窗体顶端

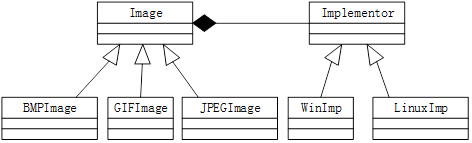
# 面向对象程序设计

您现在的位置：[希赛网](http://www.educity.cn/" \o "希赛网) > [云阅读](http://www.educity.cn/jiaocheng/) > [软件设计师考试试题分类精解（2018版）](http://www.educity.cn/jiaocheng/zt251.html" \t "http://www.educity.cn/jiaocheng/_blank) > **试题1(2017年下半年试题5)**

第 18 章：面向对象程序设计作者：[希赛软考学院](http://www.educity.cn/jiaocheng/a%cf%a3%c8%fc%c8%ed%bf%bc%d1%a7%d4%ba.html" \o "希赛软考学院" \t "http://www.educity.cn/jiaocheng/_blank)    来源：希赛软考学院    2017年11月21日

# **试题1(2017年下半年试题5)**

阅读下列说明和C++代码，将应填入（  ）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5-1所示的类图。

  
**图5-1**

【C++代码】  
#include<iostream>   
#include<string>  
using namespace std;  
   
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  
};  
class Implementor {  
public:  
                （1）;   //  显示像素矩阵 m  
};  
class WinImp:public Implementor {  
public:  
        void doPaint(Matrix m) { /\*调用 Windows 系统的绘制函数绘制像素矩阵\*/ }  
};  
class LinuxImp:public Implementor{  
public:  
        void doPaint(Matrix m) { /\*调用 Linux 系统的绘制函数绘制像素矩阵\*/ }  
};  
class Image {  
public:  
       void setImp(Implementor\*imp)  {this->imp = imp;}  
       virtual viod parseFile(string fileName) = 0;  
protected:  
        Implementor\*imp;  
};  
class BMPImage:public Image{  
         //此处省略代码  
};  
class GIFImage:public Image{  
public:  
        void  parseFile(string fileName) {  
            //此处解析 GIF 文件并获得一个像素矩阵对象m  
              （2）;显示像素矩阵m  
        }  
};  
class JPEGImage:public Image{  
         //  此处代码省略  
};  
   
int main() {  
        //  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");  
        return 0;  
}

**试题分析**

问题解析  
第一空是显示像素矩阵 m  
从类图来看 Implementor是WinImp和LinuxImp两子类的父类。那就需要从子类中去找共同的方法，然后把它们抽象出来。  
共同的方法为: void doPaint(Matrix m) ;抽象就成了 virtual void doPaint(Matrix m) = 0；  
  
第二空是显示像素矩阵m  
在Image的类和其子类中，要显示像素矩阵，可以使用调用Implementor类的方法doPaint，而Image类中定义了对象imp。  
即调用的方法为： imp->doPaint(m)  
  
第三空是构造出Gif图像的对象 new GIFImage()

第四空是要在Linux操作系统上查看，需要一个LinuxImp的对象 . new LinuxImp()

 第五空是把imageImp对象传递，以便能够查看Gif图像文件，image->setImp(imageImp)

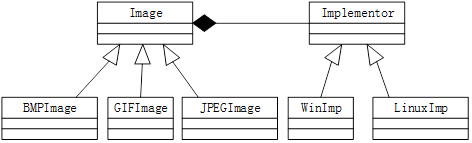
**试题答案**

（5）

1.virtual void doPaint(Matrix m) = 02. imp->doPaint(m)3. new GIFImage()4. new LinuxImp()5. image->setImp(imageImp)

# **试题2(2017年下半年试题6)**

阅读下列说明和Java代码，将应填入（  ）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**

【Java代码】  
import jav（6）A.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");  
         }  
}

**试题分析**

问题解析  
第一空是显示像素矩阵 m  
从类图来看 Implementor是WinImp和LinuxImp两子类的父类。那就需要从子类中去找共同的方法，然后把它们抽象出来。  
共同的方法为: void doPaint(Matrix m) ;抽象就成了 abstract void doPaint(Matrix m)；此处别忘了abstract关键字。是抽象方法。  
  
第二空是显示像素矩阵m  
在Image的类和其子类中，要显示像素矩阵，可以使用调用Implementor类的方法doPaint，而Image类中定义了对象imp。  
即调用的方法为： imp.doPaint(m)  
  
第三空是构造出Gif图像的对象 new GIFImage()

第四空是要在Linux操作系统上查看，需要一个LinuxImp的对象 . new LinuxImp()

 第五空是把imageImp对象传递，以便能够查看Gif图像文件，image.setImp(imageImp)

**试题答案**

（6）

1.abstract void doPaint(Matrix m)

2.imp.doPaint(m)

3.new GIFImage()

4.new LinuxImp()

5.image.setImp(imageImp)

# **试题3(2017年上半年试题5)**

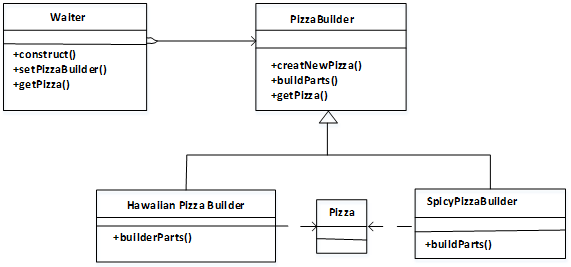
阅读下列说明和 C++代码，将应填入(n)处的字句写在答题纸的对应栏内。  
【说明】  
某快餐厅主要制作井出售儿童套餐，一般包括主餐(各类比萨)、饮料和玩具，其餐品种类可能不同，但其制作过程相同。前台服务员(Waiter)调度厨师制作套餐。现采用生成器(Builder) 模式实现制作过程，得到如图 5-1 所示的类图。  


图5-1     类图

【C++代码】  
#include<iostream>  
#include <string>  
using namespace std;  
   
class Pizza {  
private:  string parts;  
public:  
    void setParts(string parts) {       this->parts=parts;   }  
    string getParts() {    return parts;     }  
};  
class PizzaBuilder {  
protected:Pizza\*  pizza;  
public:  
    Pizza\* getPizza() {    retum pizza;   }  
    void createNewPizza() { pizza = new Pizza();      }  
      (    1    );  
}  
class HawaiianPizzaBuilder :public PizzaBuilder {  
public:  
       void buildParts() {  pizza->setParts("cross +mild + ham&pineapple");     }  
};  
  
class SpicyPizzaBuider: public PizzaBuilder {  
public:  
       void buildParts() {  pizza->setParts("pan baked +hot + ham&pineapple");      }  
   
}  
   
Class Waiter{  
Private:  
      PizzaBuilder\*  pizzaBuilder;  
public:  
      void setPizzaBuilder(PizzaBuilder\* pizzaBuilder)  {    /\*设置构建器\*/  
         (   2   )  
      }  
     Pizza\* getPizza() {   return pizzaBuilder->getPizza(); }  
     void construct() {      /\*构建\*/  
             pizzaBuilder->createNewPizza();  
              (   3   )  
      }   
};  
   
int main(){  
        Waiter\*waiter=new Waiter();  
        PizzaBuilder\*hawaiian pizzabuilder=new HawaiianPizzaBuilder()  
   
(    4   );  
(    5   );  
cout<< "pizza: "<< waiter->getPizza()->getParts()<< endl;  
}  
  
程序的输出结果为:  
pizza: cross + mild + ham&pineapple

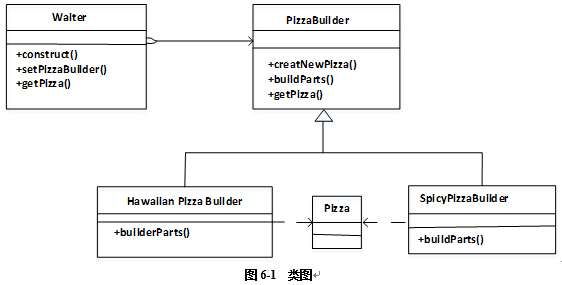
**试题分析**

1.从类图中可以看到这个buildparts函数缺失，又子类中也有buildParts，所以它是虚函数，在扩展类中定义使用  
2.这部分填写设置构建器内容，在waiter类里面，定义pizzaBuilder  
3.从类图知道，构建方法应该是buildParts，当前对象是pizzaBuilder。  
4.前面定义了对象waiter，新建hawaiian\_pizzabuilder类，调用waiter的set方法  
5.调用waiter类中的construct方法，这样可以得到

**试题答案**

（5）（1）virtual void buildParts()  
（2）this->pizzaBuilder=pizzaBuilder  
（3）pizzaBuilder->buildParts()  
（4）waiter->setPizzaBuilder(hawaiian\_pizzabuilder)  
（5）waiter->construct()

# **试题4(2017年上半年试题6)**

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。  
  
【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

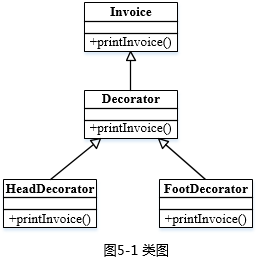
**试题分析**

1.看类图，还差一个buildparts方法，再看下面的类也有buildparts方法，知道应该是abstract void buildParts();  
2.这部分填写设置构建器内容，在waiter类里面，定义pizzaBuilder  
3.从类图知道，构建方法应该是buildParts，当前对象是pizzaBuilder。  
4.前面定义了对象waiter，新建hawaiian\_pizzabuilder类，调用waiter的set方法  
5.调用waiter类中的construct方法，这样可以得到Pizza.

**试题答案**

（6）（1）abstract void buildParts();  
（2）this.pizzaBuilder=pizzaBuilder  
（3）pizzaBuilder.buildParts()  
（4）waiter.setPizzaBuilder(hawaiian\_pizzabuilder)  
（5）waiter.construct()

# **试题5(2016年下半年试题5)**

阅读下列说明和C++-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（lnvoice）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5-1所示的类图。  
  
  
【C++代码】  
#include   
using namespace std;  
class Invoice{  
public:  
    （1）    {          
   cout<<"This is the content of the invoice!"<<endl;  
     }  
};  
class Decorator : public Invoice {  
     Invoice \*ticket;  
public:  
    Decorator(lnvoice \*t)      { ticket = t; }  
     void printInvoice(){  
         if(ticket != NULL)  
 （2）;  
   }  
 };  
class HeadDecorator : public Decorator{  
public:  
    HeadDecorator(lnvoice\*t): Decorator(t) { }  
     void printInvoice() {  
            cout<< "This is the header of the invoice! "<< endl;  
     （3）      ;  
    }  
  };  
   class FootDecorator : public Decorator{  
 public:  
    FootDecorator(Invoice \*t): Decorator(t) { }  
     void printlnvoice(){  
                 （4）   ;  
             cout<< "This is the footnote of the invoice!"<< endl;  
    }  
};  
int main(void) {  
    Invoice t;  
    FootDecorator f(&t);  
    HeadDecorator h(&f);  
    h.printInvoice();  
    cout<<”------------------------”<<endl;  
     FootDecorator a(NULL) ;                                                                                                                                      
    HeadDecorator b(     （5）    );  
    b.printInvoice();  
    return 0;  
}  
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!

**试题分析**

1.Invoice类下，定义虚函数，按类图，函数名是printInvoice  
2.前面定义对象名是ticket,那么在ticket不为空的时候调用函数printInvoice  
3.这部分填写发票的抬头，看类图应该实现函数printInvoice,Decorator装饰模式使用该方法  
4.这部分是发票的脚注，看类图应该实现函数printInvoice,Decorator装饰模式使用该方法  
5.FootDecorator a(NULL) ;脚步的装饰参数是a，调用a参数，

**试题答案**

（5）

（1） virtual void printInvoice()

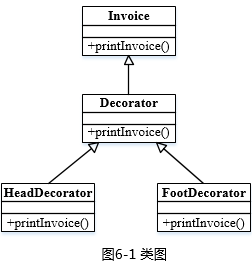
（2） ticket->printInvoice()

（3） Decorator::printInvoice()

（4） Decorator::printInvoice()

（5） &a

# **试题6(2016年下半年试题6-10)**

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（lnvoice）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。  
  
【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(lnvoice t){  
         ticket = t;  
}  
   public void printInvoice(){  
         if(ticket != null)  
                          （1）       ;  
  }     
}  
class HeadDecorator extends Decorator{  
       public HeadDecorator(lnvoice 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 printlnvoice(){  
                       （ 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!

