2021 上

阅读下列说明和Java代码，将应填入（n）处的字句写在答题纸的对应栏内。

【说明】

层叠菜单是窗口风格的软件系统中经常采用的一种系统功能组织方式。层叠菜单（如图6-1示例）中包含的可能是一个菜单项（直接对应某个功能），也可能是一个子菜单，现在采用组合（composite）设计模式实现层叠菜单，得到如图6-2所示的类图

层叠菜单（如图6-1示例）暂缺

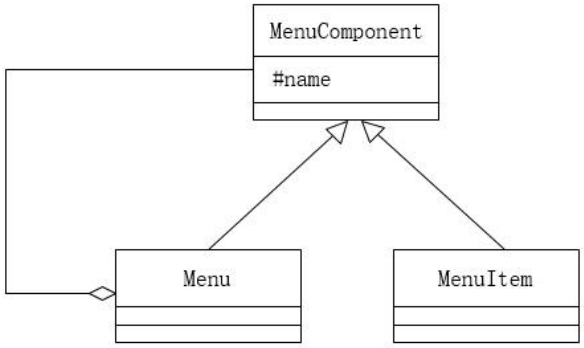

图6-2 类图

5.Java 程序设计真题

```
import java.util.*;

abstract class MenuComponent { // 构成层叠菜单的元素
 (1)  String name; // 菜单项或子菜单名称
public viod printName() { System.out.println(name); }
public  (2)  ;
public abstract boolean removeMenuElement(MenuComponent element);
public (3) ;
}

class MenuItem extends MenuComponent {
public MenuItem(String name) { this.name=name; }
public boolean addMenuElement(MemuComponent element) { return false; }
public boolean removeMenuElement(MenuComponent element){ return false; }
public List<MenuComponent> getElement(){ return null; }
}

class Menu extends MemuComponent {
private (4);
public Menu(String name){
this.name = name;
this.elementList = new ArrayList<MenuComponent>;
}
public boolean addMenuElement(MenuComponent element){
return elementList.add(element);
}
public boolean removeMenuElement(MenuComponent element){
return elementList.remove(element);
}
public List<MenuComponent> getElement() {return elementList;}
}

class CompositeTest {
public static void main(String[] args) {
MenuComponent mainMenu = new Menu("AB");   //此处字符不清晰，以"AB"代替原文
MenuComponent subMenu = new Menu("Chart");
MenuComponent element = new MenuItem("On This Sheet");
(5);
subMenu.addMenuElement(element);
printMenus(mainMenu);
}
private static void printMenus(MenuComponent ifile){
ifile.printName();
List<MenuComponent> children = ifile.getElement();
if(children == null) return; //打印
for(MenuComponent element; children){
printMenus(element);
}
}
}
```

2021 下

阅读下列说明和Java代码，将应填入（n）处的字句写在题纸的对应栏内。

【说明】

享元（flyweight）模式主要用于减少创建对象的数量，以低内存占用，提高性能。现要开发一个网络围棋程序允许多个玩家联机下棋。由于只有一台服务器，为节内存空间，采用享元模式实现该程序，得到如图6-1所的类图。

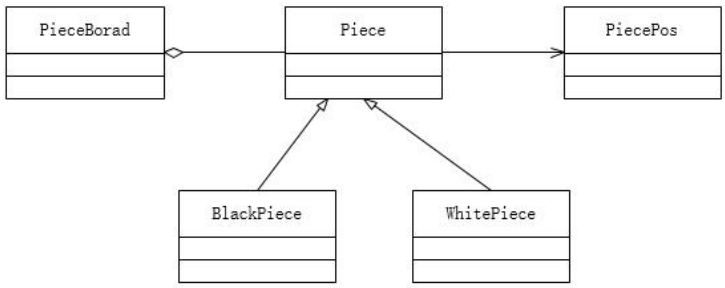

图6-1 类图

【Java代码】

```
import java.util.*:
enum PieceColor {BLACK，WHITE} / /棋子颜色
class PiecePos { / /棋子位置
private intx;
private int y;
pubic PiecePos(int a,int b){
x=a;
y=b;
}
public int getX( ){
return x;
}
public int getY( ){
return y;
 }
}
```

```
abstract class Piece { / / 棋子定义
protected PieceColor m _ color; / / 颜色
protected Piecemopos m _ pos; / / 位置
public Piece(PieceColor,color PiecePos pos){
m_color=color;
m _ pos=pos;
}
(1);
}
class BlackPiece extends Piece{
public BlackPiece(PieceColor  color,PiecePos pos){
super(color,pos);
}
public void draw ( ) {
System out println("draw a black piece");
 }
}
class WhitePiece extends Piece{
public WhitePiece(PieceColor  color,PiecePos pos){
super(color,pos);
}
public void draw( ) {
System.out.println("draw a white piece");
 }
}
```

