Module Name: Contemporary Topics in Software Engineering

Module Code : ITP4507

Assigment

Student Name: Li Chi Fung

Student Number: 20011201

Sections 1 page 3

Section 2 page4-6

Section 3 page 7-8

Section4 page 9-10

Section 5 page 11

Section 6 page 13-31

Assignment Report In addition to the system development，you are required to write up a Short Report covers following sections：

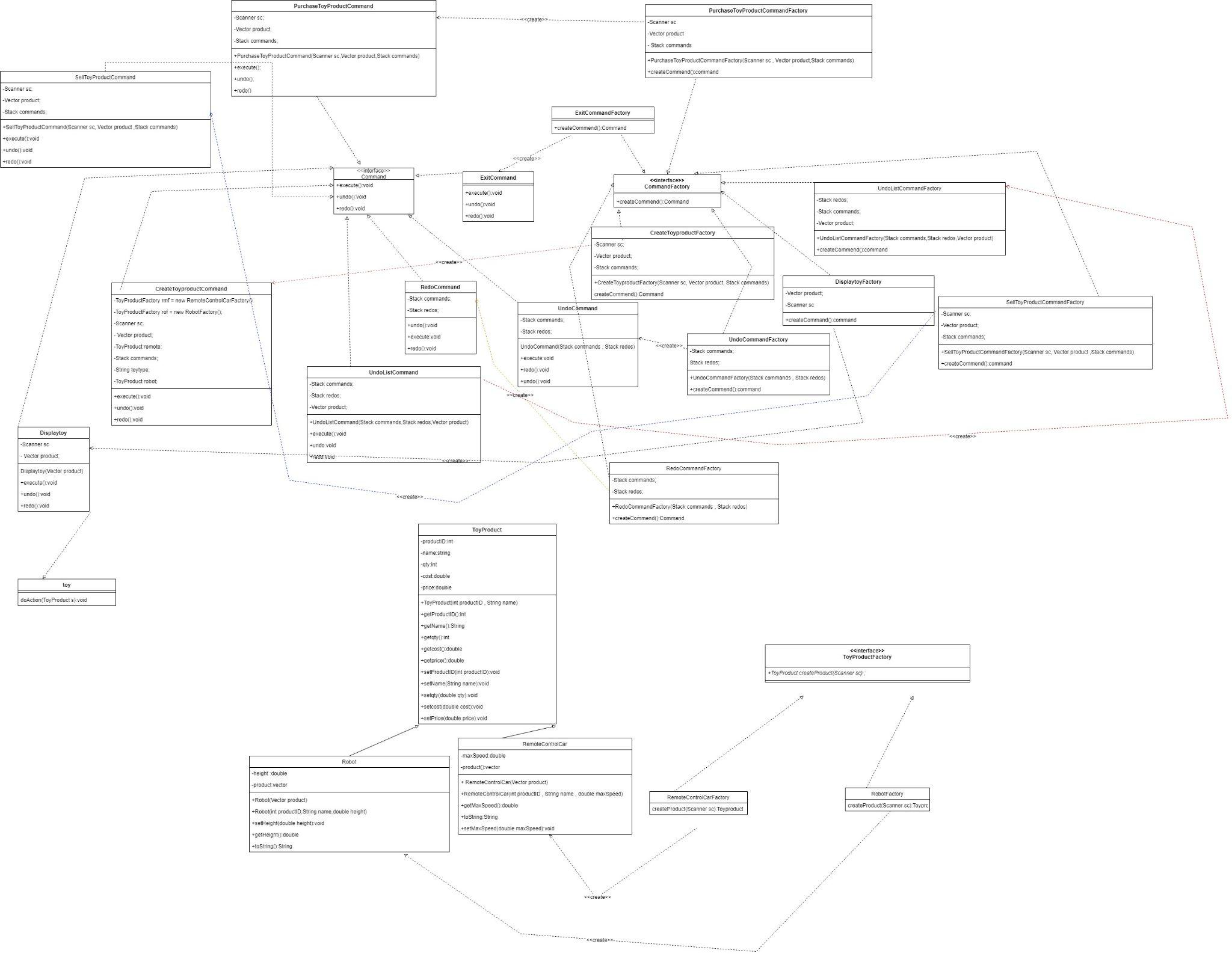
1. Assumptions regarding the problem context

First, the system is able to display all the products and change the value in the list. For example price, quality and cost.