**试题分析**

1.前面定义对象名是ticket,那么在ticket不为空的时候调用方法printInvoice  
2.发票的抬头部分，看类图知道也是实现printInvoice.  
3.发票的脚注部分，看类图知道也是实现printInvoice.  
4.注意调用次序，先调用抬头的方法，再调用脚注的方法  
5.这部分是使用了装饰器模式，比对不使用装饰器模式的优点，调用次序是先装饰模式，再抬头，再尾注

**试题答案**

（6）

（1） ticket.printInvoice()

（2） ticket.printInvoice()

（3） ticket.printInvoice()

（4） new FootDecorator(new HeadDecorator(t))

（5） new FootDecorator(new HeadDecorator(new Decorator(null)))

# **试题7(2016年上半年试题5)**

    阅读下列说明和C++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5-1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5-1所示的类图。

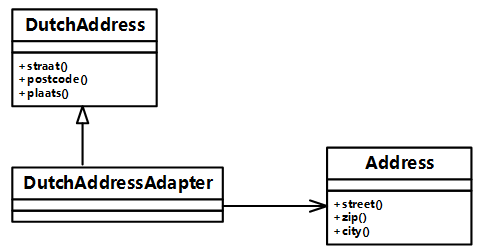


图5-1 适配器模式类图

【C++代码】

#include <iostream>

using namespace std;

class Address{

public:

    void stree()    { /\*  实现代码省略  \*/  }

    void zip()      { /\*  实现代码省略  \*/  }

    void city()     { /\*  实现代码省略  \*/  }

∥其他成员省略

};

class DutchAddress {

public:

    virtual void straat()=0;

    virtual void postcode()=0;

    virtual void plaats()=0;

//其他成员省略

};

class DutchAddressAdapter : public DutchAddress {

private:

       （1）   ;

public:

    DutchAddressAdapter(Address \*addr) {

        address = addr;

    }

    void straat() {

          （2） ;

    }

    void postcode(){

          （3） ;

    }

    void plaat(){

         （4） ;

    }

//其他成员省略

};

void testDutch(DutchAddress \*addr){

       addr->straat();

       addr->postcode();

       addr->plaats();

}

int main(){

    Address\*addr = new Address();

      （5）  ;

    cout<< "\n The DutchAddress\n"<< endl;

    testDutch(addrAdapter);

    return 0;

}

**试题分析**

1.定义一个地址对象  
2.3.4 按类图中adress的三个方法  
5.新建对象addr，调用testDutch(addrAdapter)

**试题答案**

（5）

（1）Address\* address;

（2）address->street();

（3）address->zip();

（4）address->city();

（5）DutchAddress \*addrAdapter=new DutchAddressAdaptor(addr);

# **试题8(2016年上半年试题6)**

试题六（共15分）

    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。

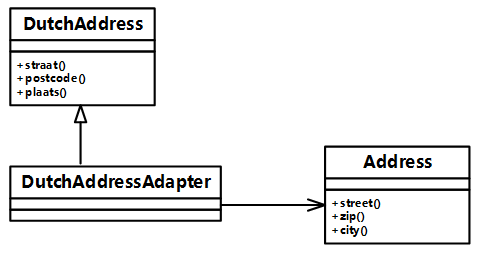


图6-1 适配器模式类图

【Java代码】

import jav（6）A．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();

    }

}

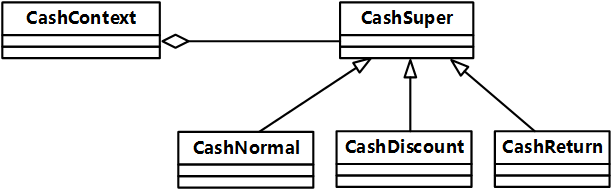
**试题分析**

1.定义一个地址对象  
2.3.4 按类图中adress的三个方法  
5.新建对象addr，调用testDutch(addrAdapter)

**试题答案**

（6）（1）Address address;  
（2）address.street();  
（3）address.zip();  
（4）address.city();  
（5）DutchAddress addrAdapter=new DutchAddressAdapter(addr);

# **试题9(2015年下半年试题5)**

    阅读下列说明和C++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5-1所示的类图。  
      
                                                                   图5-1 策略模式类图【C++代码】  
#include <iostream>  
using namespace std;  
enum TYPE{NORMAL, CASH\_DISCOUNT, CASH\_RETURN};  
class CashSuper{  
public:  
   （1）;  
};  
class CashNormal : public CashSuper {     //正常收费子类  
public:  
    double acceptCash(double money) {     retum money;    }  
};   
class CashDiscount : public CashSuper {  
private:  
    double moneyDiscount;     //  折扣率  
public:  
   CashDiscount(double discount) {    moneyDiscount= discount;    }  
   double acceptCash(double money) {    retum money \* moneyDiscount;    }  
};  
class CashRetum : public CashSuper {      // 满额返利  
private:          
    double moneyCondition;      // 满额数额  
    double moneyReturn;       // 返利数额

public:

    CashRetnm(double motieyCondition, double moneyReturn) {

       this->moneyCondition=moneyCondition;

       this->moneyReturn=moneyReturn;

    }

    double acceptCash(double money) {

       double result = money;

       if(money>=moneyCondition)

           result=money-(int)(money/moneyCondition ) \* moneyReturn;

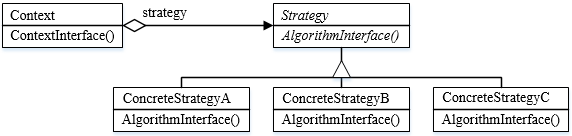
       return  result ;

    }

};

class CashContext {  
private:  
   CashSuper \*cs;  
public:  
   CashContext(int type) {  
     switch(type) {  
      case NORMAL:           //正常收费  
                  (2)       ;  
          break;  
      case CASH\_RETURN:       //满300返100  
                  (3)        ;  
           break;  
      case CASH\_DISCOUNT:     //打八折  
                 (4)        ;  
          break;  
   }  
}  
double GetResult(double money) {  
            (5)     ;  
    }  
};  
//此处略去main()函数

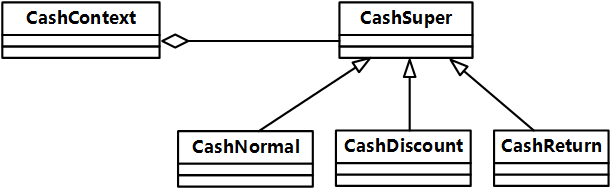
**试题分析**

策略模式的结构图如下：  


**试题答案**

（5）（1）virtual double acceptCash(double money) = 0  
（2）cs = new CashNormal()  
（3）cs = new CashReturn(300,100)  
（4）cs = new CashDiscount(0.8)  
（5）return cs->acceptCash(money)

# **试题10(2015年下半年试题6)**

    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】

import jav（6）A．util.\*;

enum TYPE { NORMAL, CASH\_DISCOUNT, CASH\_RETURN};

interface CashSuper {

   public   （1）  ;

}

class CashNormal implements CashSuper{    // 正常收费子类

   public double accptCash(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) {

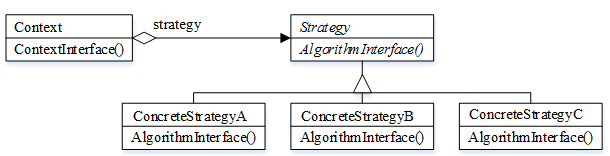
           （5）   ;

   }

   //此处略去main()函数

}

**试题分析**

本题考查策略 (Strategy)  模式的基本概念和应用。  
Strategy模式的设计意图是，定义一系列的算法，把它们一个个封装起来，并且使它们可以相互替换。此模式使得算法可以独立于使用它们的客户而变化，其结构图如下图所示。  
  
• Strategy (策略)定义所有支持的算法的公共接口。 Context 使用这个接口来调用某ConcreteStrategy 定义的算法。  
•     ConcreteStrategy(具体策略)以 Strategy 接口实现某具体算法。  
•     Context（上下文）用一个ConcreteStrategy对象来配置；维护一个对 Strategy对象的引用；可定义一个接口来让Strategy 访问它的数据。  
Strategy模式适用于：  
•许多相关的类仅仅是行为有异。"策略"提供了一种用多个行为中的一个行为来配置一个类的方法。  
•需要使用一个算法的不同变体。例如，定义一些反应不同空间的空间/时间权衡的算法。当这些变体实现为一个算法的类层次时，可以使用策略模式。  
•算法使用客户不应该知道的数据。可使用策略模式以避免暴露复杂的 、与算法相 关的数据结构。  
一个类定义了多种行为，并且这些行为在这个类的操作中以多个条件语旬的形式出现，将相关的条件分支移入它们各自的 Strategy 类中，以代替这些条件语句 。  
本题中类 CashSuper 对应于上图中的类 Strategy，类CashNormal 、CashDiscout 和CashReturn 分别代表 3 种不同的具体促销策略 。CashSuper 类提供其 3 个子类的公共操作接口，由子类给出 3 种不同促销策略的具体实现。在 C++语言中，可以采用继承+(纯) 虚拟函数来实现。从 3 个子类 CashNormal、CashDiscout 和 CashReturn的代码可以看出，公共操作接口为 double acceptCash(double money) 。由于不需要父类 CashSuper 提供任何的促销实现方式，所以这里采用纯虚拟函数。应填入空（1）处的语句是 "virtual double acceptCash(double money) = 0"。  
空（2）-（4）都出现在类 CashContext 中，该类对应于上图中的类 Context ，其作用是依据策略对象来调用不同的策略算法 。因此空（2）-（4）的是根据不同的 case 分支来创建不同的策略对象。由此可知空（2）-（4）分别应填入"cs=newCashNormal()"、"cs =new CashDiscount(0.8) "和"cs = new CashReturn(300，100)"。  
方法 GetResult 是对接口的调用，从而计算出来用不同促销策略之后应付的费用， 这里需要通过 CashSuper 的对象 cs 来调用公共操作接口，因此第（5）空应填入 "return cs.acceptCash(money)"。

**试题答案**

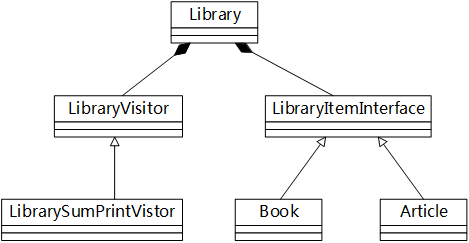
（6）（1）double acceptCash(double money)               
（2）cs = new CashNormal()  
（3）cs = new CashDiscount(0.8)  
（4）cs = new CashReturn(300,100)  
（5）return cs.acceptCash(money)

# **试题11(2015年上半年试题5)**

    阅读下列说明和C++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5-1所示的类图。

  
                                       图5-1 Visitor模式类图

【C++代码】

class LibraryVisitor;

class LibraryItemInterface{

public:

           (1)      ;

};

class Article : public LibraryItemInterface {

private:

    string  m\_title;        //论文名

    string  m\_author;    //论文作者

    int m\_start\_page;

    int m\_end\_page;

public:

    Article(string p\_author, string p\_title, int p\_start\_page,int p\_end\_page );

    int getNumberOfPages();

    void accept(Library Visitor\* visitor);

};

class Book : public LibraryItemInterface {

private:

    string  m\_title;       //书名

    string  m\_author;   //作者

    int m\_pages;         //页数

public:

    Book(string p\_author, string p\_title, int p\_pages);

    int getNumberOfPages();

     void accept(LibraryVisitor\* visitor);

};

class LibraryVisitor {

public:

          (2)     ;

          (3)     ;

    virtual void printSum() = 0;

};

class LibrarySumPrintVisitor : public LibraryVisitor  {          //打印总页数

private:

    int sum;

public:

    LibrarySumPrintVisitor();

    void visit(Book\* p\_book);

    void visit(Article\* p\_article);

    void printSum();

};

// visitor.cpp

int Article: :getNumberOfPages(){

    retum m\_end\_page - m\_start\_page;

}

void Article::accept(LibraryVisitor\* visitor) {       (4)      ;}

Book: :Book(string p\_author, string p\_title, int p\_pages ) {

    m\_title = p\_title;

    m\_author = p\_author;

    m\_pages = p\_pages;

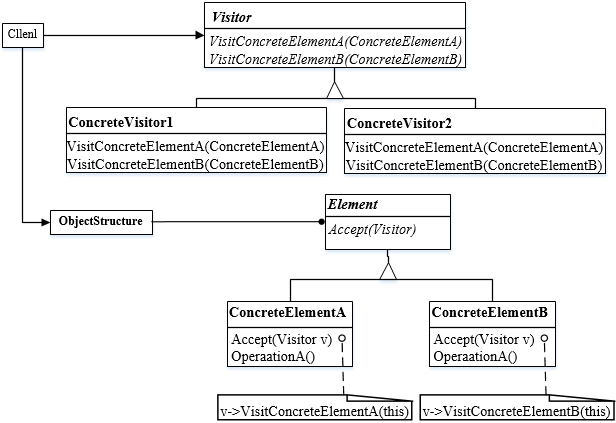
}

int Book::getNumberOfPages(){    return m\_pages;  }

void Book::accept(LibraryVisitor\* visitor){       (5)     ;  }

//其余代码省略

**试题分析**



**试题答案**

（5）

（1）virtual void accept(LibraryVisitor\* visitor)=0

（2）virtual void visit(Book\* p\_book)=0

（3）virtual void visit(Article\* p\_article)=0

（4）(LibraryVisitor\*)visitor->visit(this)

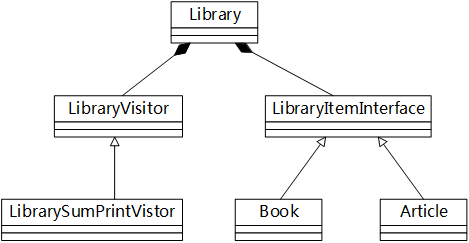
（5）(LibraryVisitor\*)visitor->visit(this)

# **试题12(2015年上半年试题6)**

    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

【Java 代码】

import jav（6）A．util.\*;

interface LibraryVisitor {

          （1）　  ;

          （2）　  ;

    void printSum();

}

class LibrarySumPrintVisitor implements LibraryVisitor {          //打印总页数

    private int sum = 0;

    public void visit(Book p\_book) {

        sum = sum + p\_book.getNumberOfPages();

    }

    public void visit(Article p\_article) {

        sum = sum + p\_article.getNumberOfPages();

    }

    public void printSum(){

        System.out.println("SUM = " + sum);

    }

}

interface LibraryItemInterface {

           (3)       ;

}

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;