5.Java 程序设计真题

```
class PieceBoard{
/ / 棋盘上已有的棋子
private static final  ArrayList<(2)>m_arrayPiece=new ArrayList
private String m _ blackName;  / / 黑方名称
private String m _ whiteName;  / / 白方名称
public PieceBoard(String black,String  white){
m_blackName=black;
m_whiteName=white;
}
/ / 一步棋，在棋盘上放一颗棋子
public void SetePiece(PieceColor  color,PiecePos pos){
(3)piece=null;
if（color = = PieceColor.BLACK）{ / / 放黑子
piece = new BlackPiece（color，pos）；  / / 获取一颗黑子
System．out．println（m _ blackName + " 在位置（" + pos．getX( )+","+pos.getY( )+")");
(4) ;
}
else { / / 放白子
piece = new WhitePiece（color，pos）；  / / 获取一颗白子
System．out．println（m whiteName + " 在位置（" + pos．getX0）+","+pos.getYO+")");
(5) ;
}
m_arrayPiece.add(piece);
}
}
```

2020

阅读下列说明和Java代码，将应填入（n） 处的字句写在答 题纸的对应栏内。

【说明】

在线支付是电子商务的一一个重要环节，不同的电子商务平台提供了不同的支付接口。现在需要整合不同电子商务平台的支付接口，使得客户在不同平台上购物时，不需要关心具体的支付接口。拟采用中介者(Mediator) 设计模式来实现该需求，所设计的类图如图6-1所示。

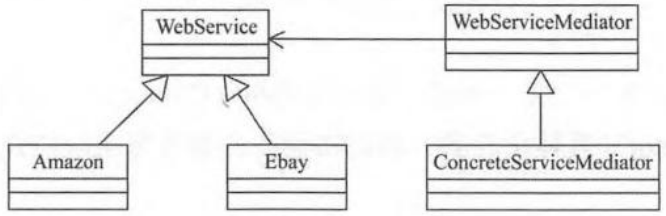

图 6-1　类图

5.Java 程序设计真题

【Java代码】

```
import java.util.*;

interface WebServiceMediator {
   public __(1)__;
   public void SetAmazon(WebService amazon);
   public void SetEbay(WebService ebay);
}
abstract class WebService {
    protected __(2)__ mediator;
    public abstract void SetMediator(WebServiceMediator mediator);
    public __(3)__;
    public abstract void search(double money);
}
class ConcreteServiceMediator implements WebServiceMediator {
    private WebService amazon;
    private WebService ebay;
    public ConcreteServiceMediator() {
        amazon = null;
        ebay = null;
    }
    public void SetAmazon(WebService amazon) {
        this.amazon = amazon;
    }
    public void SetEbay(WebService ebay) {
        this.ebay = ebay;
    }
    public void buy(double money, WebService service) {
        if(service == amazon)
            amazon.search(money);
        else
            ebay.search(money);
    }
}
class Amazon extends WebService {
    public void SetMediator(WebServiceMediator mediator) {
        this.mediator = mediator;
    }
    public void buyService(double money) {
        __(4)__;
    }
    public void search (double money) {
        System.out.println("Amazon receive: " + money) ;
    }
}
class Ebay extends WebService {
    public void SetMediator(WebServiceMediator mediator) {
        this.mediator = mediator;
    }
    public void buyService(double money) {
         (5) ;
```

2019 下

阅读下列说明和Java代码，将应填入 （n） 处的字句写在答题纸的对应栏内。

【说明】

某文件管理系统中定义了类OfficeDoe和DocExplorer。当类OfficeDoe发生变化时，类DocExplorer的所有对象都要更新其自身的状态。现采用观察者（Observer）设计模式来实现该需求，所设计的类图如图6-1所示。