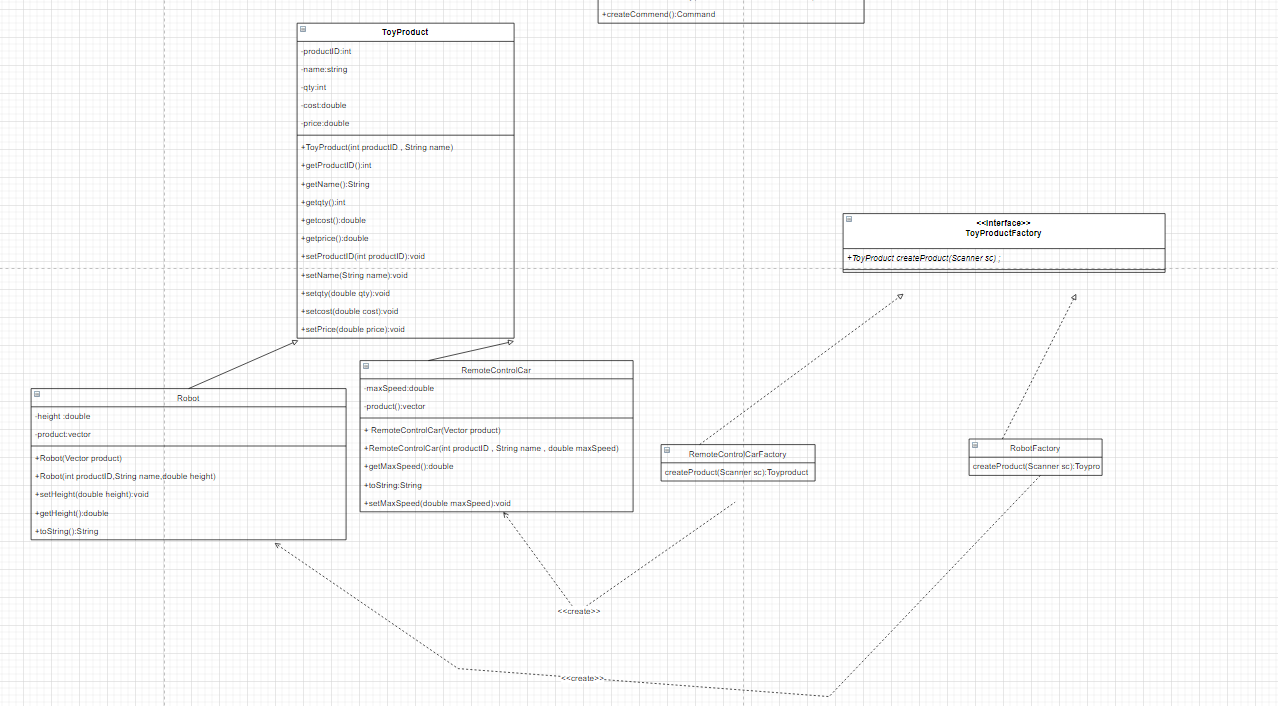
Second, the other function of the system is checking the unavailable value. For example, the product\_ID has been used before. Also, the incorrect data type would be unavailable. For example, the data type of product\_ID is a number, if the user enters a string into product\_ID, the result will be an error.

Third, before the user uses the undo command, there are several things that have to be completed first, such as create toy , sell toy and purchase toy command. Otherwise, it will result in nothing to undo. For the redo command, the undo command must be finished first.