    }

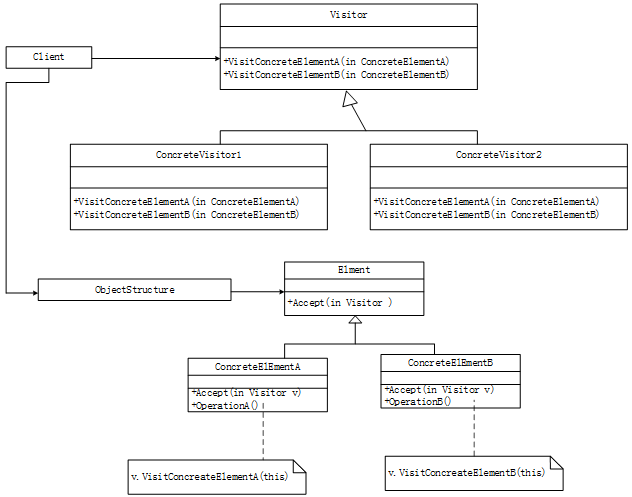
    public void accept(LibraryVisitor visitor){

              (5)       ;

    }

}

**试题分析**

本题考查Visitor (访问者)模式的基本概念和应用。  
访问者模式是行为设计模式中的一种。行为模式不仅描述对象或类的模式，还描述它们之间的通信模式。这些模式刻画了在运行时难以跟踪的复杂的控制流。访问者模式表示一个作用于某对象结构中的各元素的操作。它使在不改变各元素的类的前提下可以定义作用于这些元素的新操作。此模式的结构图如下图所示。  
  
•     Visitor(访问者)为该对象结构中 ConcreteElement   的每一个类声明一个Visit  操作。该操作的名字和特征标识了发送 Visit 请求给该访问者的哪个类。这使得访问者可以确定正被访问元素的具体的类。这样访问者就可以通过该元素的特定接口直接访问它。  
• ConcreteVisitor （具体访问者）实现每个有 Visitor 声明的操作，每个操作实现本算法的一部分，而该算法片段乃是对应于结构中对象的类。ConcreteVisitor 为该算法提供了上下文并存储它的局部状态。这一状态常常在遍历该结构的过程中累积结果。  
• Element (元素)定义以一个访问者为参数的 Accept 操作。  
• ConcreteElement   (具体元素)实现以一个访问者为参数的 Accept 操作。  
• ObjectStructure  (对象结构〉能枚举它的元素；可以提供一个高层的接口以允许该访问者访问它的元素；可以是一个组合或者一个集合，如一个列表或一个无序集合。  
本题中类Library 对应着上图中的 Client ，LibraryVisitor 对应着 Visitor ， LibrarySumPrintVisitor 对应着 ConcreteVisitor。 LibraryItemInterface 对应着上图中的元素部分。下面可以结合程序代码来完成程序填空了。  
（1）和（2）空与类 LibraryVisitor  有关。由前文分析己知，  LibraryVisitor  对应着访问者模式中的  Visitor，其作用是为类LibrarySumPrintVisitor声明Visit操作。类 LibrarySumPrintVisitor 需要访问两种不同的元素，每种元素应该对应不同的 visit 操作。 再结合类 LibrarySumPrintVisitor 的定义部分，可以得知（2）和（3）处应给出分别以 Book和Article 为参数的 visit 方法。因此（1）和（2）处分别为 "void visit(Book  p\_book) "、"void visit(Article  p\_article) "。  
LibraryItemInterface 在本题中充当着 Element 的作用，其中应定义以一个访问者为参数的 Accept操作。对照实现该接口的两个子类Article  和 Book  的代码，可以得知该操作的原型是void accept(LibraryVisitor  visitor) 。由此可以得知，（3）处应填写“void accept(Library Visitor visitor)”。  
（4）和（5）处考查的是 accept 接口的实现。由访问者模式的结构图可以看出，在Book 和Article 中 accept 方法的实现均为 Visitor.visit（this）。

**试题答案**

（6）

（1）void visit(Book p\_book)

（2）void visit(Article p\_article)

（3）void accept(LibraryVisitor visitor)

（4）visitor.visit(this)

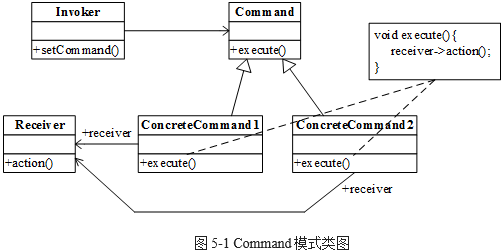
（5）visitor.visit(this)

# **试题13(2014年下半年试题5)**

    阅读下列说明和C++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，每个插槽都有开关按钮，对应着一个不同的灯。利用该遥控器能够统一控制房间中该厂商所有品牌灯具的开关，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1-1所示。



【C++代码】

class Light {

public:

  Light(string name) { /\* 代码省略 \*/ }

  void on() { /\* 代码省略 \*/ }    // 开灯

  void off() { /\* 代码省略 \*/ }  // 关灯

};

class Command {

public:

      （1）    ;

}；

class LightOnCommand:public Command { // 开灯命令

private:

  Light\* light;

public:

  LightOnCommand(Light\* light) { this->light=light; }

  void execute() {     （2）    ; }

};

class LightOffCommand:public Command { // 关灯命令

private:

  Light \*light;

public:

  LightOffCommand(Light\* light) { this->light=light; }

  void execute() {     （3）    ; }

};

class RemoteControl{ // 遥控器

private:

  Command\* onCommands[7];

  Command\* offCommands[7];

public:

  RemoteControl() { /\* 代码省略 \*/ }

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

        （4）    =onCommand;

        （5）    =offCommand;

  }

  void onButtonWasPushed(int slot) {     （6）    ; }

  void offButtonWasPushed(int slot) {     （7）    ; }

};

int main() {

  RemoteControl\* remoteControl=new RemoteControl();

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

  Light\* kitchenLight=new Light("kitchen");

  LightOnCommand\* livingRoomLightOn=new LightOnCommand(livingRoomLight);

  LightOffCommand\* livingRoomLightOff=newLightOffCommand(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);

  /\* 其余代码省略 \*/

  return 0;

}

**试题分析**

本题考察设计模式的实现，难度较小。根据类图和已有代码可写出空缺的代码，书写方式注意java和C++的区别即可。

**试题答案**

（5）

（1）virtual void execute()=0

（2）light->on()

（3）light->off()

（4）onCommands[slot]

（5）offCommands[slot]

（6）onCommands[slot]->execute()

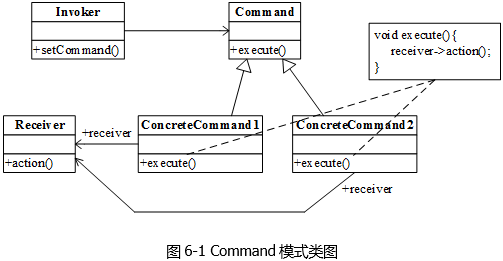
（7）offCommands[slot]->execute()

# **试题14(2014年下半年试题6)**

    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1-1所示。



【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);

  }

}

**试题分析**

本题考察设计模式的实现，难度较小。根据类图和已有代码可写出空缺的代码，第一空是Command接口的实现为interface Command，二三空分别执行开灯关灯的操作，分别为light.on()和light.off()。四五六七空为几种方法的实现，分别为onCommands[slot]、offCommands[slot]和onCommands[slot].execute()、offCommands[slot].execute()。

**试题答案**

（6）

（1）interface Command

（2）light.on()

（3）light.off()

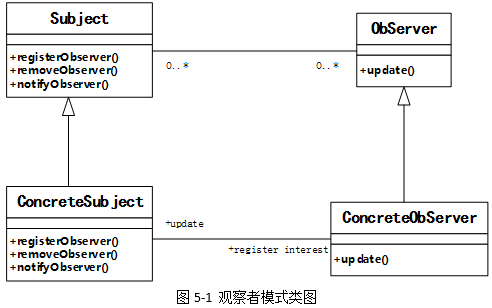
（4）onCommands[slot]

（5）offCommands[slot]

（6）onCommands[slot].execute()

（7）offCommands[slot].execute()

# **试题15(2014年上半年试题5)**

     阅读下列说明和C++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5-1所示。  
    

【C++代码】  
#include <iostream>  
#include <vector>  
using namespace std;  
class Observer {  
public:  
    virtual void update(float temp, float humidity, float cleanness)=0;  
};  
class Subject {  
public:  
    virtual void registerObserver(Observer\* o) = 0; //注册对主题感兴趣的观察者  
    virtual void removeObserver(Observer\* o) = 0; //删除观察者  
    virtual void notifyObservers() = 0;//当主题发生变化时通知观察者  
};  
class EnvironmentData : public     （1）     {  
private:  
vector<Observer\*> observers;  
float temperature, humidity, cleanness;  
public:  
    void registerObserver(Observer\* o) { observers.push\_back(o); }  
    void removeObserver(Observer\* o) { /\* 代码省略 \*/ }  
    void notifyObservers() {  
for(vector<Observer\*>::const\_iterator it = observers.begin();            it != observers.end(); it++)  
{       （2）    ; }  
}  
Void measurementsChanged() {     （3）    ; }  
void setMeasurements(float temperature, float humidity, float cleanness) {  
    this->temperature = temperature;  
    this->humidity = humidity;  
    this->cleanness = cleanness;  
        （4）    ;  
    }  
};  
class CurrentConditionsDisplay : public     （5）    {  
private:    
    float temperature, humidity, cleanness;  
    Subject\* envData;  
public:  
    CurrentConditionsDisplay(Subject\* envData) {  
        this->envData = envData;  
           （6）   ;         
}  
void update(float temperature, float humidity, float cleanness)  {this->temperature = temperature;  
    this->humidity = humidity;  
    this->cleanness = cleanness;  
    display();  
}  
void display() { /\* 代码省略 \*/ }  
};  
int main() {  
    EnvironmentData\* envData = new EnvironmentData();  
    CurrentConditionsDisplay\* currentDisplay = new CurrentConditionsDisplay(envData);  
    envData->setMeasurements(80, 65, 30.4f);  
    return 0;  
}

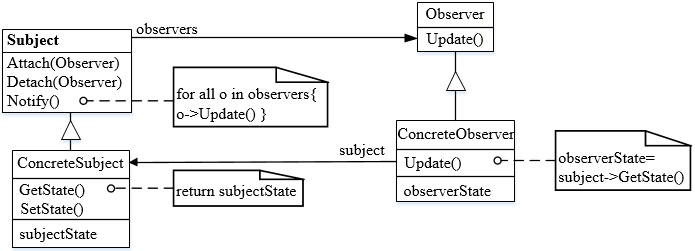
**试题分析**

EnvironmentData是环境数据，也就是我们要监测的对象，即主题（Subject），因此(1)处为Subject。

（2）处为通知观察者，因此遍历观察者容器，遍历到一个观察者对象，则更新该观察者的数据，即调用观察者的update(）方法。

当环境数据变化时，需要通知观察者，因此(4)处是调用环境变化方法measurementsChanged()，通过此方法通知观察者更新数据，因此（3）处为notifyObservers()。

根据CurrentConditionsDisplay 类中的update()方法可知：CurrentConditionsDisplay 是个观察者，因此（5）处为Observer

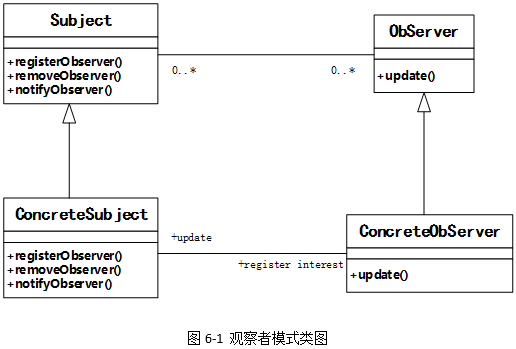
（6）是将观察者添加到主题中去。

**试题答案**

（5）

（1）Subject  
（2）(\*it)->update(temperature,humidity,cleanness)  
（3）notifyObservers()  
（4）measurementsChanged()  
（5）Observer()  
（6）this->envData->registerObserver(this)

# **试题16(2014年上半年试题6)**

    阅读下列说明和Java代码，将应填入 （n）处的字句卸载答题纸的对应栏内.  
【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。  
    

【Java代码】  
import jav（6）A．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);  
        envDat（7）A．setMeasurements(80, 65, 30.4f);  
    }  
}

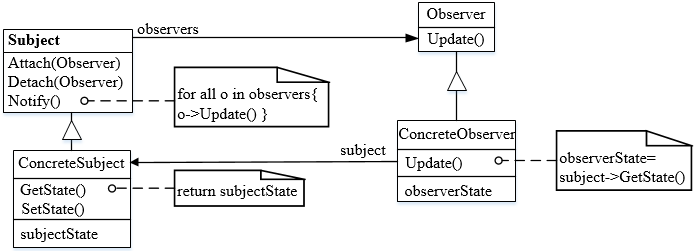
**试题分析**

EnvironmentData是环境数据，也就是我们要监测的对象，即主题（Subject），因此(1)处为Subject。

（2）处为通知观察者，因此遍历观察者容器，遍历到一个观察者对象，则更新该观察者的数据，即调用观察者的update(）方法。

当环境数据变化时，需要通知观察者，因此(4)处是调用环境变化方法measurementsChanged()，通过此方法通知观察者更新数据，因此（3）处为notifyObservers()。

根据CurrentConditionsDisplay 类中的update()方法可知：CurrentConditionsDisplay 是个观察者，因此（5）处为Observer

（6）是将观察者添加到主题中去。

**试题答案**

（6）

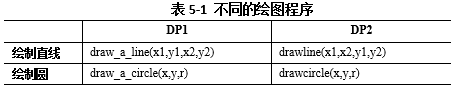
（1）Subject  
（2）observer.update(temperature,humidity,cleanness)  
（3）notifyObservers()  
（4）measurementsChanged()  
（5）Observer  
（6）envData.registerObserver(this)