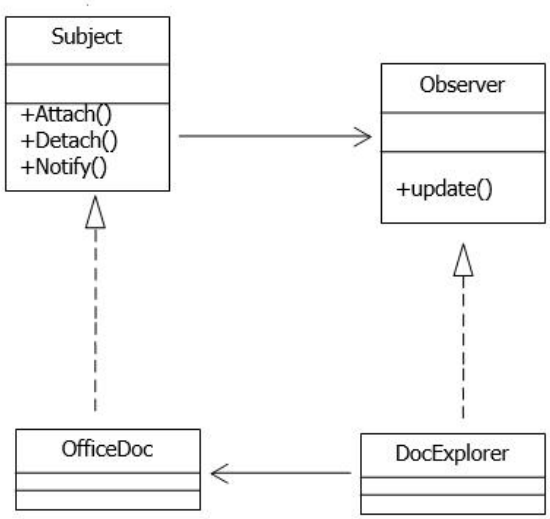

图6-1

5.Java 程序设计真题

【Java代码】

```
import java.util.*;
interface Observer{
    public  (1)   ;
}

interface Subject{
    public void Attach(Observer obs);
    public void Detach(Observer obs);
    public void Notify();
    public void setStatus(int status);
    public int getStatus();
}

class OfficeDoc implements Subject{
    private List<  (2)  > myObs;
    private String mySubjectName;
    private int m_status;

    public OfficeDoc(String name){
        mySubjectName=name;
        this.myObs=new ArrayList<Observer>();
        m_status=0;
    }
    public void Attach(Observer obs){this.myObs.add(obs);}
    public void Detach(Observer obs){this.myObs.remove(obs);}
    public void Notify(){
        for(Observer obs:this.myObs){
            (3) ;
        }
    }
    public void setStatus(int status){
        m_status=status;
        System.out.println("SetStatus subject["+mySubjectName+"]status:"+status);
    }
    public int getStatus(){return m_status;}
}

class DocExplorer implements Observer{
    private String myObsName;
    public DocExplorer(String name,  (4)   sub){
        myObsName=name;
        sub.  (5)   ;
    }
    public void update(){
        System.out.println("update observer["+myObsName+"]");
    }
}

class ObserverTest{
    public static void main(String []args) {
      System.out.println("Hello World!");
       Subject subjectA=new OfficeDoc("subject A");
       Observer oberverA=new DocExplorer("observer A",subjectA);
       subjectA.setStatus(1);
       subjectA.Notify();
    }
}
```

2019 上

阅读下列说明和Java代码，将应填入（n）处的字句写在答题纸的对应栏内。

【说明】

某软件公司欲开发一款汽车竞速类游戏，需要模拟长轮胎和短轮胎急刹车时在路面上留下的不同痕迹，并考虑后续能模拟更多种轮胎急刹车时的痕迹。现采用策略（Strategy）设计模式来实现该需求，所设计的类图如图5-1所示。

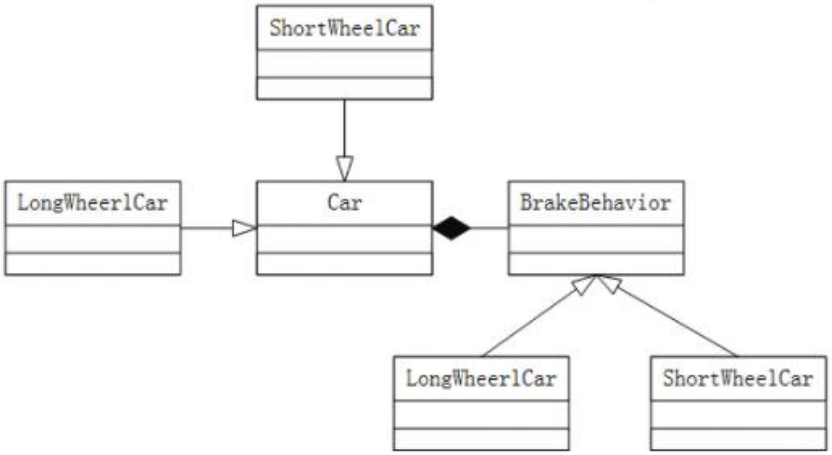

图5-1 类图

5.Java 程序设计真题

【Java 代码】

```
import java.util.*;
interface BrakeBehavior  {
      public     (1)      ;
        /*  其余代码省略  */
};
class LongWheelBrake implements BrakeBehavior {
     public void stop() { System.out.println("模拟长轮胎刹车痕迹！  "); }
       /*  其余代码省略 */
};

class ShortWheelBrake implements BrakeBehavior {
       public void stop() { System.out.println("模拟短轮胎刹车痕迹！  "); }
       /* 其余代码省略 */
};
abstract class Car {
     protected    _____(2)____ wheel;
     public  void brake() {    ____(3)____ ; }
     /* 其余代码省略 */
};
class ShortWheelCar extends Car {
      public ShortWheelCar(BrakeBehavior behavior) {
              __(4)__ ;
      }
      /* 其余代码省略 */
};
class StrategyTest{
    public static void main(String[] args) {
       BrakeBehavior brake = new ShortWheelBrake();
       ShortWheelCar car1 = new ShortWheelCar(brake);
       car1.     __(5)__  ;
    }
}
```

2018 下

阅读下列说明和 Java代码，将应填入（n）处的字句写在答题纸的对应栏内。

【说明】

某航空公司的会员积分系统将其会员划分为:普卡 (Basic) 、银卡(Silver)和金卡 (Gold)

三个等级。非会员 (Non Member)可以申请成为普卡会员。会员的等级根据其 一年内累积的里程数进行调整。描述会员等级调整的状态图如图 6-1 所示 。现采用状态 (State) 模式