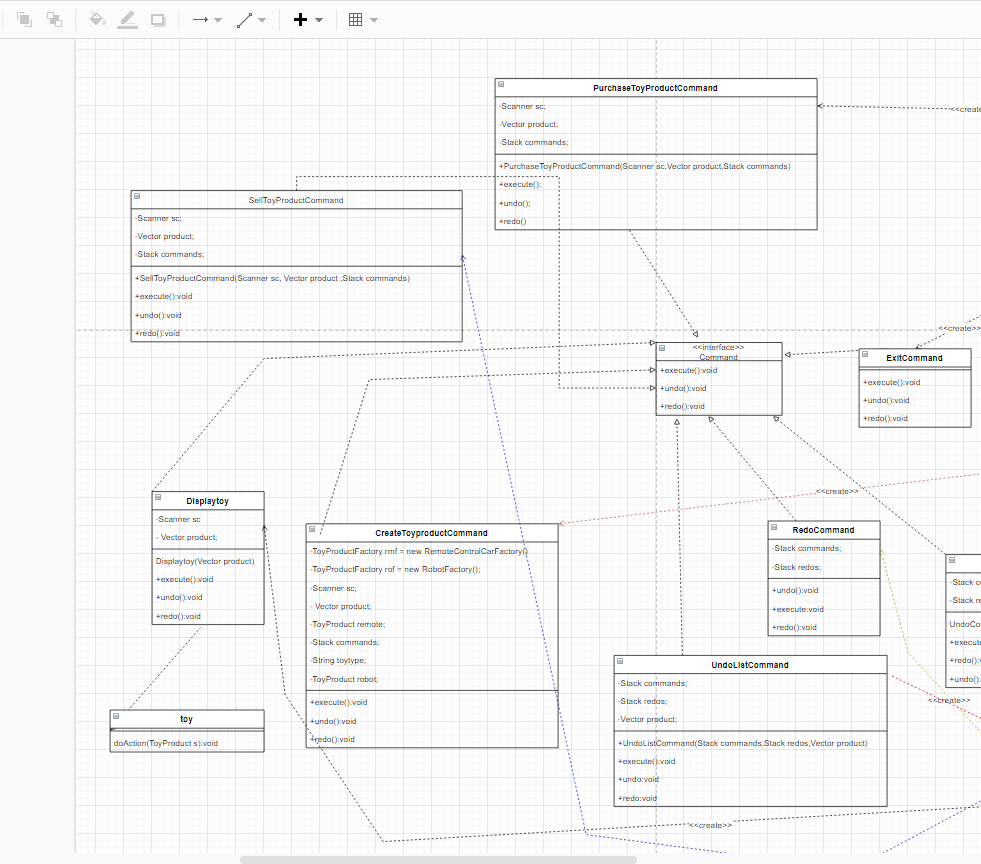
2. Application design with class diagram



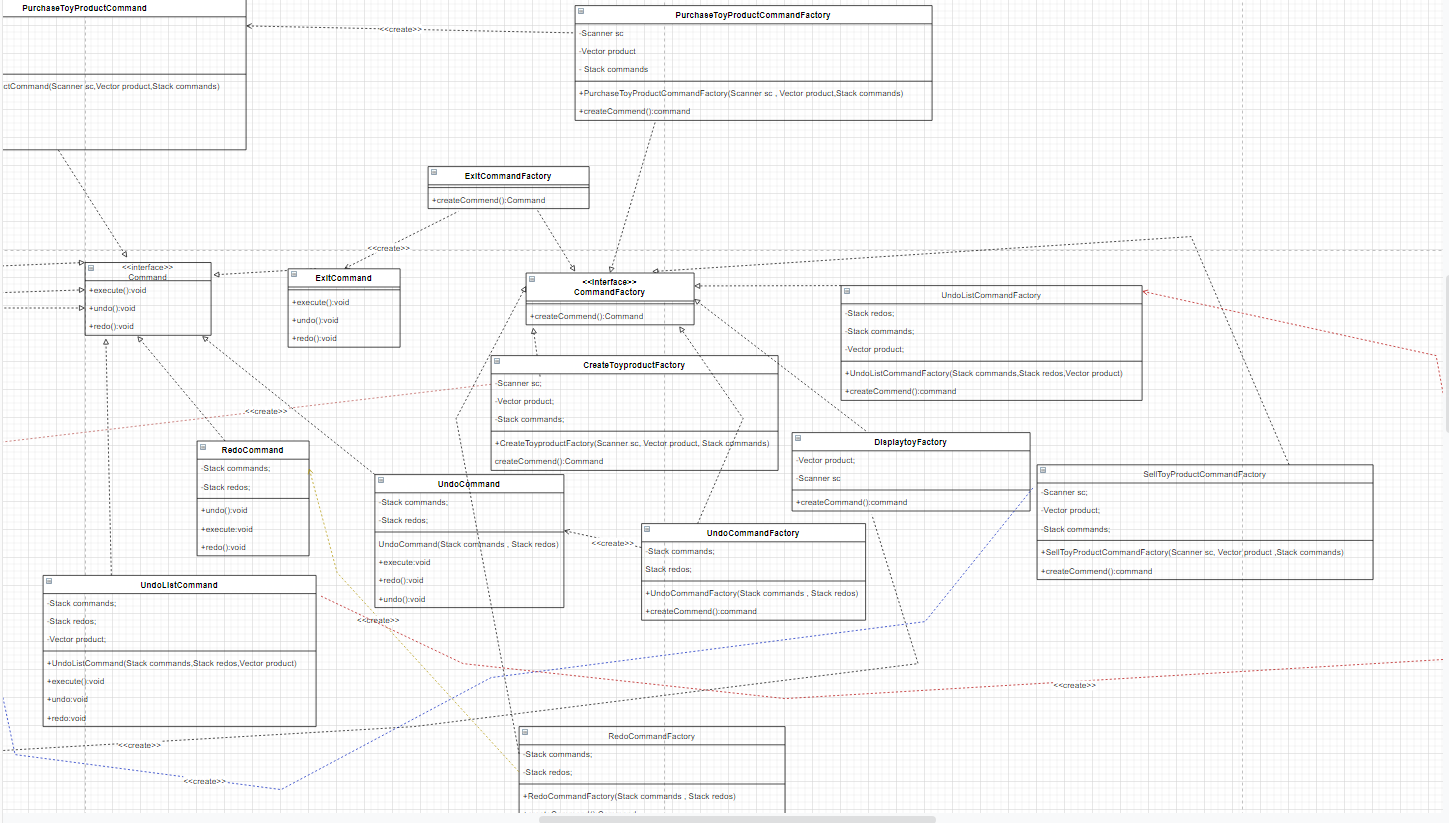
Toy produce:



Command:



CommandFactory:



Memento

一張含有 文字, 地圖, 室內 的圖片

自動產生的描述

I have a jpg in file, it can see more clearly.

3. Discussion and explanation on each of the design patterns applied to the application

Command:

Use to do a execute all the command

CommandFactory:

Use to create a command

CreateToyproduct:

Use to create a toy product

CreateToyproductFactory:

Let the main class to call the class with open close

Displaytoy:

Use to define the displaytoy function

DisplaytoyFactory:

Let the main to call the class with open close

ExitCommand:

Use to define the exit function

ExitCommandFactory:

Let the main call the class with open close

Main:

enter the command and execute the command

PurchaseToyProductCommand:

Use to purchase a product

CreateToyproductFactory:

Let the main call the class with open close

RemoteControlCar:

Use to define the Remote Control Car value

RemoteControlCarFactory:

Use to create a Remote Control Car product

Robot:

Use to define the Robot value

RobotFactory:

Use to create a robot product

SellToyProductCommand:

Use to Sell a product function

SellToyProductCommandFactory:

Let the main call the class with open close

toy:

Ues to print out all the toy product

ToyProduct:

Use to define the toy product value

ToyProductFactory:

Let the create product function to call this class to create a product

UndoListCommand:

Use to print out all the undo list and redo list

UndoListCommandFactory:

Let the main call the class with open close

4. User Guide

說明書

* User can input a command(c,d,p,s,u,r,l,x)

User input c command:

The user will be asked which type of toy (ro/rc) should be created first. If the user inputs ro, it will create a Robot product. If the user inputs rc, it will create a Remote Control Car.

* User input d command:

The user will be asked which product(\*/ product\_ID) should be displayed.

If the user input a \*,

It will display all the toy products such as price, name, ID, quantity, other info, cost

If the user inputs a product\_ID, It will display the information of the product such as price, name, id, quantity, height/Max speed, cost.

* User input p command:

The user will be asked which one they want to change(input a product\_ID)

also, they need to input the quantity of what they want to purchase and what cost is purchase.

In the end, it will output a result which they are changed.

* User input s command:

The system will request the user to input the product\_id for what quantity should to sold and the price selling for.

When the user finishes inputting all the values, it will output a changed result.

* User input u command:

The system undoes a command which already does a command before. If the user didn't anything before, the system will get a reminder. For example, users input u command when just start to begin using the system, the system will output a remind (Nothing to undo).

* User input r command:

The system will redo one command which is the system already undo a command before. If the user didn't undo a command before but the user inputs r command the system will output a remind(nothing to redo).

* User input l command:

The system will list out all the undo command lists and redo command lists. If the list is empty, it will display empty. Otherwise, it will display the list of redo and undo lists.

* User input x command:

The system will exit.

5. Test Plan and Test Cases

寫test既野

Test Plan:

Checking purchase product if nothing in toy product already create.