# **试题17(2013年下半年试题5-8)**

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

【说明】

    欲开发一个绘图软件，要求使用不同的绘图程序绘制不同的图形。以绘制直线和圆形为例，对应的绘图程序如表5-1所示。



   该绘图软件的扩展性要求，将不断扩充新的图形和新的绘图程序。为了避免出现类爆炸的情况，现采用桥接（Bridge）模式来实现上述要求，得到如图5-1所示的类图。

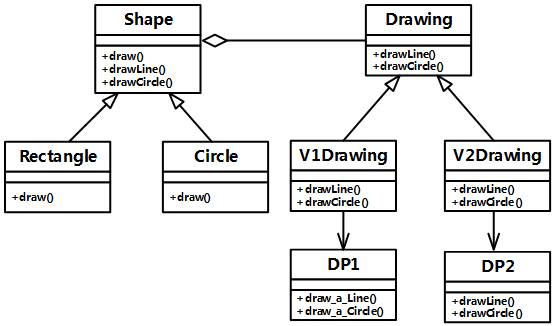


                                                                   图5-1 类图

【**C++代码**】

class  DP1{

public:

    static void draw\_a\_line(double  x1,double  y1,double  x2,double  y2){/\*代码省略\*/}

    static void draw\_a\_circle(double  x,double  y,double  r){ /\*代码省略\*/}

};

class  DP2{

public:

    static void drawline(double x1,double x2,double y1,double y2){/\*代码省略\*/}

    static void drawcircle(double x,double y,double r){/\*代码省略\*/}

};

class  Drawing{

public:

        （1）    ;

        （2）    ;

};

class V1Drawing:public Drawing{

public:

     void drawLine(double  x1,double  y1,double  x2,  double  y2){/\*代码省略\*/}

     void drawCircle(double  x, double  y, double  r){    （3）    ;}

};

class V2Drawing:public  Drawing{

public:

    void drawLine(double  x1,double  y1,double  x2,  double  y2){/\*代码省略\*/}

     void drawCircle(double  x, double   y, double  r){    （4）    ;}

};

class   Shape{

public:

       （5）   ;

    Shape(Drawing  \* dp){ \_dp = dp;}

    void drawLine(double  x1, double  y1,double  x2, double y2){\_dp->drawLine(x1,y1,x2,y2);}

    void drawCircle(double  x, double  y, double   r){\_dp->drawCircle(x,y,r);}

private:  Drawing \* \_dp;

};

class  Rectangle:public  Shape{

public:

    void draw(){/\*代码省略\*/}

//其余代码省略

};

class Circle:public Shape{

private:  double \_x,\_y,\_r;

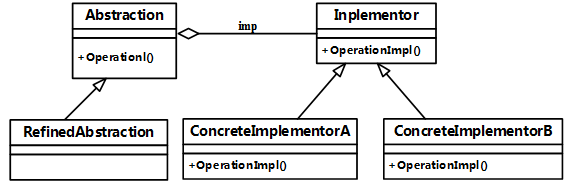
public:

     Circle(Drawing  \* dp, double   x, double  y, double   r):    （6）    {\_x = x; \_y = y; \_r = r;}

    void draw(){ drawCircle(\_x, \_y, \_r);}

};

**试题分析**

根据类图，可知：Drawing类有两个函数drawLine()、drawCircle(),在从继承于它的两个子类V1Drawing、V2Drawing中drawLine()、drawCircle()得到返回值类型为void及函数参数。则（1）为virtual void drawLine(double  x1,double  y1,double  x2,  double  y2)=0;（2）为virtual void drawCircle(double  x, double   y, double  r)=0。  
根据V1Drawing与DP1之间的关联关系，V2Drawing与DP2之间的关联关系，可知(3)是去调用DP1中的静态函数draw\_a\_circle(double  x,double  y,double  r)，则（3）为DP1::draw\_a\_circle(x,y,r)；(4)问同理，答案为DP2::drawcircle(x,y, r)。  
（5）问从类图中可知，Shape有三个方法draw()、drawLine()、drawCircle()；分析题目中给出的Shape的定义，少了一个draw()函数，可知（5）为draw()，在根据继承于Shape的Circle和Rectangle分析，函数返回值为void，参数为空，则（5）为virtual void draw() = 0。

**试题答案**

（5）

（1）virtual void drawLine(double x1 , double y1 , double x2 , double y2)=0

（2）virtual void drawCircle(double x , double y , double r)=0

（3）DP1::draw\_a\_circle(x,y,r)

（4）DP2::drawcircle(x,y,r)

（5）virtual void  draw() = 0

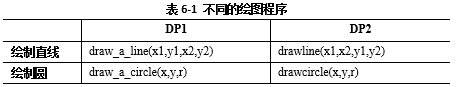
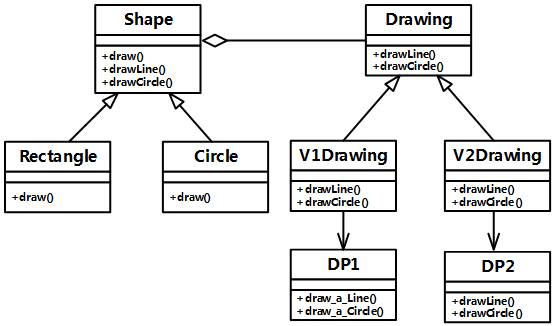
（6）Shape(dp)

# **试题18(2013年下半年试题6-9)**

    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】

    欲开发一个绘图软件，要求使用不同的绘图程序绘制不同的图形。以绘制直线和圆形为例，对应的绘图程序如表6-1所示。

  
      
    该绘图软件的扩展性要求，将不断扩充新的图形和新的绘图程序。为了避免出现类爆炸的情况，现采用桥接（Bridge）模式来实现上述要求，得到如图6-1所示的类图。  
      
                                                            图6-1 类图

【Java代码】

  （1）   Drawing{

  （2）  ;

  （3）  ;

}

class DP1{

    static public void draw\_a\_line(double x1, double y1, double x2, double y2) {/\*代码省略\*/}

    static public void draw\_a\_circle(double x, double y, double r) {/\*代码省略\*/}

}

class DP2{

    static public void drawline(double x1, double y1, double x2 ,double y2){/\*代码省略\*/}

    static public void drawcircle (double x, double y, double r){/\*代码省略\*/}

}

class V1Drawing implements Drawing{

    public void drawLine(double x1, double y1, double x2 ,double y2){/\*代码省略\*/}

    public void drawCircle(double x, double y, double r){  （4）  ;}

}

class V2Drawing implements Drawing{

    public void drawLine(double x1, double y1, double x2 ,double y2){/\*代码省略\*/}

    public void drawCircle (double x, double y, double r){  （5）  ;}

}

abstract class Shape{

    private Drawing \_dp;

      （6）  ;

    Shape(Drawing dp) {\_dp=dp;}

    public void drawLine(double x1, double y1, double x2 ,double y2){\_dp.drawLine(x1,y1,x2,y2); }

    public void drawCircle (double x, double y, double r){ \_dp.drawCircle(x,y,r);}

}

class Rectangle extends Shape{

    private double \_x1,\_x2,\_y1,\_y2;

    public Rectangle(Drawing dp,double x1, double y1, double x2 ,double y2)   {/\*代码省略\*/}

    public void draw(){/\*代码省略\*/}

}

class Circle extends Shape{

    private double \_x, \_y, \_r;

    public Circle(Drawing dp,double x, double y, double r) {/\*代码省略\*/}

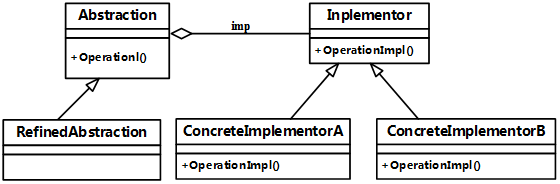
    public void draw(){drawCircle(\_x,\_y,\_r);}

}

**试题分析**

本题考查Java面向对象程序设计。这类题型主要涉及两个方面的考查内容：Java语法与设计模式。所谓语法考查是要求考生清楚接口、抽象类、抽象方法等的定义，相关意思能用正确方式表达出来。而设计模式层次的考查主要是模式间关联的建立。

       在本题中，（1）~（3）都是典型的语法问题，由于Drawing在后面被实现（implements）了，所以他是一个接口，帮（1）为：public interface。继续通过这条线索分析可知，Drawing应有drawLine和drawCircle两个方法的定义。故（2）（3）分别为：public void drawLine(double x1, double y1, double x2 ,double y2)和public void drawCircle (double x, double y, double r)。

       接下来的第（4）和（5）是同样类型的空，都需要完成函数的实现部分，此处的实现，一般不要求考生真正实现功能，而是通过别的函数完成该职能。从对应的类图可以了解到V1Drawing与DP1存在依赖关系，而DP1中有draw\_a\_circle（）的函数已完成了画圆的操作，所以在此只需要调用这个方法即可。所以（4）和（5）分别为：DP1. draw\_a\_circle(x,y,r)和DP2. drawcircle(x,y,r)。最后的（6），可以通过分析类图得知，因为类图中标明Shape类中有：draw()、drawLine()和drawCircle()三个函数，而在Shape类中缺draw()，所以需要补上该函数的定义，即abstract public void draw()。  


**试题答案**

（6）

（1）public interface

（2）public void drawLine(double x1, double y1, double x2 ,double y2)

（3）public void drawCircle (double x, double y, double r)

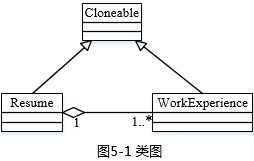
（4）DP1. draw\_a\_circle(x,y,r)

（5）DP2. drawcircle(x,y,r)

（6）abstract public void draw()

# **试题19(2013年上半年试题5-9)**

现要求实现一个能够自动生成求职简历的程序，简历的基本内容包括求职者的姓名、性别、年龄及工作经历。希望每份简历中的工作经历有所不同，并尽量减少程序中的重复代码。  
现采用原型模式（Prototype）来实现上述要求，得到如图5-1所示的类图。

  
【C++代码】  
#include<string>  
using namespace std;  
class  Cloneable{  
public:  
       (1)  ;  
};  
class  workExperience:public  Cloneable{   //工作经历  
private:  
      string  workData;  
      string  company;  
public:  
     Cloneable \* clone(){  
      (2)  ;  
     Obj->workDate= this->workDate;  
     Obj->company = this->company;  
     return Obj;  
}  
 //其余代码省略  
};

class  Resume:public  Cloneable{      //简历  
private:  
        string name;  
        string sex;  
        string age;  
        WorkExperience \* work;  
        Resume(WorkExperience \* work){  
        this->work =    (3)  ;  
        }  
public:  
        Resume(string name){/\*实现省略\*/}  
        void SetPersonInfo(string sex,string age){/\*实现省略\*/}  
        void SetWorkExperience(string workDate,string company){/\*实现省略\*/}  
        Cloneable \* Clone(){  
         (4)   ;  
        Obj->name = this->name;  
       Obj->sex = this->sex;  
       Obj->age = this->age;  
       return Obj;  
}  
};

int  main(){  
        Resume \* a  = new Resume(“张三”);  
        a->SetPersonInfo(“男”,“29”)；  
        a-> SetWorkExperience(“1998-2000”,“XXX公司”);  
        Resume \* b =    (5)   ;  
        b-> SetWorkExperience(“2001-2006”,“YYY公司”);  
        return 0;  
}

**试题分析**

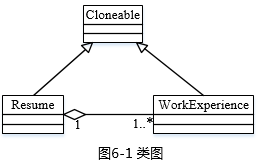
本题考查原型模型的概念及应用。  
原型模型的主要思想： 先借用已有系统作为原型模型，通过“样品”不断改进，使得最后的产品就是用户所需要的。 原型模型通过向用户提供原型获取用户的反馈，使开发出的软件能够真正反映用户的需求。同时，原型模型采用[逐步求精](http://baike.baidu.com/view/551684.htm" \t "http://www.educity.cn/jiaocheng/_blank)的方法完善原型，使得原型能够“快速”开发，避免了像[瀑布模型](http://baike.baidu.com/view/551037.htm" \t "http://www.educity.cn/jiaocheng/_blank)一样在冗长的开发过程中难以对用户的反馈作出快速的响应。相对[瀑布模型](http://baike.baidu.com/view/551037.htm" \t "http://www.educity.cn/jiaocheng/_blank)而言，原型模型更符合人们开发软件的习惯，使目前较流行的一种实用软件生存期模型。  
Prototype模式其实就是常说的“虚拟构造函数”一个实现，C++的实现机制中并没有支持这个特性，但是通过不同派生类实现的Clone接口函数可以完成与“虚拟构造函数”同样的效果。  
题中声明一个虚拟基类，所有的原型都是从这个基类继承，（1）所代表的就是这个基类中的纯虚函数，需要供继承者自行实现，即为virtual Cloneable \* Clone()=0，（1）声明一个抽象基类，并定义clone()函数为纯虚函数。然后根据基类实例化各个子类，并且实现赋复制构造函数，并实现clone()函数，由此可知（2）处为WorkExperience \* Obj，（3）处为Work，（4）处为Resume \* Obj。在main函数中实现Resume \* b对\*a的复制，故根据C++语法（5）中为a->Clone()。

**试题答案**

（5）

（1）virtual Cloneable \*Clone()=0  
（2）WorkExperience \*obj = new WorkExperience()  
（3）(WorkExperience \*) work->Clone()  
（4）Resume \*obj = new Resume(this->work)  
（6）(Resume \*)a->Clone()

# **试题20(2013年上半年试题6)**

现要求实现一个能够自动生成求职简历的程序，简历的基本内容包括求职者的姓名、性别、年龄及工作经历。希望每份简历中的工作经历有所不同，并尽量减少程序中的重复代码。  
现采用原型模式（Prototype）来实现上述要求，得到如图6-1所示的类图。  