实现上述场景，得到如图 6-2 所示的类图。

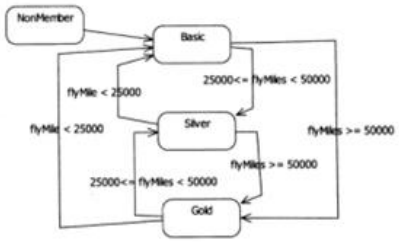

图 6-1 会员等级调整状态图

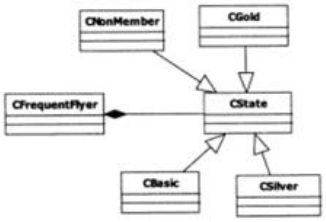

图 6-2 状态模式类图

【Java 代码】

```
import java.util.*;

abstract class CState {
    public int flyMiles;        // 里程数
    public _____(1)_____;  // 根据累积里程数调整会员等级
}
class CNoCustomer extends CState {     // 非会员
    public double travel(int miles, FrequentFlyer context) {
        System.out.println("Your travel will not account for points");
        return miles;              // 不累积里程数
    }
}
class CBasic extends CState {       // 普卡会员
    public double travel(int miles, FrequentFlyer context) {
        if(context.flyMiles >= 25000 && context.flyMiles < 50000)
            _____(2)_____;
        if(context.flyMiles >= 50000)
            _____(3)_____;
        return miles;

    }
}

class CGold extends CState {            // 金卡会员
    public double travel(int miles, FrequentFlyer context) {
        if(context.flyMiles >= 25000 && context.flyMiles < 50000)
            _____(4)_____;
        if(context.flyMiles < 25000)
            _____(5)_____;
        return miles + 0.5*miles;       // 累积里程数
    }
}
class CSilver extends CState {        // 银卡会员
    public double travel(int miles, FrequentFlyer context) {
        if(context.flyMiles <= 25000)
            context.setState(new CBasic());
        if(context.flyMiles >= 50000)
            context.setState(new CGold());
        return (miles + 0.25*miles);      // 累积里程数
    }
}

class FrequentFlyer {
    CState state;
    double flyMiles;
    public FrequentFlyer(){
        state = new CNoCustomer();
        flyMiles = 0;
        setState(state);
    }
    public void setState(CState state){     this.state = state;   }
    public void travel(int miles) {
        double bonusMiles = state.travel(miles, this);
        flyMiles = flyMiles + bonusMiles;
    }
}
```

2018 上

阅读下列说明和Java代码，将应填入（n）处的字句写在答题纸的对应栏内。

【说明】

生成器（Builder）模式的意图是将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。图6-1所示为其类图。

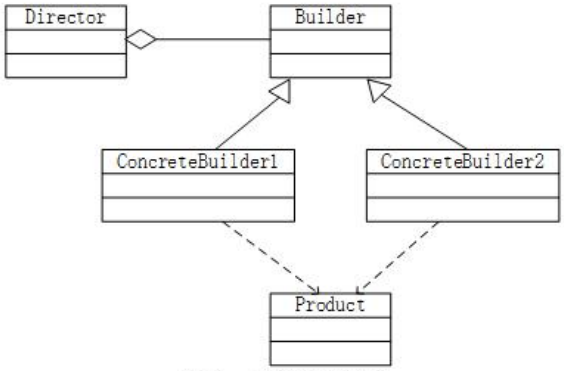

图 6-1　生成器模式类图

【Java代码】

```
import java.util.*;

class Product {
        private String partA;
        private String partB;
        public Product() {}
        public void setPartA(String s) { partA = s; }
        public void setPartB(String s) { partB = s; }
}
interface Builder {
       public    (1)    ;
       public void buildPartB();
       public    (2)    ;
}
class ConcreteBuilder1 implements Builder {
        private Product product;
        public ConcreteBuilder1() { product = new Product();  }
        public void buildPartA() {      (3)    ("Component A"); }
        public void buildPartB() {      (4)    ("Component B"); }
        public Product getResult() { return product;}
}
class ConcreteBuilder2 implements Builder {

        // 代码省略

}

class Director {
        private Builder builder;
        public Director(Builder builder) {this.builder = builder; }
        public void construct() {
                (5)    ;
              // 代码省略
        }
}

class Test {
        public static void main(String[] args) {
              Director director1 = new Director(new ConcreteBuilder1());
              director1.construct();
        }
}
```

2017 上

阅读下列说明和 Java代码，将应填入 （n）处的字句写在答题纸的对应栏内。

【说明】

某快餐厅主要制作并出售儿童套餐，一般包括主餐（各类比萨）、饮料和玩具，其餐品种类可能不同，但其制作过程相同。前台服务员（Waiter）调度厨师制作套餐。现采用生成器（Builder）模式实现制作过程，得到如图 6-1 所示的类图。