一張含有 文字 的圖片

自動產生的描述

Checking sell product if nothing in toy product already have

一張含有 文字 的圖片

自動產生的描述

Doing undo command need to do something before

一張含有 文字 的圖片

自動產生的描述

Doing redo command need to udo first

一張含有 文字 的圖片

自動產生的描述

the undo and redo is empty than show that is empty

一張含有 文字 的圖片

自動產生的描述

6. Well documented Source Code

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

**public** **class** Caretaker {

**private** Stack undoList;

**private** Stack redoList;

**public** Caretaker(){

        undoList = new Stack();

        redoList = new Stack();

    }

**public** **void** saveToyProduct(ToyProduct toyProduct){

        undoList.push(new Memento(toyProduct));

    }

**public** **void** undo(){

        if(undoList.size()!=0){

            Memento m =(Memento) undoList.pop();

            redoList.push(m);

            m.restore();

            System.out.println(m.getToyproduct().getCost() + m.getToyproduct().getPrice()+ m.getToyproduct().getQty());

        }else{

            System.out.println("nothing to undo");

        }

    }

**public** **void** redo(){

        if(redoList.size()!=0){

            Memento m = (Memento) redoList.pop();

            undoList.push(m);

            m.restore();

            System.out.println( m.getToyproduct().getCost() + m.getToyproduct().getPrice()+ m.getToyproduct().getQty() );

        }else{

            System.out.println("nothing to redo");

        }

    }

}

**public** **interface** Command {

**public** **void** execute();

**public** **void** undo();

**public** **void** redo();

}

**public** **interface** CommandFactory {

**public** Command createCommend();

}

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

**public** **class** CreateToyproductCommand **implements** Command{

**private** **static** ToyProductFactory rmf = new RemoteControlCarFactory();

**private** **static** ToyProductFactory rof = new RobotFactory();

**private** Scanner sc;

**private** Vector product;

**private** ToyProduct robot;

**private** ToyProduct remote;

**private** Stack commands;

    String toytype;

**public** CreateToyproductCommand(Scanner sc, Vector product, Stack commands){

        this.sc = sc;

        this.product=product;

        this.commands =commands;

        remote = null;

        robot =null;

    }

**public** **void** execute(){

        System.out.println("Enter toy type (ro=Robot/rc=Remote Control Car): ");

        toytype = sc.next();

        if(toytype.equals("rc")){

            ToyProduct c = rmf.createProduct(sc);

            product.add(c);

        }else{

            ToyProduct c = rof.createProduct(sc);

            product.add(c);

        }

    }

**public** **void** undo(){

        if(product.isEmpty()){

            System.out.println("Noting to undo");

        }else{

            if(toytype.equals("rc")){

                product.remove(remote);

            }else{

                product.remove(robot);

            }

        }

    }

**public** **void** redo(){

        if(product.isEmpty()){

            System.out.println("Noting to undo");

        }else{

            if(toytype.equals("rc")){

                product.add(remote);

            }else{

                product.add(robot);

            }

        }

    }

}

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

**public** **class** CreateToyproductFactory **implements** CommandFactory{

**private** Scanner sc;

**private** Vector product;

**private** Stack commands;

**public** CreateToyproductFactory(Scanner sc, Vector product, Stack commands){

        this.sc=sc;

        this.product=product;

        this.commands=commands;

    }

**public** Command createCommend(){

            Command c = new CreateToyproductCommand(sc,product,commands);

            commands.push(c);

            return c;

    }

}

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

**public** **class** Displaytoy **implements** Command{

**private** Scanner sc = new Scanner(System.in);

**private** Vector product;

**public** Displaytoy(Vector product){

        this.product=product;

    }

**public** **void** execute(){

        System.out.println("Enter ID(\* to display all)");

        String id = sc.next();

        System.out.println("Toy product information");

        if(id.equals("\*")){

            System.out.println("ID  "+"   Name  "+"      Quantity"+"    Other Info"+"  Cost "+" Price");

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

                ToyProduct t = (ToyProduct) product.elementAt(i);

                toy.doAction(t);

            }

        }else{

            Enumeration enu = product.elements();

            Robot t = (Robot) enu.nextElement();

            System.out.println("ID: "+t.getProductID());

            System.out.println("Name: "+t.getName());

            System.out.println("Quantity "+t.getQty());

            System.out.println("Height(cm): "+ t.getHeight());

            System.out.println("Cost($): "+t.getCost());

            System.out.println("Price($): "+t.getPrice());

        }

    }

**public** **void** undo(){

    }

**public** **void** redo(){

    }

}

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

**public** **class** DisplaytoyFactory **implements** CommandFactory{

**private** Scanner sc = new Scanner(System.in);

**private** Vector product;

**public** DisplaytoyFactory(Vector product){

        this.product=product;