【Java代码】  
public  class  workExperience   (1)   Cloneable{   //工作经历  
private  String  workDate;  
private  String  company;  
public  Object  clone(){  
      (2)  ;  
      Obj.workDate= this.workDate;  
      Obj.company = this.company;  
       return Obj;  
}  
  //其余代码省略  
}

public  class  Resume    (3)    Cloneable{      //简历  
private  String  name;  
private  String  sex;  
private  String  age;  
private  WorkExperience  work;

public  Resume(string name){  
        this.name = name;  
        work = new WorkExperience();  
}  
private  Resume(WorkExperience  work){  
      this.work =   (4)   ;  
}  
public  void  SetPersonInfo(string sex,string age){/\*实现省略\*/}  
public  void  SetWorkExperience(string workDate,string company){/\*实现省略\*/}  
public  Object  clone(){  
     Resume Obj =    (5)   ;  
     return Obj;  
}  
}  
  
Class WorkResume{  
public  static  void  main(){  
      Resume   a  = new Resume(“张三”);  
      a.SetPersonInfo(“男”,“29”)；  
      a.SetWorkExperience(“1998-2000”,“XXX公司”);  
      Resume  b =    (6)   ;  
      b. SetWorkExperience(“2001-2006”,“YYY公司”);  
}  
}

**试题分析**

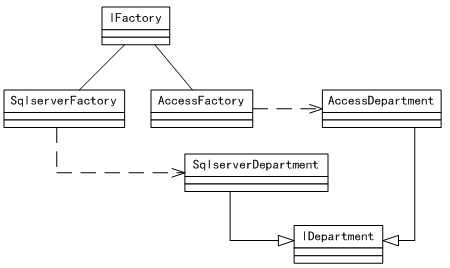
本题考查原型模型的概念及应用。  
原型模型的主要思想： 先借用已有系统作为原型模型，通过“样品”不断改进，使得最后的产品就是用户所需要的。 原型模型通过向用户提供原型获取用户的反馈，使开发出的软件能够真正反映用户的需求。同时，原型模型采用[逐步求精](http://baike.baidu.com/view/551684.htm" \t "http://www.educity.cn/jiaocheng/_blank)的方法完善原型，使得原型能够“快速”开发，避免了像[瀑布模型](http://baike.baidu.com/view/551037.htm" \t "http://www.educity.cn/jiaocheng/_blank)一样在冗长的开发过程中难以对用户的反馈作出快速的响应。相对[瀑布模型](http://baike.baidu.com/view/551037.htm" \t "http://www.educity.cn/jiaocheng/_blank)而言，原型模型更符合人们开发软件的习惯，使目前较流行的一种实用软件生存期模型。  
所有java类都继承自java.lang.Object，而object类提供一个clone()方法，可以将一个java对象复制一份。因此在java中可以直接使用Object提供的clone()方法来实现对象的克隆。能够实现克隆的java类必须实现一个标识接口Cloneable，表示这个java支持复制。  
题中WorkExperience类和Resume类需要实现Cloneable接口，故（1）和（3）为implements，WorkExperience中需要实现Clone()方法，并将自身复制一份，有下面代码可知（2）为WorkExperience obj = new WorkExperience()。Resume类中的私有构造方法实现WorkExperience深复制，故（4）中为（WorkExperience）work.clone()，而Resume类中Clone方法实现自身的复制，故（5）中为new Resume(this.work)。  
在main中实现Resume b对a的复制，故（6）中为(Resume)a.Clone()。

**试题答案**

（6）

（1）implements   
（2）WorkExperience obj = new Workexperience()  
（3）implements   
（4）(WorkExperience)work.Clone()  
（5） new Resume(this.work)  
（6）(Resume)a.Clone()

# **试题21(2012年下半年试题5)**

阅读下列说明和C++代码，将应填入(n)处的字句写在答题纸的对应栏内。  
【说明】  
现欲开发一个软件系统，要求能够同时支持多种不同的数据库，为此采用抽象工厂模式设计该系统。以SQL Server和Access两种数据库以及系统中的数据库表Department为例，其类图如图5-1所示。  
  
图5-1 类图

【C++代码】  
#include <iostream>  
using namespace std;  
   
class Department{/\*代码省略\*/}；  
class IDepartment{  
public:  
    (1)    =0；  
    (2)    =0；  
}；  
   
class SqlserverDepartment:(3){  
public:  
  void Insert(Department\* department){  
   cout <<"Insert a record into Department in SQL Server!\n";  
∥其余代码省略  
    }  
    Department GetDepartment(int id){  
   /\*代码省略\*/  
    }  
}；  
   
class AccessDepartment: (4) {  
public:  
    void Insert(Department\* department){  
   cout <<"Insert a record into Department in ACCESS!\n"；  
       ∥其余代码省略  
    }  
    Department GetDepartment(int id){  
   /\*代码省略\*/  
    }  
}；  
     
(5){  
public:  
    (6)=0；  
}；  
class SqlServerFactory:public IFactory{  
public:  
    IDepartment\*CreateDepartment(){  return new SqlserverDepartment();  }  
    ∥其余代码省略  
}；  
   
class AccessFactory:public IFactory{  
public:  
    IDepartment\* CreateDepartment(){  return new AccessDepartment();}  
    ∥其余代码省略  
}；

**试题分析**

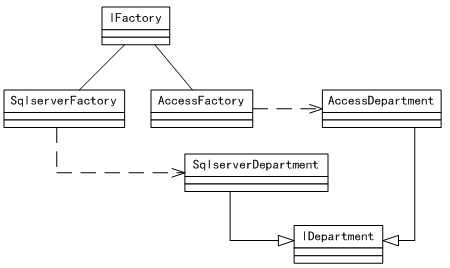
本题考查基本面向对象设计模式的运用能力。  
抽象工厂设计模式主要是提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。从题目给出的类图可知SqlserverDepartment和AccessDepartment继承于Idepartment。而从第（1）和第（2）空处的程序语句可以知道，这里是定义纯虚函数，而类Idepartment一个抽象类，而在这里需要定义一个什么样的纯虚函数，就需要根据SqlserverDepartment和AccessDepartment类的内容来了解。在这两个类里面都有Insert和GetDepartment这两个函数，因此在Idepartment类中定义的纯虚函数就是这两个函数，因此第（1）空应该填virtual void Insert(Departmet\* department)，而第（2）空应该填virlual Department GetDepartment(int id)。  
第（3）空和第（4）空是一样的，因为类SqlserverDepartment和AccessDepartment都是继承抽象类Idepartment，而一般情况下的继承方式都是public，所以这两空的答案都是public Idepartment。  
从第（5）空出现的位置，不难知道这里是定义一个类，结合前后程序，可以知道这里定义的类是Ifactory，这是一个抽象类，因此该空的答案为class Ifactory。  
第（6）空是定义抽象类Ifactory的纯虚函数，从后面的程序可以看出，需要定义的纯虚函数是CreateDepartment，因此第（6）空的答案是virtual IDcpartment\* CreateDepartment()。

**试题答案**

（5）

(1)  virtual void Insert(Departmet\* department)  
(2)  virlual Department GetDepartment(int id)  
(3)  public IDepartment  
(4)  public IDepartmcnt  
(5)  class Ifactory  
(6)  virtual IDcpartment\* CreateDepartment()

# **试题22(2012年下半年试题6)**

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】  
现欲开发一个软件系统，要求能够同时支持多种不同的数据库，为此采用抽象工厂模式设计该系统。以SQL Server和Access两种数据库以及系统中的数据库表Department为例，其类图如图6-1所示。  
  
图6-1 类图

【Java代码】  
import jav（6）A．util.\*;  
class Department{/\*代码省略\*/}  
   
interface IDepartment{  
    (1)    ；  
    (2)    ；  
}  
   
class SqlserverDepartment (3) {  
    public voidInsert(Department department){  
       System.out.println(”Insert a record into Department in SQL Server!");  
       ∥其余代码省略  
    }  
    public Department GetDepartment(int id){  
  /\*代码省略\*/  
    }  
}  
   
classAccessDepartment(4) {  
    public void Insert(Department department){  
   System.out.println("Insert a record into Department in ACCESS!”);  
   ∥其余代码省略  
    }  
    public Department GetDepartment(int id){  
       /\*代码省略\*/  
    }  
}  
(5)  {  
    (6)   ；  
}  
   
class SqlServerFactory implements IFactory{  
    public IDepartment CreateDepartment(){  
  retum new SqlserverDepartment();  
    }  
    ∥其余代码省略  
}  
class AccessFactory implements IFactory{  
    public IDepartment CreateDepartment(){  
  return new AccessDepartment();  
    }  
    ∥其余代码省略  
}

**试题分析**

本题考查基本面向对象设计模式的运用能力。  
抽象工厂设计模式主要是提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。从题目给出的类图可知SqlserverDepartment和AccessDepartment继承于接口Idepartment。而从第（1）和第（2）空处的程序语句可以知道，这里是定义抽象函数，但在这里需要定义一个什么样的抽象函数，就需要根据SqlserverDepartment和AccessDepartment类的内容来了解。在这两个类里面都有Insert和GetDepartment这两个函数，因此在Idepartment中定义的抽象函数就是这两个函数，因此第（1）空应该填 void Insert(Departmet department)，而第（2）空应该填Department GetDepartment(int id)。  
第（3）空和第（4）空是一样的，因为类SqlserverDepartment和AccessDepartment都是实现接口Idepartment，而实现接口都是用关键字implements，所以这两空的答案都是implements Idepartment。  
从第（5）空出现的位置，不难知道这里是定义一个接口，结合前后程序，可以知道这里定义的接口是Ifactory，因此该空的答案为interface Ifactory。  
第（6）空是定义接口Ifactory的抽象函数，从后面的程序可以看出，需要定义的抽象函数是CreateDepartment，因此第（6）空的答案是Idcpartment CreateDepartment()。

**试题答案**

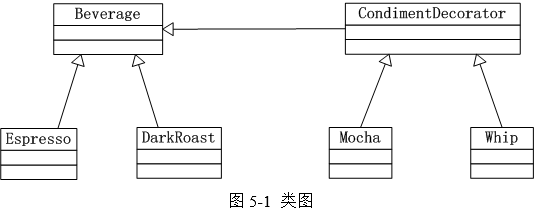
（6）

(1)  void Insert(Department department)  
(2)  Department GetDepartment(int id)  
(3)  implements lDepartment  
(4)  implements IDepartment  
(5)  interface IFactory  
(6)  IDepartment CreateDepartment()

# **试题23(2012年上半年试题5)**

阅读下列说明和C++代码，将应填入（n）处的字句写在答题纸的对应栏内。  
【说明】  
某咖啡店当卖咖啡时，可以根据顾客的要求在其中加入各种配料，咖啡店会根据所加入的配料来计算费用。咖啡店所供应的咖啡及配料的种类和价格如下表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 咖啡 | 价格/杯 | 配料 | 价格/份 |
| 蒸馏咖啡（Espresso） | 25 | 摩卡（Mocha） | 10 |
| 深度烘焙咖啡（DarkRoast） | 20 | 奶泡（Whip） | 8 |

现采用装饰器（Decorator）模式来实现计算费用的功能，得到如图5-1所示的类图  


【C++代码】  
#include <iostream>  
#include <string>  
using namespace std;  
const int ESPRESSO\_PRICE = 25;  
const int DRAKROAST\_PRICE = 20;  
const int MOCHA\_PRICE = 10;  
const int WHIP\_PRICE = 8;  
class Beverage {  //饮料  
   （1） ：string description;  
public：  
   （2） ( ){ return description; }  
   （3） ;  
};  
class CondimentDecorator : public Beverage {   //配料  
protected:  
    （4）  ;  
};  
class Espresso : public Beverage {  /／蒸馏咖啡  
public:  
Espresso ( ) {description="Espresso"; }  
int cost ( ){return ESPRESSO\_PRICE; }  
};  
class DarkRoast : public Beverage { //深度烘焙咖啡  
public:  
    DarkRoast( ){ description = "DardRoast"; }  
    int cost( ){ return DRAKROAST\_PRICE; }  
 };  
class Mocha : public CondimentDecorator {  ／／摩卡  
public:  
    Mocha（Beverage\*beverage）{  this->beverage=beverage; }  
    string getDescription( ){ return beverage->getDescription( )+"，Mocha"; }  
    int cost( ){ return MOCHA\_PRICE+beverage->cost( ); }  
 };  
class Whip :public CondimentDecorator {   //奶泡  
public:  
    Whip（Beverage\*beverage） { this->beverage=beverage; }  
    string getDescription( ) {return beverage->getDescription( )+"，Whip"; }  
    int cost( ) { return WHIP\_PRICE+beverage->cost( ); }  
  };  
   
int main() {  
    Beverage\* beverage = new DarkRoast( );  
    beverage=new Mocha( （5） ）;  
    beverage=new Whip（ （6） ）;  
cout<<beverage->getDescription ( )<<"￥"<<beverage->cost( ) endl;  
    return 0;  
 }  
编译运行上述程序，其输出结果为：  
DarkRoast， Mocha， Whip ￥38

**试题分析**

    本题考查了C++语言的应用能力和装饰设计模式的应用。  
    第（1）空很明显，是要说明属性description在类Beverage中的类型，应该是私有的、受保护的或公有的，从后面的程序我们可以看出，子类中继承使用了该属性，因此这里只能定义为受保护的，因此第（1）空的答案为protected。  
    第（2）空处也很明显，是要给出一个函数的定义，并且该函数的函数体是“return description;”，从子类奶泡和摩卡中我们不难发现这个函数应该是getDescription，因此本空的答案为virtual string getDescription。  
    第（3）空需要结合后面各子类才能发现，在Beverage中还应该定义一个函数cost()，而这个函数在Beverage中并没有实现，因此要定义为纯虚函数，所以第（3）空的答案为virtual int cost()=0。  
    第（4）空在类CondimentDecorator中，且是该类唯一的一条语句，而他的子类分别是奶泡和摩卡，在奶泡和摩卡这两个类中，都用到了Beverage\*beverage，而在使用之前并没有说明，因此这就可以说明，Beverage\*beverage是在父类CondimentDecorator中定义的，子类直接继承使用，因此第（4）空的答案为Beverage\*beverage。  
    第（5）和第（6）空在主函数当中，其中第（5）空是要创建一个Mocha对象，应该调用的是类Mocha的构造函数，从类Mocha中，我们可以看出，其构造函数Mocha的参数是一个Beverage类型的对象指针，而在主函数中，开始就定义了一个Beverage类型的对象指针beverage，因此这里只需填写beverage即可。同理第（6）空的答案也是beverage。