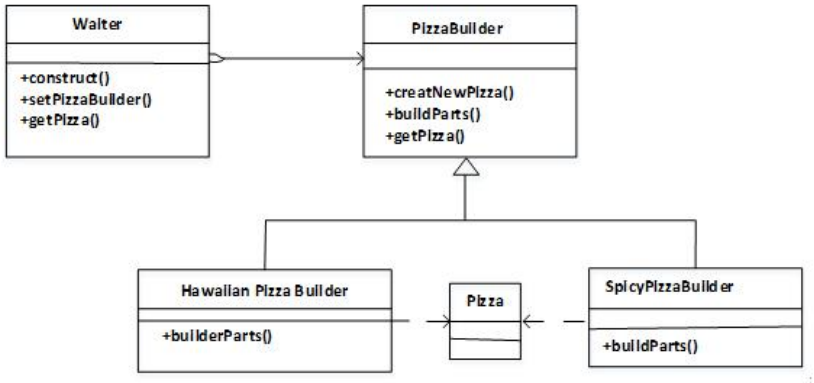

图 6-1 类图

5.Java 程序设计真题

图 6-1 类图

【Java代码】

```
class Pizza  {
      private String parts;
      public void setParts(String parts) {      this.parts = parts;  }
      public String toString() {       return this.parts;    }
}

abstract class PizzaBuilder {
       protected Pizza pizza;
       public Pizza getPizza() { return pizza;   }
       public void  createNewPizza() {       pizza = new Pizza();      }
       public   (1)    ;
}

class HawaiianPizzaBuilder extends PizzaBuilder {
  public void buildParts() {      pizza.setParts("cross + mild + ham&pineapp1e");
}

class SpicyPizzaBuilder extends PizzaBuilder {
    public void buildParts() {  pizza.setParts("pan baked + hot + pepperoni&salami");          }
}

class Waiter {
       private PizzaBuilder pizzaBuilder;

       public void setPizzaBuilder(PizzaBuilder pizzaBuilder) {   /*设置构建器*/
              ( 2 )    ;
      }
      public Pizza getPizza(){ return pizzaBuilder.getPizza(); }

     public void construct() {      /*构建*/
          pizzaBuilder.createNewPizza();
            ( 3 )    ;
      }
}

Class FastFoodOrdering {
       public static viod mainSting[]args) {
            Waiter waiter = new Waiter();
            PizzaBuilder hawaiian_pizzabuilder = new HawaiianPizzaBuilder();

          ( 4 )    ;
          ( 5 )    ;
          System.out.println("pizza: " + waiter.getPizza());
      }
}
```

程序的输出结果为：

Pizza:cross + mild + ham&pineapple

2017 下

阅读下列说明和Java代码，将应填入（n）处的字句写在答题纸的对应栏内。

【说明】

某图像预览程序要求能够查看BMP、JPEG和GIF三种格式的文件，且能够在Windows和Linux两种操作系统上运行。程序需具有较好的扩展性以支持新的文件格式和操作系统。为满足上述需求并减少所需生成的子类数目，现采用桥接（Bridge）模式进行设计，得到如图6-1所示的类图。

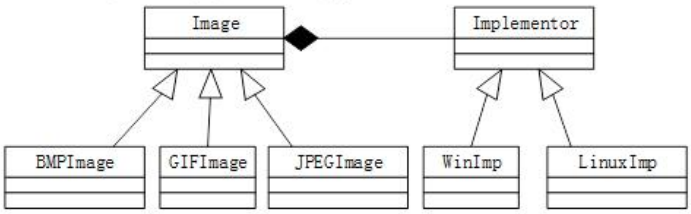

图6-1

5.Java 程序设计真题

【Java代码】

```
import java.util.*;

class Matrix{  // 各种格式的文件最终都被转化为像素矩阵
      // 此处代码省略
};
abstract class Implementor{
      public      (1)      ;  // 显示像素矩阵 m
};
class WinImp extends Implementor{
       public void doPaint(Matrix m){   // 调用 Windows 系统的绘制函数绘制像素矩阵
       }
};
class LinuxImp extends Implementor{
      public void doPaint(Matrix m){   // 调用 Linux 系统的绘制函数绘制像素矩阵
      }
};
abstract class Image {
      public void setImp(Implementor imp) { this.imp = imp; }
      public abstract void parseFile(String fileName);
      protected Implementor imp;
};
class BMPImage extends Image{
       // 此处代码省略
};
class GIFImage extends Image{
      public void parseFile(String fileName) {
         // 此处解析 BMP 文件并获得一个像素矩阵对象 m
               (2)    ;  // 显示像素矩阵 m
      }
};
class JPEGImage extends Image{
       //此处代码省略
};
class Main{
       public static void main(String[]args) {
          // 在 Linux 操作系统上查看 demo.gif 图像文件
          Image image=      (3)     ;
          Implementor imageImp=     (4)     ;
                (5)     ;
          Image.parseFile("demo.gif");
       }
}
```

2016 上

阅读下列说明和Java代码，将应填入__（n）__处的字句写在答题纸的对应栏内。

【说明】

某软件系统中，已设计并实现了用于显示地址信息的类Address（如图6-1所示），现要求提供基于Dutch语言的地址信息显示接口。为了实现该要求并考虑到以后可能还会出现新的语言的接口，决定采用适配器（Adapter）模式实现该要求，得到如图6-1所示的类图。