    }

**public** Command createCommend(){

        Command c = new Displaytoy(product);

        return c;

    }

}

**public** **class** ExitCommand **implements** Command{

**public** **void** execute(){

        System.exit(0);

    }

**public** **void** undo(){

    }

**public** **void** redo(){

    }

}

**public** **class** ExitCommandFactory **implements** CommandFactory{

**public** Command createCommend(){

        return new ExitCommand();

    }

}

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

**public** **class** Main {

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

        Scanner sc = new Scanner(System.in);

        Vector product = new Vector();              *// store product*

        Stack commands = new Stack();               *// store undo*

        Stack redos = new Stack();                  *// for the redo*

        CommandFactory [] f = new CommandFactory[999];

        f['c'] = new CreateToyproductFactory(sc, product,commands);

        f['x'] = new ExitCommandFactory();

        f['d'] = new DisplaytoyFactory(product);

        f['p'] = new PurchaseToyProductCommandFactory(sc, product, commands);

        f['s'] = new SellToyProductCommandFactory(sc, product, commands);

        f['u'] = new UndoCommandFactory(commands, redos);

        f['r'] = new RedoCommandFactory(commands, redos);

        f['l'] = new UndoListCommandFactory(commands, redos, product);

     while(true){

        System.out.println();

        System.out.println("Toy Inventory Management System (TIMS)");

        System.out.println("Please enter command: [c | d | p | s | u | r | l | x]");

        System.out.println("c = create toy, d = display toy, p = purchase toy, s = sell toy, ");

        System.out.println("u = undo, r = redo, l = list undo/redo, x = exit system");

        System.out.println();

        String commend= sc.next(); *//reads string.*

        System.out.println();

**char** commande;

        commande =  commend.charAt(0);

        Command c = f[commande].createCommend();

        c.execute();

        switch(commend){

           case "c":

            redos.clear();

        }

     } *// end white*

    }

}

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

**public** **class** Memento {

**private** ToyProduct toyproduct;

**private** **double** cost;

**private** **double** price;

**private** **int** qty;

**public** Memento(ToyProduct toyProduct){

        this.toyproduct = toyProduct;

        this.cost = toyProduct.getCost();

        this.price=toyProduct.getPrice();

        this.qty=toyProduct.getQty();

    }

**public** **void** restore(){

*//toyProduct.commands = this.commands;*

        toyproduct.setCost(this.cost);

        toyproduct.setPrice(this.price);

        toyproduct.setQty(this.qty);

    }

**public** ToyProduct getToyproduct(){

        return this.toyproduct;

    }

}

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

**public** **class** PurchaseToyProductCommand **implements** Command{

**private** Scanner sc;

**private** Vector product;

**private** Stack commands;

**public** PurchaseToyProductCommand(Scanner sc,Vector product,Stack commands){

        this.product = product;

        this.sc=sc;

        this.commands=commands;

    }

**public** **void** execute(){

        System.out.println("Enter code:");

**int** code = sc.nextInt();

        System.out.println("Quantity to purchased:");

**int** qty =sc.nextInt();

        System.out.println("Purchasing cost:");

**double** cost = sc.nextDouble();

        if(product.size() >0){

            if(code == 1001){

                Enumeration enu = product.elements();

                ToyProduct tp = (ToyProduct)enu.nextElement();

                tp.setQty(qty);

                tp.setCost(cost);

                System.out.println("Purchased"+tp.getQty()+" Box of "+tp.getName()+".Current quantity is $"+tp.getCost()+".Price is $"+tp.getPrice());

            }else{

                Enumeration enu = product.elements();

                ToyProduct tp = (ToyProduct)enu.nextElement();

                tp.setQty(qty);

                tp.setCost(cost);

                System.out.println("Purchased"+tp.getQty()+" Box of "+tp.getName()+".Current quantity is $"+tp.getCost()+".Price is $"+tp.getPrice());

            }

        }else{

            System.out.println("noting to purchase");

        }

    }

**public** **void** undo(){

    }

**public** **void** redo(){

    }

}

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

**public** **class** PurchaseToyProductCommandFactory **implements** CommandFactory{

**private** Scanner sc;

**private** Vector product;

**private** Stack commands;

**public** PurchaseToyProductCommandFactory(Scanner sc , Vector product,Stack commands){

        this.sc =sc;

        this.product =product;

        this.commands =commands;