**试题答案**

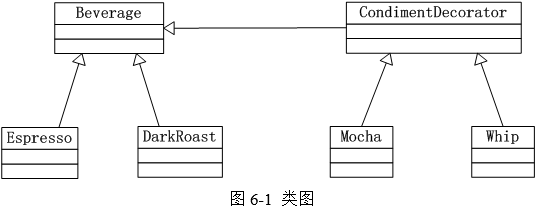
（5）

(1) protected  
(2) virtual string getDescription  
(3) virtual int cost()=0  
(4) Beverage\*beverage  
(5) beverage  
(6) beverage

# **试题24(2012年上半年试题6)**

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】  
某咖啡店当卖咖啡时，可以根据顾客的要求在其中加入各种配料，咖啡店会根据所加入的配料来计算费用。咖啡店所供应的咖啡及配料的种类和价格如下表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 咖啡 | 价格/杯 | 配料 | 价格/份 |
| 蒸馏咖啡（Espresso） | 25 | 摩卡（Mocha） | 10 |
| 深度烘焙咖啡（DarkRoast） | 20 | 奶泡（Whip） | 8 |

现采用装饰器（Decorator）模式来实现计算费用的功能，得到如图6-1所示的类图  


【Java代码】  
import jav（6）A．util.\*;  
   （1） class Beverage {    //饮料  
    String description = "Unknown Beverage";  
    public   （2）  (){return description;}  
    public   （3）  ;  
}  
  
abstract class CondimentDecorator extends Beverage {  //配料  
   （4）  ;  
}  
   
class Espresso extends Beverage {    //蒸馏咖啡  
    private final int ESPRESSO\_PRICE = 25;  
    public Espresso() {   description="Espresso";  }  
    public int cost() {   return ESPRESSO\_PRICE;   }  
}  
   
class DarkRoast extends Beverage {  //深度烘焙咖啡  
    private finalint DARKROAST\_PRICE = 20;  
    public DarkRoast0 { description = "DarkRoast";   }  
    public int cost(){ rcturn DARKROAST PRICE;   }  
}  
class Mocha extends CondimentDecorator {  //摩卡  
    private final int MOCHA\_PRICE = 10;  
    public Mocha（Beverage beverage） {  
    this.beverage = beverage;  
}  
   
    public String getDescription() {  
    return beverage.getDescription0 + "， Mocha";  
}  
    public int cost() {  
    return MOCHA\_PRICE + beverage.cost();  
    }  
 }  
   
class Whip extends CondimentDecorator {   //奶泡  
    private finalint WHIP\_PRICE = 8;  
    public Whip（Beverage beverage） { this.beverage = beverage; }  
    public String getDescription() {  
    return beverage.getDescription()+"， Whip";  
}  
    public int cost() { return WHIP\_PRICE + beverage.cost(); }  
}  
  
public class Coffee {  
    public static void main(String args[]) {  
    Beverage beverage = new DarkRoast();  
    beverage=new Mocha(  (5)  );  
beverage=new Whip (  (6)  ) ;  
    System.out.println(beverage.getDescription0 +"￥" +beverage.cost());  
}  
}  
编译运行上述程序，其输出结果为：  
DarkRoast， Mocha， Whip ￥38

**试题分析**

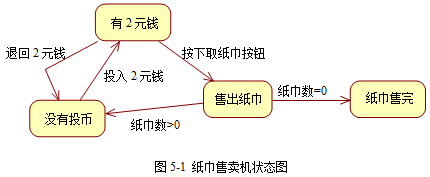
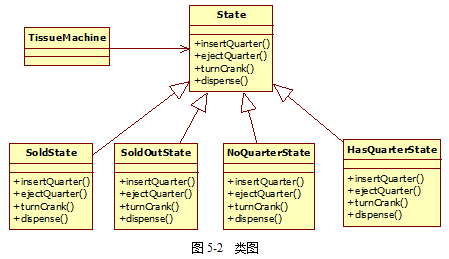
    本题考查了Java语言的应用能力和装饰设计模式的应用。  
    第（1）空很明显，是要给类Beverage前添加定义的关键字，从整个程序来看，我们应该要将类Beverage定义为抽象类，需要在前面添加关键字abstract，因此第（1）空的答案为abstract。  
    第（2）空处也很明显，是要给出一个函数的定义，并且该函数的函数体是“return description;”，从子类奶泡和摩卡中我们不难发现这个函数应该是getDescription，而该函数的返回类型String，因此本空的答案为String getDescription。  
    第（3）空需要结合后面各子类才能发现，在Beverage中还应该定义一个函数cost()，而这个函数在Beverage中并没有实现，因此要定义为抽象函数，所以第（3）空的答案为abstract int cost()=0。  
    第（4）空在类CondimentDecorator中，且是该类唯一的一条语句，而他的子类分别是奶泡和摩卡，在奶泡和摩卡这两个类中，都用到了Beverage beverage，而在使用之前并没有说明，因此这就可以判定，Beverage beverage是在父类CondimentDecorator中定义的，子类直接继承使用，因此第（4）空的答案为Beverage beverage。  
    第（5）和第（6）空在主函数当中，其中第（5）空是要创建一个Mocha对象，应该调用的是类Mocha的构造函数，从类Mocha中，我们可以看出，其构造函数Mocha的参数是一个Beverage类型的对象引用，而在主函数中，开始就定义了一个Beverage类型的对象引用beverage，因此这里只需填写beverage即可。同理第（6）空的答案也是beverage。

**试题答案**

（6）

(1) abstract  
(2) String getDescription  
(3) abstract int cost()  
(4) Beverage beverage  
(5) beverage  
(6) beverage

# **试题25(2011年下半年试题5)**

【说明】  
某大型商场内安装了多个简易的纸巾售卖机，自动出售2元钱一包的纸巾，且每次仅售出一包纸巾。纸巾售卖机的状态图如图5-1所示。  
  
采用状态（State）模式来实现该纸巾售卖机，得到如图5-2所示的类图。其中类State为抽象类，定义了投币、退币、出纸巾等方法接口。类SoldState、SoldOutState、NoQuarterState和HasQuarterState分别对应图5-1中纸巾售卖机的4种状态：售出纸巾、纸巾售完、没有投币、有2元钱。  


【C++代码】  
#include <iostream>  
using namespace std;  
// 以下为类的定义部分  
class TissueMachine;   // 类的提前引用  
class State {  
public:  
   virtual void insertQuarter() = 0;  //投币  
   virtual void ejectQuarter() = 0;    //退币  
   virtual void turnCrank()= 0;   //按下“出纸巾”按钮  
   virtual void dispense() = 0;    //出纸巾  
};  
/\* 类SoldOutState、NoQuarterState、HasQuarterState、SoldState的定义省略，每个类中均  
定义了私有数据成员TissueMachine\* tissueMachine; \*/  
class TissueMachine {  
private:  
     （1）   \*soldOutState， \*noQuarterState， \*hasQuarterState，\*soldState， \*state ;  
   int count; //纸巾数  
public:  
  TissueMachine(int numbers);  
  void setState(State\* state);  
  State\* getHasQuarterState();  
  State\* getNoQuarterState();  
  State\* getSoldState();  
  State\* getSoldOutState();  
  int getCount();  
  // 其余代码省略  
};  
// 以下为类的实现部分  
void NoQuarterState ::insertQuarter() {  
    tissueMachine->setState(  （2）  );  
}  
void HasQuarterState ::ejectQuarter() {  
    tissueMachine->setState(  （3）  );  
}  
void SoldState ::dispense() {  
    if(tissueMachine->getCount() > 0) {  
  tissueMachine->setState(  （4）  );  
    }  
    else {  
  tissueMachine->setState(  （5）  );  
    }  
}  // 其余代码省略

**试题分析**

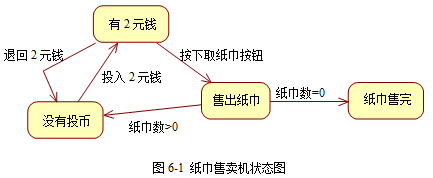
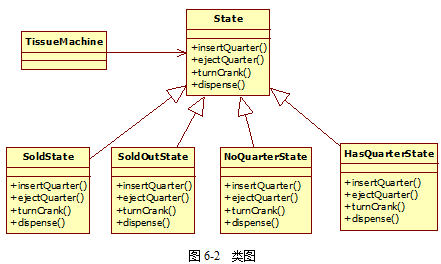
    本题考查基本面向对象设计模式的运用能力。  
    状态设计模式主要是能够使一个对象的内在状态改变时允许改变其行为，使这个对象看起来像是改变了其类。由类图可知类State是类SoldState、SoldOutState、NoQuarterState和HasQuarterState分的父类，它抽象了这四个类的共有属性和行为。在使用中，无论是这四个类中那个类的对象，都可被当作State对象来使用。  
    而根据题目的描述，我们可以知道一个纸巾售卖机它由4种状态，分别是售出纸巾、纸巾售完、没有投币、有2元钱。  
    在本题中，根据程序我们不难知道第（1）空是要定义5个对象指针，而这些对象指针都应该属于State类型，因此第一空答案为State。  
    而第（2）在类NoQuarterState（没有投币）的insertQuarter()函数中，而这个函数是投币函数，在该函数中，使用了tissueMachine类的setState方法，该方法是设置纸巾售卖机的当前状态，根据题目给出的纸巾售卖机状态图，我们可以知道，从没有投币状态，经过投币后，应该转换到有2元钱状态。而setState方法的参数是一个State的对象，因此第（2）空应该是一个有2元钱对象，而这里我们可以新创建一个该对象，也可以通过tissueMachine类的getHasQuarterState方法来获得这样一个对象，所以第（2）空答案应该是“tissueMachine->getHasQuarterState()”或“new HasQuarterState”。  
    而第（3）在类HasQuarterState（有2元钱）的ejectQuarter()函数中，而这个函数是退币函数，在该函数中，也使用了tissueMachine类的setState方法，该方法是设置纸巾售卖机的当前状态，根据题目给出的纸巾售卖机状态图，我们可以知道，从有2元钱状态，经过退币后，应该转换到没有投币状态。而setState方法的参数是一个State的对象，因此第（3）空应该是一个没有投币对象，而这里我们可以新创建一个该对象，也可以通过tissueMachine类的getNoQuarterState方法来获得这样一个对象，所以第（3）空答案应该是“tissueMachine-> getNoQuarterState()”或“new NoQuarterState”。  
    而同样的道理，我们可以知道第（4）空的答案是“tissueMachine->getNoQuarterState()”或“new NoQuarterState”。第（5）空的答案是“tissueMachine->getSoldOutState()”或“new SoldOutState”。

**试题答案**

（5）

（1）State  
（2）tissueMachine->getHasQuarterState()  
（3）tissueMachine->getNoQuarterState()  
（4）tissueMachine->getNoQuarterState()  
（5）tissueMachine->getSoldOutState()

# **试题26(2011年下半年试题6)**

【说明】  
某大型商场内安装了多个简易的纸巾售卖机，自动出售2元钱一包的纸巾，且每次仅售出一包纸巾。纸巾售卖机的状态图如图6-1所示。  
  
采用状态（State）模式来实现该纸巾售卖机，得到如图6-2所示的类图。其中类State为抽象类，定义了投币、退币、出纸巾等方法接口。类SoldState、SoldOutState、NoQuarterState和HasQuarterState分别对应图6-1中纸巾售卖机的4种状态：售出纸巾、纸巾售完、没有投币、有2元钱。  


【Java代码】  
import jav（6）A．util.\*;  
interface State {  
  public void insertQuarter();    //投币  
  public void ejectQuarter();    //退币  
  public void turnCrank();   //按下“出纸巾”按钮  
  public void dispense();   //出纸巾  
}  
class TissueMachine {  
    （1）   soldOutState， noQuarterState， hasQuarterState， soldState， state;  
  state = soldOutState;  
  int count = 0;    //纸巾数  
  public TissueMachine(int numbers) {  /\* 实现代码省略 \*/ }  
  public State getHasQuarterState()  {  return hasQuarterState;  }  
  public State getNoQuarterState()   {  return noQuarterState;  }  
  public State getSoldState()    {  return soldState;   }  
  public State getSoldOutState() {  return soldOutState;    }  
  public int getCount()   {  return count;  }  
// 其余代码省略  
}  
   
class NoQuarterState implements State {  
   TissueMachine tissueMachine;  
  public void insertQuarter() {  
    tissueMachine.setState(  （2）  );  
  }   
//构造方法以及其余代码省略  
}  
class HasQuarterState implements State {  
   TissueMachine tissueMachine;  
  public void ejectQuarter() {   
    tissueMachine.setState(  （3）  );  
  }  
//构造方法以及其余代码省略  
}  
class SoldState implements State {  
  TissueMachine tissueMachine;  
  public void dispense() {  
    if(tissueMachine.getCount() > 0)  {  
 tissueMachine.setState(  （4）  );  
    } else {  
  tissueMachine.setState(  （5）  );  }  
    }  
}