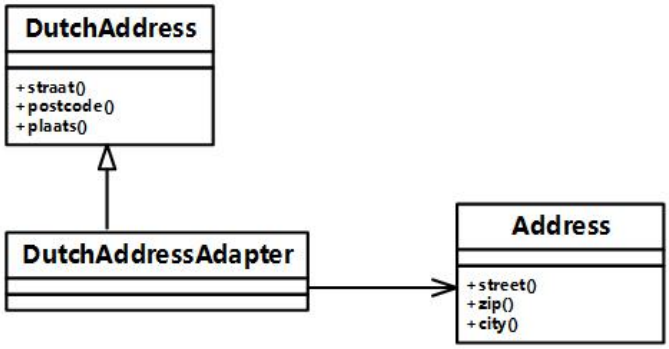

图6-1 适配器模式类图

【Java代码】

```
import java.util.*;

Class Address{
   public void street()  {   //实现代码省略  }
   public void zip()     {   //实现代码省略  }
   public void city()    {   //实现代码省略  }
//其他成员省略
};

class DutchAddress{
   public void straat()   {   //实现代码省略  }
   public void postcode() {   //实现代码省略  }
   public void plaats()   {   //实现代码省略  }
//其他成员省略
};
```

```
class DutchAddressAdapter extends DutchAddress {
    private   (1)   ;

    public DutchAddressAdapter (Address addr){
        address= addr;
    }

    public void straat() {
        (2)   ;
    }

    public void postcode() {
        (3)   ;
    }

    public void plaats(){
        (4)   ;
    }
//其他成员省略
};

class Test {
   public static void main(String[] args) {
      Address  addr= new Address();
      (5)   ;
      System.out.println("\n The DutchAddress\n");
      testDutch(addrAdapter);
   }

   Static void  testDutch(DutchAddress addr){
       addr.straat();
       addr.postcode();
       addr.plaats();
   }
}
```

2016 下

阅读下列说明和java代码，将应填入 （n） 处的字句写在答题纸的对应栏内。

【说明】

某发票（Invoice）由抬头（Head）部分、正文部分和脚注（Foot）部分构成。现采用装饰（Decorator）模式实现打印发票的功能，得到如图6-1所示的类图。

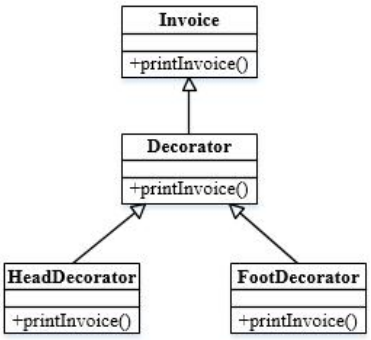

图6-1 类图

【java代码】

```
class invoice{
public void printInvoice(){
        System.out.println ( "This is the content of the invoice!");
   }
}
class Decorator extends Invoice {
   protected Invoice ticket;
   public Decorator(Invoice t){
       ticket = t;
}
   public void printInvoice(){
       if(ticket != null)
            ____(1)____;
 }
}
class HeadDecorator extends Decorator{
      public HeadDecorator(Invoice t){
         super(t);
}
    public void printInvoice (){
         Systent.out.println( "This is the header of the invoice! ");
            ____(2)____;
   }
}
```

```
class FootDecorator extends Decorator {
  public FootDecorator(Invoice t){
  super(t);
}
   public void printInvoice(){
        _____(3)_____;
         Systent.out.println( "This is the footnote of the invoice! ");
  }
}
Class test {
  public static void main(String[] args){
   Invoice t =new Invioce();
   Invoice ticket;
   ticket=___(4)_____;
   ticket.printInvoice();
   Systent.out.println("----------");
   ticket=_____(5)_____;
   ticket.printInvoice();
  }
}
```

程序的输出结果为：

```
This is the header of the invoice!
This is the content of the invoice!
This is the footnote of the invoice!
 ----------
This is the header of the invoice!
This is the footnote of the invoice!
```

2015 下

阅读下列说明和Java代码，将应填入 (n) 处的字句写在答题纸的对应栏内。

【说明】

某大型购物中心欲开发一套收银软件，要求其能够支持购物中心在不同时期推出的各种促销活动，如打折、返利（例如，满300返100）等等。现采用策略（Strategy）模式实现该要求，得到如图6-1所示的类图。