    }

**public** Command createCommend(){

        Command c = new PurchaseToyProductCommand(sc, product,commands);

        commands.push(c);

        return c;

    }

}

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

**public** **class** RedoCommand **implements** Command{

**private** Stack commands;

**private** Stack redos;

**public** RedoCommand(Stack commands , Stack redos){

        this.commands = commands;

        this.redos = redos;

    }

**public** **void** execute(){

        if(redos.empty()){

            System.out.println("Noting can undo");

        }else{

            Command c = (Command) redos.pop();

            c.redo();

            commands.push(c);

            System.out.println("Redo completed");

        }

    }

**public** **void** undo(){

    }

**public** **void** redo(){

    }

}

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

**public** **class** RedoCommandFactory **implements** CommandFactory{

**private** Stack commands;

**private** Stack redos;

**public** RedoCommandFactory(Stack commands , Stack redos){

        this.commands=commands;

        this.redos=redos;

    }

**public** Command createCommend(){

        return new RedoCommand(commands, redos);

    }

}

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

**public** **class** RemoteControlCar **extends** ToyProduct{

**private** **double** maxSpeed;

**private** Vector product;

**public** RemoteControlCar(Vector product){

        this.product =product;

    }

**public** RemoteControlCar(**int** productID , String name , **double** maxSpeed){

        super(productID , name);

        this.maxSpeed = maxSpeed;

    }

**public** **double** getMaxSpeed() {

        return maxSpeed;

    }

**public** **void** setMaxSpeed(**double** maxSpeed) {

        this.maxSpeed = maxSpeed;

    }

**public** String toString() {

*// TODO Auto-generated method stub*

        return super.toString()+"         "+getMaxSpeed()+"km/hr"+"       "+super.getCost()+"     "+super.getPrice();

    }

}

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

**public** **class** RemoteControlCarFactory **implements** ToyProductFactory{

*//private Scanner sc;*

    @**Override**

**public** ToyProduct createProduct(Scanner sc){

            System.out.println("Enter id");

**int** id = sc.nextInt();

            System.out.println("Enter name"+sc.nextLine());

            String name = sc.nextLine();

            System.out.println("maximum speed(km/hr)");

**double** speed = sc.nextDouble();

            System.out.println(id+","+name+","+speed);

            System.out.println("New toy product record created.");

            return new RemoteControlCar(id, name, speed);

    }

}

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

**public** **class** Robot **extends** ToyProduct{

**private** **double** height;

**private** Vector product;

**public** Robot(Vector product){

        this.product =product;

    }

**public** Robot(**int** productID,String name,**double** height){

        super(productID , name);

        this.height=height;

    }

**public** **double** getHeight() {

        return height;

    }

**public** **void** setHeight(**double** height) {

        this.height = height;

    }

    @**Override**

**public** String toString() {

*// TODO Auto-generated method stub*

        return super.toString()+"         "+getHeight()+"cm"+"       "+super.getCost()+"     "+super.getPrice();

    }

}

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

**public** **class** RobotFactory **implements** ToyProductFactory{

*//private Scanner sc;*

**public** ToyProduct createProduct(Scanner sc){

                System.out.println("Enter id");

**int** id = sc.nextInt();

                System.out.println("Enter name"+sc.nextLine());

                String name = sc.nextLine();

                System.out.println("Enter height(cm)");

**double** height = sc.nextDouble();

                System.out.println(id+","+name+","+height);

                System.out.println("New toy product record created.");

                return new Robot(id, name,height);

     }

}

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

**public** **class** SellToyProductCommand **implements** Command{

**private** Scanner sc;

**private** Vector product;

**private** Stack commands;

**public** SellToyProductCommand(Scanner sc, Vector product ,Stack commands){

        this.product=product;

        this.sc=sc;

        this.commands = commands;

    }

**public** **void** execute(){

        System.out.println("Enter code:");

**int** code = sc.nextInt();

        System.out.println("Quantity to be sold:");

**int** qty =sc.nextInt();

        System.out.println("Selling price:");

**double** price =sc.nextDouble();

        if(product.size() >0){

            if(code == 1001){

                Enumeration enu = product.elements();

                ToyProduct tp = (ToyProduct)enu.nextElement();

                if( qty<tp.getQty() ){

                    System.out.println("Invalid quantity (current quantity < selling quantity).");

                }else{

                    tp.setQty(qty);

                    tp.setPrice(price);