**试题分析**

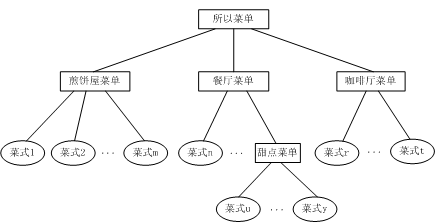
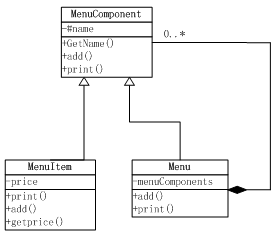
    本题考查基本面向对象设计模式的运用能力。  
    状态设计模式主要是能够使一个对象的内在状态改变时允许改变其行为，使这个对象看起来像是改变了其类。由类图可知类State是类SoldState、SoldOutState、NoQuarterState和HasQuarterState分的父类，它抽象了这四个类的共有属性和行为。在使用中，无论是这四个类中那个类的对象，都可被当作State对象来使用。  
    而根据题目的描述，我们可以知道一个纸巾售卖机它由4种状态，分别是售出纸巾、纸巾售完、没有投币、有2元钱。  
    在本题中，根据程序我们不难知道第（1）空是要定义5个对象的引用，而这些变量都应该属于State类型，因此第一空答案为State。  
    而第（2）在类NoQuarterState（没有投币）的insertQuarter()函数中，而这个函数是投币函数，在该函数中，使用了tissueMachine类的setState方法，该方法是设置纸巾售卖机的当前状态，根据题目给出的纸巾售卖机状态图，我们可以知道，从没有投币状态，经过投币后，应该转换到有2元钱状态。而setState方法的参数是一个State的对象，因此第（2）空应该是一个有2元钱对象，而这里我们可以新创建一个该对象，也可以通过tissueMachine类的getHasQuarterState方法来获得这样一个对象，所以第（2）空答案应该是“tissueMachine.getHasQuarterState()”或“new HasQuarterState”。  
    而第（3）在类HasQuarterState（有2元钱）的ejectQuarter()函数中，而这个函数是退币函数，在该函数中，也使用了tissueMachine类的setState方法，该方法是设置纸巾售卖机的当前状态，根据题目给出的纸巾售卖机状态图，我们可以知道，从有2元钱状态，经过退币后，应该转换到没有投币状态。而setState方法的参数是一个State的对象，因此第（3）空应该是一个没有投币对象，而这里我们可以新创建一个该对象，也可以通过tissueMachine类的getNoQuarterState方法来获得这样一个对象，所以第（3）空答案应该是“tissueMachine. getNoQuarterState()”或“new NoQuarterState”。  
    而同样的道理，我们可以知道第（4）空的答案是“tissueMachine.getNoQuarterState()”或“new NoQuarterState”。第（5）空的答案是“tissueMachine.getSoldOutState()”或“new SoldOutState”。

**试题答案**

（6）

（1）State  
（2）tissueMachine.getHasQuarterState()  
（3）tissueMachine.getNoQuarterState()  
（4）tissueMachine.getNoQuarterState()  
（5）tissueMachine.getSoldOutState()

# **试题27(2011年上半年试题5)**

阅读下列说明和C++代码，将应填入空(n)处的字句写在答题纸的对应栏内。  
【说明】     
    某饭店在不同的时段提供多种不同的餐饮，其菜单的结构图如下图所示。  
   
    现在采用组合(Composition)模式来构造该饭店的菜单，使得饭店可以方便地在其中增加新的餐饮形式，得到如下图所示的类图。其中MenuComponent为抽象类，定义了添加(add)新菜单和打印饭店所有菜单信息(print)的方法接口。类Menu表示饭店提供的每种餐饮形式的菜单，如煎饼屋菜单、咖啡屋菜单等。每种菜单中都可以添加子菜单，例如图中的甜点菜单。类MenuItem表示菜单中的菜式。  


（）  
  
【C++代码】  
#include<iostream>  
#include<list>  
#include <string>  
using namespace std;  
class MenuComponent{  
protected:  string name;  
public:  
  MenuComponent(string name){  this->name= name;)  
  string getName(){ return name;)     
    (1)   ;   //添加新菜单     
  virtual void print()=0；//打印菜单信息  
     
}；     
class Menultem: public MenuComponent{  
private:double price;  
public:  
    Menultem(string name， double price):MenuComponent(name){ this->price= price;  
    double getPrice(){ return price;)  
    void add(MenuComponent\* menuComponent){ retum；}//添加新菜单  
    void print(){ cout<<"  "  <<getName0<<"，"<<getPrice0<<endl;)  
}；  
class Menu:public MenuComponent{  
private:    list<  (2)  >  menuComponents;  
public:  
  Menu(string name):  MenuComponent(name){}  
  void add(MenuComponent\* menuComponent) //添加新菜单  
  {  (3)  ；    }  
  void print(){  
  cout<<"\n"<<getNameO<<"\n-------------------"<<endl;  
  std::list<MenuComponent \*>::iterator iter，  
  for(iter= menuComponents.begin0; iter！=menuComponents.end0; iter++)  
    (4) ->print();  
    }  
}；  
void main0{  
  MenuComponent\* alIMenus= new Menu("ALL MENUS");  
  MenuComponent\* dinerMenu= new Menu("DINER MENU");  
  ……//创建更多的Menu对象，此处代码省略  
  alIMenus->add(dinerMenu);  //将dinerMenu添加到餐厅菜单中  
  ……//为餐厅增加更多的菜单，此处代码省略  
 (5) ->printO;  //打印饭店所有菜单的信息

**试题分析**

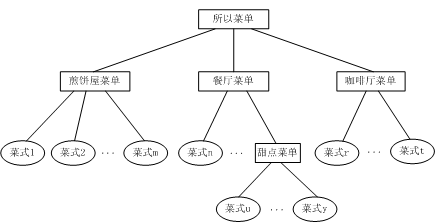
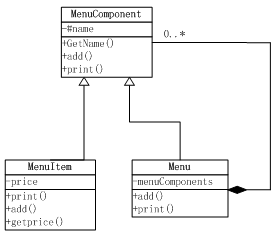
    本题考查基本面向对象设计模式的运用能力。  
    组合设计模式主要是表达整体和部分的关系，并且对整体和部分对象的使用无差别。由UML结构图知MenuComponent是MenuItem类和Menu类的父类，它抽象了两个类的共有属性和行为。在使用中，无论是MenuItem对象还是Menu对象，都可被当作MenuComponent对象来使用。另外由UML结构图可知，类MenuComponent和Menu之间存在共享关系，即Menu对象可以共享其它的Menu对象和MenuItem对象。  
    第（1）空是要在类MenuComponent中定义添加新菜单的方法，即add方法，而类MenuComponent是抽象类，因此该方法也应该是抽象方法，及需要加关键字virtual，因此空（1）应填virtual void add(MenuComponent\* menuComponent)=0，而第（2）空很明显应该填MenuComponent\*；空（3）是要实现Menu类下的add函数，而根据题目描述“每种菜单中都可以添加子菜单”，因此这里要实现添加子菜单，因此本空应填menuComponents.push\_back(menuComponent)。在类Menu类下的print函数中，定义了对象iter，因此第4空可以通过这个对象来引用print()方法，因此第（4）空答案应该填(\*iter)。而第5空为了能打印饭店所有菜单的信息，应该填alIMenus。

**试题答案**

（5）

(1)  virtual void add(MenuComponent\* menuComponent)=0     
(2)  MenuComponent\*     
(3)  menuComponents.push\_back(menuComponent)     
(4)  (\*iter)  
(5)  allMenus

# **试题28(2011年上半年试题6)**

 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】  
    某饭店在不同的时段提供多种不同的餐饮，其菜单的结构图如下图所示。  
   
   
    现在采用组合( Composition)模式来构造该饭店的菜单，使得饭店可以方便地在其中增加新的餐饮形式，得到如下图所示的类图。其中MenuComponent为抽象类，定义了添加(add)新菜单和打印饭店所有菜单信息(print)的方法接口。类Menu表示饭店提供的每种餐饮形式的菜单，如煎饼屋菜单、咖啡屋菜单等。每种菜单中都可以添加子菜单，例如图中的甜点菜单。类Menultem表示菜单中的菜式。   
 

【Java代码】  
import jav（6）A．util.\*;  
 (1)  MenuComponent{  
  protected String name;  
  (2) ;//添加新菜单  
  public abstract void print();    //打印菜单信息  
  public String getName(){ return name;}  
}  
class Menultem extends MenuComponent{  
    private double price;  
    public MenuItem(String name, double price){  
     this.name= name;  this.price= price;  
   }  
    public double getPrice(){return price;)   
    public void add(MenuComponent menuComponent){ return;）//添加新菜单  
    public void print(){  
      System.out.print(" "+ getName());     
      System.out.println(","+ getPrice());  
    }  
}  
class Menu extends MenuComponent{    
    private List<MenuComponent> menuComponents= new ArrayList<MenuComponent>();  
    public Menu(String name){ this.name= name;)  
    public void add(MenuComponent menuComponent){//添加新菜单  
  menuComponents. (3);  
}  
    public void print(){  
    System.out.print("\n"+ getName());  
    System.out.println(","+"---------------");  
    Iterator iterator = menuComponents.iterator();  
    while(iterator.hasNext()){  
   MenuComponent menuComponent= (MenuComponent)iterator.next();  
         (4) ;  
     }  
    }  
}  
class MenuTestDrive{  
  public static void main(String args[]){  
     MenuComponent   aIIMenus= new Menu("ALL MENUS");  
     MenuComponent   dinerMenu = new Menu("DINER MENU”);  
     ……//创建更多的Menu对象，此处代码省略  
    allMenus.add(dinerMenu);   //将dinerMenu添加到餐厅菜单中  
    ……//为餐厅增加更多的菜单，此处代码省略  
      (5);   //打印饭店所有菜单的信息  
    }  
 }

**试题分析**

    本题考查基本面向对象设计模式的运用能力。  
    组合设计模式主要是表达整体和部分的关系，并且对整体和部分对象的使用无差别。由UML结构图知MenuComponent是MenuItem类和Menu类的父类，它抽象了两个类的共有属性和行为。在使用中，无论是MenuItem对象还是Menu对象，都可被当作MenuComponent对象来使用。另外由UML结构图可知，类MenuComponent和Menu之间存在共享关系，即Menu对象可以共享其它的Menu对象和MenuItem对象。  
    第（1）空是要在类MenuComponent定义前加一个关键字，而类MenuComponent是抽象类，因此空（1）应填abstract class或public abstract class，而第（2）空是要定义忝加新菜单的方法，该方法应该是抽象方法，因此第（2）空的答案应该是public abstract void add(MenuComponent menuComponent)；很明显应该填MenuComponent\*；空（3）是要实现Menu类下的add函数，而根据题目描述“每种菜单中都可以添加子菜单”，因此这里要实现添加子菜单，因此本空应填add(menuComponent)。在空（4）需要调用print()方法，因此空（4）处应填menuComponent.print()。而第5空为了能打印饭店所有菜单的信息，应该填aIIMenus.print()。

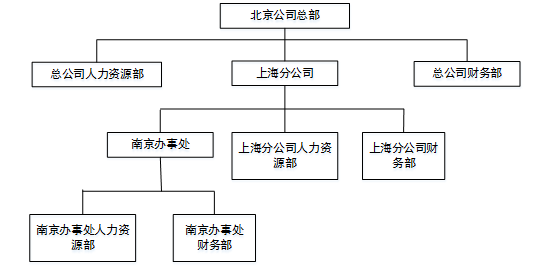
**试题答案**

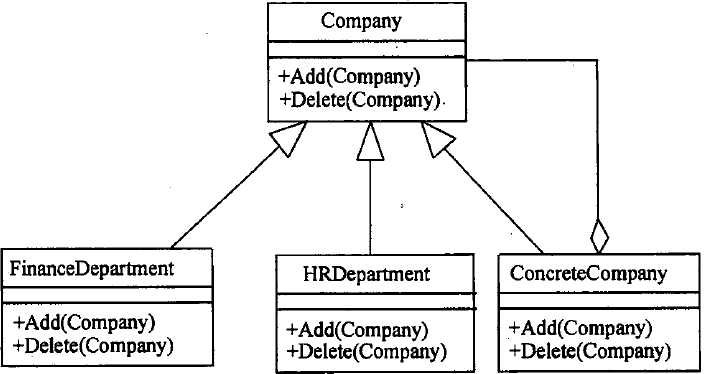
（6）

(1)  abstract class或public abstract class     
(2) public abstract void add(MenuComponent menuComponent)  
或abstract void add(MenuComponent menuComponent)  
或protected abstract void add(MenuComponent menuComponent)     
(3) add(menuComponent)     
(4) menuComponent.print()  
(5) aIIMenus.print()

# **试题29(2010年下半年试题5)**

阅读下列说明和C++代码，将应填入（n）处的字句写在答题纸的对应栏内。  
【说明】  
某公司的组织结构图如图5-1所示，现采用组合（Composition）设计模式来构造该公司的组织结构，得到如图5-2所示的类图。



**图5-1  组织结构图  
**

**图5-2  类图**

其中Company为抽象类，定义了在组织结构图上添加（Add）和删除（Delete）分公司/办事处或者部门的方法接口。类ConcreteCompany表示具体的分公司或者办事处，分公司或办事处下可以设置不同的部门。类HRDepartment和FinanceDepartment分别表示人力资源部和财务部。