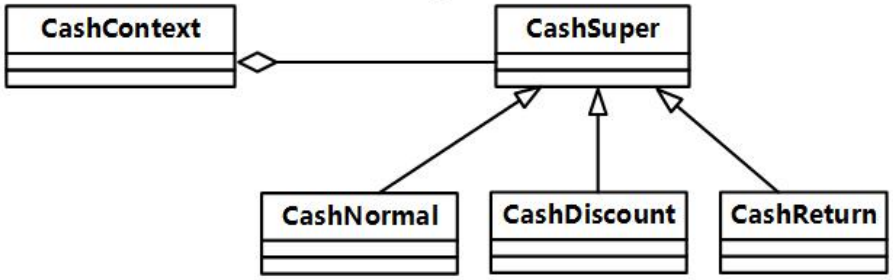

图6-1 策略模式类图

【Java代码】

5.Java 程序设计真题

【Java代码】

```
import java.util.*;
enum TYPE { NORMAL, CASH_DISCOUNT, CASH_RETURN};
interface CashSuper {
  public __(1)__;
}
class CashNormal implements CashSuper{    // 正常收费子类
  public double acceptCash(double money){
    return money;
  }
}
class CashDiscount implements CashSuper {
  private double moneyDiscount;                  // 折扣率
  public CashDiscount(double moneyDiscount) {
    this moneyDiscount = moneyDiscount;
  }
  public double acceptCash(double money) {
    return money* moneyDiscount;
  }
}
class CashReturn implements CashSuper {              // 满额返利
  private double moneyCondition;
  private double moneyReturn;
  public CashReturn(double moneyCondition, double moneyReturn) {
    this.moneyCondition =moneyCondition;     // 满额数额
    this.moneyReturn =moneyReturn;           // 返利数额
  }
  public double acceptCash(double money) {
    double result = money;
    if(money >= moneyCondition )
      result=money-Math.floor(money/moneyCondition ) * moneyReturn;
    return result;
  }
```

```
}
class CashContext_{
  private CashSuper cs;
  private TYPE  t;
  public CashContext(TYPE t)  {
    switch(t){
      case NORMAL:     // 正常收费
        __(2)__;
        break;
      case CASH_DISCOUNT:      // 打8折
        __(3)__;
        break;
      case CASH_RETURN:     // 满300返100
        __(4)__;
        break;
    }
  }
  public double GetResult(double money) {
     __(5)__;
  }
  //此处略去main()函数
}
```

2015 上

阅读下列说明和Java代码，将应填入 (n) 处的字句写在答题纸的对应栏内。

【说明】

某图书管理系统中管理着两种类型的文献：图书和论文。现在要求统计所有馆藏文献的总页码（假设图书馆中有一本540页的图书和两篇各25页的论文，那么馆藏文献的总页码就是590页）。采用Visitor（访问者）模式实现该要求，得到如图6-1所示的类图。

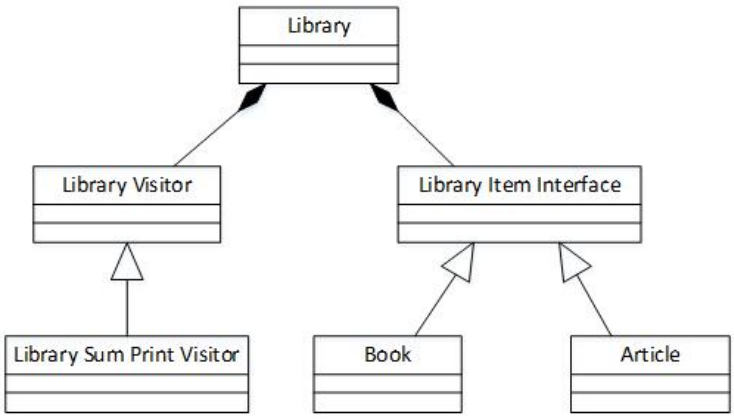

图6-1 Visitor模式类图

```
class  Article implements LibraryItemInterface{
    private String m_title;      //论文名
    private String m_author;    //论文作者
    private int    m_start_page;
    private int    m_end_page;
    public Article(String p_author, String p_title,int p_start_page,int p_end_page){
        m_title=p_title;
        m_author= p_author;
        m_end_page=p_end_page;
    }
    public int getNumberOfPages(){
        return m_end_page - m_start_page;
    }
    public void accept(LibraryVisitor Visitor){
        ____(4)____;
    }
}
class Book implements LibraryItemInterface{
    private String m_title;        //书名
    private String m_author;     //书作者
    private int    m_pages;        //页教
    public Book(String p_author, String p_title,int p_ pages){
        m_title= p_title;
        m_author= p_author;
        m_pages= p_pages;
    }
    public int getNumberOfPages(){
        return m_pages;
    }
    public void accept(LibraryVisitor visitor){
        ___(5)____;
    }
}
```

2014 上

阅读下列说明和Java代码，将应填入__（n）__处的字句写在答题纸的对应栏内.