                    System.out.println("Sold "+qty+"boxes of "+tp.getName()+".Current quantity is "+tp.getQty()+" Selling price is $"+tp.getPrice());

                }

            }else{

                Enumeration enu = product.elements();

                ToyProduct tp = (ToyProduct)enu.nextElement();

                tp.setQty(qty);

                tp.setPrice(price);

                System.out.println("Sold "+ qty+"boxes of "+tp.getName()+".Current quantity is "+tp.getQty()+" Selling price is $"+tp.getPrice());

            }

        }else{

            System.out.println("noting to sell");

        }

    }

**public** **void** undo(){

    }

**public** **void** redo(){

    }

}

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

**public** **class** SellToyProductCommandFactory **implements** CommandFactory{

**private** Scanner sc;

**private** Vector product;

**private** Stack commands;

**public** SellToyProductCommandFactory(Scanner sc, Vector product ,Stack commands){

        this.product=product;

        this.sc=sc;

        this.commands = commands;

    }

**public** Command createCommend(){

        Command c = new SellToyProductCommand(sc, product, commands);

        commands.push(c);

        return c;

    }

}

**public** **class** toy {

**public** **static** **void** doAction(ToyProduct s){

        System.out.println(s);

    }

}

import java**.**util**.**Vector;

**public** **class** ToyProduct {

**private** **int** productID;

**private** String name;

**private** **int** qty;

**private** **double** cost;

**private** **double** price;

**public** ToyProduct(){

    }

**public** ToyProduct(**int** productID , String name){

        this.productID = productID;

        this.name = name;

    }

**public** **int** getProductID(){

        return productID;

    }

**public** String getName(){

        return name;

    }

**public** **void** setName(String name){

        this.name = name;

    }

**public** **int** getQty(){

        return qty;

    }

**public** **void** setQty(**int** qty){

        this.qty = qty;

    }

**public** **double** getCost(){

        return cost;

    }

**public** **void** setCost(**double** cost){

        this.cost=cost;

    }

**public** **double** getPrice() {

        return price;

    }

**public** **void** setPrice(**double** price) {

        this.price = price;

    }

**public** String toString() {

*// TODO Auto-generated method stub*

        return productID+"   "+name+"         "+qty;

    }

}

import java**.**util**.**Scanner;

**public** **interface** ToyProductFactory {

**public** **abstract** ToyProduct createProduct(Scanner sc) ;

}

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

**public** **class** UndoCommand **implements** Command{

**private** Stack commands;

**private** Stack redos;

**public** UndoCommand(Stack commands , Stack redos){

        this.commands = commands;

        this.redos = redos;

    }

**public** **void** execute(){

        if(commands.size() != 0 ){

            Command c = (Command) commands.pop();

            c.undo();

            redos.push(c);

            System.out.println("Undo completed");

        }else{

            System.out.println("Command is Empty!");

        }

    }

**public** **void** undo(){

    }

**public** **void** redo(){

    }

}

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

**public** **class** UndoCommandFactory **implements** CommandFactory{

**private** Stack commands;

**private** Stack redos;

**public** UndoCommandFactory(Stack commands , Stack redos){

        this.commands = commands;

        this.redos = redos;

    }

**public** Command createCommend(){

        return new UndoCommand(commands, redos);

    }

}

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

**public** **class** UndoListCommand **implements** Command{

**private** Stack commands;

**private** Stack redos;

**private** Vector product;

**public** UndoListCommand(Stack commands,Stack redos,Vector product){

        this.commands=commands;

        this.redos = redos;

        this.product=product;

    }

**public** **void** execute(){

        System.out.println("Undo List:");

        if(commands.empty()){

            System.out.println("Empty");

        }else{

            Command c = (Command) commands.pop();

            if(c.equals(c)){

                System.out.println("Create "+c);

            }

        }

        System.out.println("Redo List:");

        if(redos.isEmpty()){

            System.out.println("Empty");

        }else{

            Command c = (Command) redos.pop();

            System.out.println(c+"");

        }

    }

**public** **void** undo(){

    }

**public** **void** redo(){

    }

}

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

**public** **class** UndoListCommandFactory **implements** CommandFactory{

**private** Stack commands;

**private** Stack redos;

**private** Vector product;

**public** UndoListCommandFactory(Stack commands,Stack redos,Vector product){

        this.commands=commands;

        this.redos = redos;

        this.product=product;

    }

**public** Command createCommend(){

        Command c = new UndoListCommand(commands, redos, product);

        return c;

    }

}