilingFan.off();**

**}**

**}**

**}**

**3)CeilingFanLowCommand.java**

**package headfirst.command.undo;**

**public class CeilingFanLowCommand implements Command {**

**CeilingFan ceilingFan;**

**int prevSpeed;**

**public CeilingFanLowCommand(CeilingFan ceilingFan) {**

**this.ceilingFan = ceilingFan;**

**}**

**public void execute() {**

**prevSpeed = ceilingFan.getSpeed();**

**ceilingFan.low();**

**}**

**public void undo() {**

**if (prevSpeed == CeilingFan.HIGH) {**

**ceilingFan.high();**

**} else if (prevSpeed == CeilingFan.MEDIUM) {**

**ceilingFan.medium();**

**} else if (prevSpeed == CeilingFan.LOW) {**

**ceilingFan.low();**

**} else if (prevSpeed == CeilingFan.OFF) {**

**ceilingFan.off();**

**}**

**}**

**}**

**4)CeilingFanMediumCommand.java**

**package headfirst.command.undo;**

**public class CeilingFanMediumCommand implements Command {**

**CeilingFan ceilingFan;**

**int prevSpeed;**

**public CeilingFanMediumCommand(CeilingFan ceilingFan) {**

**this.ceilingFan = ceilingFan;**

**}**

**public void execute() {**

**prevSpeed = ceilingFan.getSpeed();**

**ceilingFan.medium();**

**}**

**public void undo() {**

**if (prevSpeed == CeilingFan.HIGH) {**

**ceilingFan.high();**

**} else if (prevSpeed == CeilingFan.MEDIUM) {**

**ceilingFan.medium();**

**} else if (prevSpeed == CeilingFan.LOW) {**

**ceilingFan.low();**

**} else if (prevSpeed == CeilingFan.OFF) {**

**ceilingFan.off();**

**}**

**}**

**}**

**5)CeilingFanOffCommand.java**

**package headfirst.command.undo;**

**public class CeilingFanOffCommand implements Command {**

**CeilingFan ceilingFan;**

**int prevSpeed;**

**public CeilingFanOffCommand(CeilingFan ceilingFan) {**

**this.ceilingFan = ceilingFan;**

**}**

**public void execute() {**

**prevSpeed = ceilingFan.getSpeed();**

**ceilingFan.off();**

**}**

**public void undo() {**

**if (prevSpeed == CeilingFan.HIGH) {**

**ceilingFan.high();**

**} else if (prevSpeed == CeilingFan.MEDIUM) {**

**ceilingFan.medium();**

**} else if (prevSpeed == CeilingFan.LOW) {**

**ceilingFan.low();**

**} else if (prevSpeed == CeilingFan.OFF) {**

**ceilingFan.off();**

**}**

**}**

**}**

**6)Command.java**

**package headfirst.command.undo;**

**public interface Command {**

**public void execute();**

**public void undo();**

**}**

**7)DimmerLightOffCommand.java**

**package headfirst.command.undo;**

**public class DimmerLightOffCommand implements Command {**

**Light light;**

**int prevLevel;**

**public DimmerLightOffCommand(Light light) {**

**this.light = light;**

**prevLevel = 100;**

**}**

**public void execute() {**

**prevLevel = light.getLevel();**

**light.off();**

**}**

**public void undo() {**

**light.dim(prevLevel);**

**}**

**}**

**8)DimmerLightOnCommand.java**

**package headfirst.command.undo;**

**public class DimmerLightOnCommand implements Command {**

**Light light;**

**int prevLevel;**

**public DimmerLightOnCommand(Light light) {**

**this.light = light;**

**}**

**public void execute() {**

**prevLevel = light.getLevel();**

**light.dim(75);**

**}**

**public void undo() {**

**light.dim(prevLevel);**

**}**

**}**

**9)Light.java**

**package headfirst.command.undo;**

**public class Light {**

**String location;**

**int level;**

**public Light(String location) {**

**this.location = location;**

**}**

**public void on() {**

**level = 100;**

**System.out.println("Light is on");**

**}**

**public void off() {**

**level = 0;**

**System.out.println("Light is off");**

**}**

**public void dim(int level) {**

**this.level = level;**

**if (level == 0) {**

**off();**

**}**

**else {**

**System.out.println("Light is dimmed to " + level + "%");**

**}**

**}**

**public int getLevel() {**

**return level;**

**}**

**}**

**10)LightOffCommand.java**

**package headfirst.command.undo;**

**public class LightOffCommand implements Command {**

**Light light;**

**int level;**

**public LightOffCommand(Light light) {**

**this.light = light;**

**}**

**public void execute() {**

**level = light.getLevel();**

**light.off();**

**}**

**public void undo() {**

**light.dim(level);**

**}**

**}**

**11)LightOnCommand.java**

**package headfirst.command.undo;**

**public class LightOnCommand implements Command {**

**Light light;**

**int level;**

**public LightOnCommand(Light light) {**

**this.light = light;**

**}**

**public void execute() {**

**level = light.getLevel();**

**light.on();**

**}**

**public void undo() {**

**light.dim(level);**

**}**

**}**

**12)NoCommand.java**

**package headfirst.command.undo;**

**public class NoCommand implements Command {**

**public void execute() { }**

**public void undo() { }**

**}**

**13)RemoteControlWithUndo.java**

**package headfirst.command.undo;**

**import java.util.