【说明】

某实验室欲建立一个实验室环境监测系统，能够显示实验室的温度、湿度以及洁净度等环境数据。当获取到最新的环境测量数据时，显示的环境数据能够更新。

现在采用观察者（Observer）模式来开发该系统。观察者模式的类图如图6-1所示。

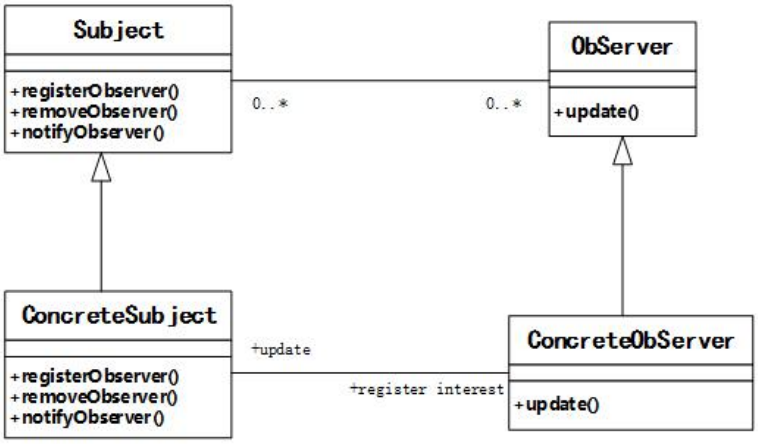

图 6-1 观察者模式类图

【Java代码】

```
import java.util.*;
interface Observer {
    public void update(float temp, float humidity, float cleanness);
}
interface Subject {
    public void registerObserver(Observer o); //注册对主题感兴趣的观察者
    public void removeObserver(Observer o);   //删除观察者
    public void notifyObservers();            //当主题发生变化时通知观察者
}
class EnvironmentData implements   (1)   {
    private ArrayList observers;
    private float temperature, humidity, cleanness;
    public EnvironmentData() {   observers = new ArrayList(); }
    public void registerObserver(Observer o) { observers.add(o); }
    public void removeObserver(Observer o)   { /* 代码省略 */ }
    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            Observer observer = (Observer)observers.get(i);
            (2) ;
        }
    }
    public void measurementsChanged() {   (3)   ; }
```

```
    public void setMeasurements(float temperature, float humidity, float cleanness) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.cleanness = cleanness;
        ___(4)___;
    }
}
class CurrentConditionsDisplay implements ___(5)___ {
    private float temperature;
    private float humidity;
    private float cleanness;
    private Subject envData;
    public CurrentConditionsDisplay(Subject envData) {
        this.envData = envData;
        ___(6)___;
}
    public void update(float temperature, float humidity, float cleanness) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.cleanness = cleanness;
        display();
}
    public void display() {/* 代码省略 */}
}
class EnvironmentMonitor{
    public static void main(String[] args) {
        EnvironmentData envData = new EnvironmentData();
        CurrentConditionsDisplay currentDisplay = new CnrrentConditionsDisplay(envData);
        envData.setMeasurements(80, 65, 30.4f);
    }
}
```

2014 下

阅读下列说明和Java代码，将应填入 (n) 处的字句写在答题纸的对应栏内。

【说明】

某灯具厂商欲生产一个灯具遥控器，该遥控器具有7个可编程的插槽，每个插槽都有开关灯具的开关，现采用Command（命令）模式实现该遥控器的软件部分。Command模式的类图如图6-1所示。

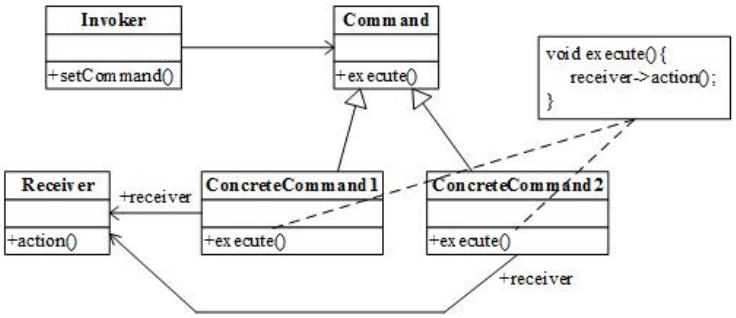

图 6-1 Command 模式类图

【Java代码】

```
class Light {
  public Light() {}
  public Light(String name) { /* 代码省略 */ }
  public void on()  { /* 代码省略 */ }    // 开灯
  public void off()  { /* 代码省略 */ }    // 关灯
  // 其余代码省略
}
  __(1)__ {
  public void execute();
}
class LightOnCommand implements Command { // 开灯命令
  Light light;
  public LightOnCommand(Light light) { this.light=light; }
  public void execute() { __(2)__ ; }
}
class LightOffCommand implements Command { // 关灯命令
  Light light;
  public LightOffCommand(Light light) { this.light=light; }
  public void execute(){ __(3)__ ; }
}
class RemoteControl { // 遥控器
  Command[] onCommands=new Command[7];
  Command[] offCommands=new Command[7];
  public RemoteControl() { /* 代码省略 */ }
  public void setCommand(int slot, Command onCommand, Command offCommand) {
    __(4)__ =onCommand;
    __(5)__ =offCommand;
  }
  public void onButtonWasPushed(int slot) {
     __(6)__ ;
  }
```

```
  public void offlButtonWasPushed(int slot){
     (7)  ;
  }
}
class RemoteLoader {
  public static void main(String[] args) {
    RemoteControl remoteControl=new RemoteControl();
    Light livingRoomLight=new Light("Living Room");
    Light kitchenLight=new Light("kitchen");
    LightOnCommand livingRoomLightOn=new LightOnCommand(livingRoomLight);
    LightOffCommand livingRoomLightOff=new LightOffCommand(livingRoomLight);
    LightOnCommand kitchenLightOn=new LightOnCommand(kitchenLight);
    LightOffCommand kitchenLightOff=new LightOffCommand(kitchenLight);
    remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);
    remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);
    remoteControl.onButtonWasPushed(0);
    remoteControl.offButtonWasPushed(0);
    remoteControl.onButtonWasPushed(1);
    remoteControl.offButtonWasPushed(1);
  }
}
```