\*;**

**//**

**// This is the invoker**

**//**

**public class RemoteControlWithUndo {**

**Command[] onCommands;**

**Command[] offCommands;**

**Command undoCommand;**

**public RemoteControlWithUndo() {**

**onCommands = new Command[7];**

**offCommands = new Command[7];**

**Command noCommand = new NoCommand();**

**for(int i=0;i<7;i++) {**

**onCommands[i] = noCommand;**

**offCommands[i] = noCommand;**

**}**

**undoCommand = noCommand;**

**}**

**public void setCommand(int slot, Command onCommand, Command offCommand) {**

**onCommands[slot] = onCommand;**

**offCommands[slot] = offCommand;**

**}**

**public void onButtonWasPushed(int slot) {**

**onCommands[slot].execute();**

**undoCommand = onCommands[slot];**

**}**

**public void offButtonWasPushed(int slot) {**

**offCommands[slot].execute();**

**undoCommand = offCommands[slot];**

**}**

**public void undoButtonWasPushed() {**

**undoCommand.undo();**

**}**

**public String toString() {**

**StringBuffer stringBuff = new StringBuffer();**

**stringBuff.append("\n------ Remote Control -------\n");**

**for (int i = 0; i < onCommands.length; i++) {**

**stringBuff.append("[slot " + i + "] " + onCommands[i].getClass().getName()**

**+ " " + offCommands[i].getClass().getName() + "\n");**

**}**

**stringBuff.append("[undo] " + undoCommand.getClass().getName() + "\n");**

**return stringBuff.toString();**

**}**

**}**

**14)RemoteLoader.java**

**package headfirst.command.undo;**

**public class RemoteLoader {**

**public static void main(String[] args) {**

**RemoteControlWithUndo remoteControl = new RemoteControlWithUndo();**

**Light livingRoomLight = new Light("Living Room");**

**LightOnCommand livingRoomLightOn =new LightOnCommand(livingRoomLight);**

**LightOffCommand livingRoomLightOff =new LightOffCommand(livingRoomLight);**

**remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);**

**remoteControl.onButtonWasPushed(0);**

**remoteControl.offButtonWasPushed(0);**

**System.out.println(remoteControl);**

**remoteControl.undoButtonWasPushed();**

**remoteControl.offButtonWasPushed(0);**

**remoteControl.onButtonWasPushed(0);**

**System.out.println(remoteControl);**

**remoteControl.undoButtonWasPushed();**

**CeilingFan ceilingFan = new CeilingFan("Living Room");**

**CeilingFanMediumCommand ceilingFanMedium =new CeilingFanMediumCommand(ceilingFan);**

**CeilingFanHighCommand ceilingFanHigh =new CeilingFanHighCommand(ceilingFan);**

**CeilingFanOffCommand ceilingFanOff =new CeilingFanOffCommand(ceilingFan);**

**remoteControl.setCommand(0, ceilingFanMedium, ceilingFanOff);**

**remoteControl.setCommand(1, ceilingFanHigh, ceilingFanOff);**

**remoteControl.onButtonWasPushed(0);**

**remoteControl.offButtonWasPushed(0);**

**System.out.println(remoteControl);**

**remoteControl.undoButtonWasPushed();**

**remoteControl.onButtonWasPushed(1);**

**System.out.println(remoteControl);**

**remoteControl.undoButtonWasPushed();**

**}**

**}**

**Adapter Pattern**

* **Ducks**

**1)Duck.java**

**package headfirst.adapter.ducks;**

**public interface Duck {**

**public void quack();**

**public void fly();**

**}**

**2)DuckAdapter.java**

**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();**

**}**

**}**

**}**

**3)DuckTestDrive.java**

**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();**

**}**

**}**

**4)MallardDuck.java**

**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");**

**}**

**}**

**5)Turkey.java**

**package headfirst.adapter.ducks;**

**public interface Turkey {**

**public void gobble();**

**public void fly();**

**}**

**6)TurkeyAdapter.java**

**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();**

**}**

**}**

**}**

**7)TurkeyTestDrive.java**

**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();**

**}**

**}**

**}**

**8)WildTurkey.java**

**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");**

**}**

**}**

* **Iterenum**

**1)EI.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());**

**}**

**}**

**}**

**2)EnumerationIterator.java**

**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();**

**}**

**}**

**3)EnumerationIteratorTestDrive.java**

**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());**

**}**

**}**

**}**

**4)IteratorEnumeration.java**

**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();**

**}**

**}**

**5)IteratorEnumerationTestDrive.java**

**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());**

**}**

**}**

**}**