 【C++代码】

#include <iostream>  
#include <list>  
#include <string>  
using namespace std;  
class  Company   {//抽象类  
protected:  
     strìng name;  
public:  
    Company(string  name)   {（1）= name;}  
        （2）; //增加子公司、办事处或部门  
        （3）;//删除子公司、办事处或部门  
};  
class ConcreteCompany: public Company  {  
private:  
     list<（4）>children; //存储子公司、办事处或部门  
public:  
   ConcreteCompany(string  name) : Company(name)  {  }  
   void  Add(Company\* c)  { （5）.push\_back(c); }  
   void  Delete(Company\* c) {  （6）.remove(c); }  
};  
   
class  HRDepartment : public  Company   {  
    public:  
       HRDepartment(string name) : Company(name)  { }    //其他代码省略  
};  
   
class  FinanceDepartment : public  Company  {  
    public:  
      FinanceDepartment(string name) : Company(name)  { } //其他代码省烙  
};  
   
void  main()  {  
   ConcreteCompany \*root = new ConcreteCompany("北京总公司") ;  
   root->Add(new HRDepartment("总公司人力资源部") ) ;  
   root->Add(new FinanceDepartment("总公司财务部") ) ;  
   
   ConcreteCompany \*comp = new ConcreteCompany("上海分公司") ;  
   comp->Add(new HRDepartment("上海分公司人力资源部") ) ;  
   comp->Add(new FinanceDepartment("上海分公司财务部") ) ;  
   （7）;  
   
   ConcreteCompany \*compl = new ConcreteCompany("南京办事处") ;  
   compl->Add(new HRDepartment("南京办事处人力资源部") ) ;  
   compl->Add(new FinanceDepartment("南京办事处财务部") ) ;  
   （8）;      //其他代码省略  
}

**试题分析**

本题考查基本面向对象设计模式的运用能力。  
组合设计模式主要是表达整体和部分的关系，并且对整体和部分对象的使用无差别。由类图知Company是ConcreteCompany类、HRDepartment 类和FinanceDepartment类的父类，它抽象了三个类的共有属性和行为。  
第（1）空是在构造函数中，被赋值为name，而name是构成函数所带的参数，那么这里是给类的一个属性name赋值，因此这空答案为：this->name。  
第（2）空与第（3）空我们可以根据注释来完成，根据题目的描述，这里只提供接口，是虚方法，因此第（2）空与第（3）空分别应该为virtual void Add(Company\* c) = 0和virtual void Delete(Company\* c) = 0，这两个方法的参数可以从后面类的相应方法中获得。  
第（4）空根据注释可以推导出应该填Company\*。第（5）空与第（6）空的答案应该一致，都应该为children。  
第（7）空和第（8）空在main函数中，用来创建组件结构图，根据题目提供的图，我们可以知道，创建了上海分公司接的后，应该将其添加至root下，因此第7空答案为root->Add(comp)，同样的道理，第（8）空的答案为comp->Add(comp1)。

**试题答案**

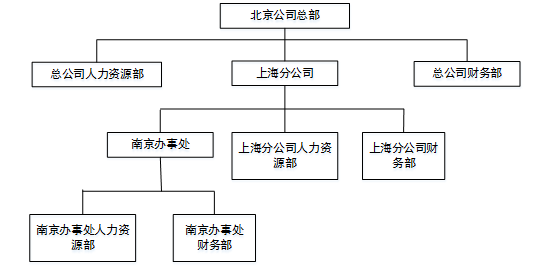
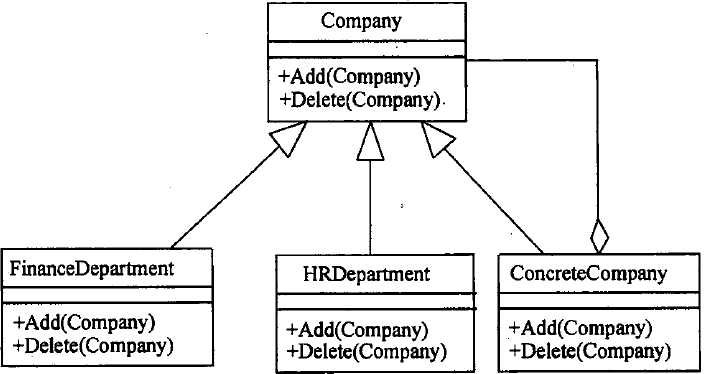
（5）

（1）this->name          （1分）  
（2）virtual void Add(Company\* c) = 0     （2分）  
（3）virtual void Delete(Company\* c) = 0         （2分）  
（4）Company\*          （2分）  
（5）children      （2分）  
（6）children      （2分）  
（7）root->Add(comp)        （2分）  
（8）comp->Add(comp1)      （2分）

# **试题30(2010年下半年试题6)**

 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】  
某公司的组织结构图如图6-1所示，现采用组合(Composition)设计模式来设计，得到如图6-2所示的类图。  
其中Company为抽象类，定义了在组织结构图上添加(Add)和删除(Delete)分公司/办事处或者部门的方法接口。类ConcreteCompany表示具体的分公司或者办事处，分公司或办事处下可以设置不同的部门。类HRDepartment和FinanceDepartment分别表示人力资源部和财务部。

  
**图6-1  组织结构图**  
****  
**图6-2  类图**

 【Java代码】

import jav（6）A．util.\*;  
（1）Company {  
  protectedString    name;  
  public Company(String name) { （2）= name; }  
  public abstract  void Add(Company  c); //增加子公司、办尊处或部门  
  public abstract  void Delete(Company  c);    //删除子公司、办事处或部门  
}  
   
class  ConcreteCompany  extends  Company {  
   private List<（3）> children = new ArrayList<（4）>();  
//存储子公司、办事处或部门  
   public  ConcreteCompany(String  name)  {  super(name); }  
   public  void  Add(Company c)   {  （5）.add(c);    }  
   public  void  Delete(Company  c)   {（6）.remove(c);    }  
}  
   
class HRDepartment extends Company {  
   public  HRDepartment(String  name)  { super(name);  }  
   //其他代码省略  
}  
   
class  FinanceDepartment   extends  Company   {  
   public  FinanceDepartment(String  name)   { super(name); }  
   //其他代码省略  
}  
   
public  class  Test  {  
  public  static void  main(String[]  args)   {  
    ConcreteCompany root =new ConcreteCompany("北京总公司");  
    root.Add(new HRDepartment("总公司人力资源部") );  
    root.Add(new FinanceDepartment("总公司财务部") );  
   
    ConcreteCompany comp =new ConcreteCompany("上海分公司");  
    comp.Add(new HRDepartment("上海分公司人力资源部") );  
    comp.Add(new FinanceDepartment("上海分公司财务部"));  
    （7）;  
   
    ConcreteCompany  compl  =  new ConcreteCompany("南京办事处");      
compl.Add(new HRDepartment("南京办事处人力资源部") );  
    compl.Add(new Fin（7）A．nceDepartment ("南京办事处财务部") );  
    （8）;    // 其他代码省略  
    }  
}

**试题分析**

本题考查了Java语言的应用能力和组合设计模式的应用。  
第（1）空在类名Company前，很明显应该要加关键字abstract class   ，因为题目描述的Company类是一个抽象类。第（2）空在构造函数中的赋值语句中，应该为this.name，就是给该类的一个属性name赋值，这里应该用this.name来引用这个属性。第（3）空与第（4）空的答案应该都为Company，这样要注意在java中与C++中的区别。第（5）和第（6）空的答案应该一样，一个用来增加节点，一个用来删除节点，都是使用的children对象。根据题目提供的组织结构图，我们可以知道，创建了上海分公司接的后，应该将其添加至root（北京总公司）下，因此第7空答案为root.Add(comp)，同样的道理，第（8）空的答案为comp.Add(comp1)。

**试题答案**

（6）

（1）abstract class     （2分）  
（2）this.name        （1分）  
（3）Company        （2分）  
（4）Company        （2分）  
（5）children      （2分）  
（6）children      （2分）  
（7）root.Add(comp)      （2分）  
（8）comp.Add(comp1)       （2分）

# **试题31(2010年上半年试题5)**

阅读下列说明和C++代码，将应填入 （n） 处的字句写在答题纸的对应栏内。  
【说明】  
　　某软件公司现欲开发一款飞机飞行模拟系统，该系统主要模拟不同种类飞机的飞行特征与起飞特征。需要模拟的飞机种类及其特征如表5-1所示。

**表5-1**

|  |  |  |
| --- | --- | --- |
| 飞机种类 | 起飞特征 | 飞行特征 |
| 直升机（Helicopter） | 垂直起飞（VerticalTakeOff） | 亚音速飞行 （SubSonicFly） |
| 客机（AirPlane） | 长距离起飞 （LongDistanceTakeOff） | 亚音速飞行 （SubSonicFly） |
| 歼击机（Fighter） | 长距离起飞 （LongDistanceTakeOff） | 超音速飞行 （SuperSonicFly） |
| 鹞式战斗机（Harrier） | 垂直起飞（VerticalTakeOff） | 超音速飞行 （SuperSonicFly） |

　为支持将来模拟更多种类的飞机，采用策略设计模式（Strategy）设计的类图如图5-1所示。

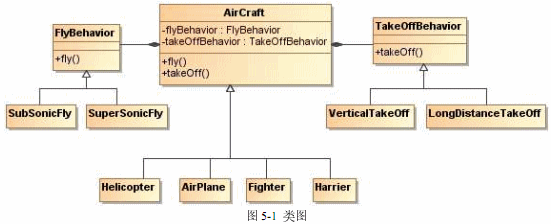


　　图5-1中，AirCraft为抽象类，描述了抽象的飞机，而类Helicopter、AirPlane、Fighter和Harrier分别描述具体的飞机种类，方法fly()和takeOff()分别表示不同飞机都具有飞行特征和起飞特征；类FlyBehavior与TakeOffBehavior为抽象类，分别用于表示抽象的飞行为与起飞行为；类SubSonicFly与SuperSonicFly分别描述亚音速飞行和超音速飞行的行为；类VerticalTakeOff与LongDistanceTakeOff分别描述垂直起飞与长距离起飞的行为。

【C++ 代码】

　#include<iostream>  
　　using namespace std;  
　　class FlyBehavior {  
　　public : virtual void fly() = 0;  
　};  
　　class SubSonicFly:public FlyBehavior{  
　　public: void fly(){ cout << "亚音速飞行！" << endl; }  
　};  
　　class SuperSonicFly:public FlyBehavior{  
　　public: void fly(){ cout << "超音速飞行！" << endl; }  
　};  
　　class TakeOffBehavior {  
　　public: virtual void takeOff() = 0;  
　};  
　　class VerticalTakeOff:public TakeOffBehavior{  
　　public: void takeOff(){ cout << "垂直起飞！" << endl; }  
　};  
　　class LongDistanceTakeOff:public TakeOffBehavior {  
　　public: void takeOff (){ cout << "长距离起飞！" << endl; }  
　};  
　　class AirCraft{  
　　protected:  
　　（1）   ;  
　　（2）   ;  
　　public:  
　　void fly(){（3）; }  
　　void takeOff() {（4）; };  
　};  
　　class Helicopter: public AirCraft {  
　　public:  
　　Helicopter (){  
　　flyBehavior = new（5）;  
　　takeOffBehavior = new（6）;  
　}  
　　　（7）{  
　　if(!flyBehavior) delete flyBehavior;  
　　if(!takeOffBehavior) delete takeOffBehavior;  
　　}  
　};  
　//  
　　其它代码省略

**试题分析**

    本题目考查了设计模式中的策略设计模式，实际上与2007年上半年考核内容相同。  
    从本题的叙述中可以看出，存在4种不同的飞机类型，但每种飞机类型的起飞特征和飞行特征并不完全相同，这就使得我们很难采用比较直接的方法来实现重用。例如，定义一个抽象的飞机类，实现飞机的起飞特征，然后4种飞机直接重用该特征。但是，我们可以观察到，尽管飞机的起飞特征和飞行特征有所不同，有一点可以肯定的是，每一种飞机都具备了飞行特征和起飞特征。因此，可以抽象出一个飞机类，其中含有飞行特征与起飞特征，但关于两个特征的实现要单独抽取出来，所以又形成了F1yBehavior类和TakeOffBehavior类分别表示抽象的飞行和起飞特征，而这两个类的子类则分别实现不同的起飞和飞行特征，最终转化为，在创建一个具体的飞机时，给其配上不同的起飞特征和飞行特征即可。  
    本题中的空（l）和空（2）应该填写成员变量，根据类图可以得知，此处应该表示的是飞行和起飞特征变量，在C++中可以采用指针一来表示。空（3）和空（4）处需要实现飞行与起飞特征，但AirCraft是抽象的类，所以把实现代理给指针一变量。Helicopter类需要指定由父类继承而来的成员变量的初始值，因为Helicopter的特征是垂直起飞和亚音速飞行，因此生成这两个特征的对象，分别赋值给flyBehavior和takeOffBehavior变量。

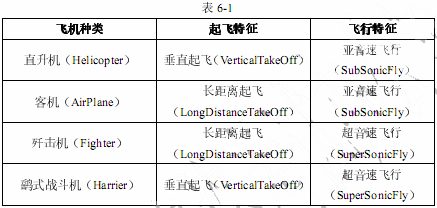
**试题答案**

（5）

    （1）FlyBehavior\*flyBehavior　　　　　　（2分）  
    （2）TakeOffBehavior\*takeOffBehavior    （2分）  
    （3）flyBehavior->fly()   　　　　　　　（2分）  
（4）takeOffBehavior->takeOff()   　　（2分）  
    （5）SubSonicFly()    　　　　　　　 （2分）  
（6）VerticalTakeOff()    　　　　　　　（2分）  
    （7）~Helicopter() 　　　　　　　　　   （3分）  
    注：空（1）与空（2）答案可互换

# **试题32(2010年上半年试题6)**

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】  
　　某软件公司现欲开发一款飞机飞行模拟系统，该系统主要模拟不同种类飞机的飞行特征与起飞特征。需要模拟的飞机种类及其特征如表6-1所示。



　　为支持将来模拟更多种类的飞机，采用策略设计模式（Strategy）设计的类图如图6-1所示。

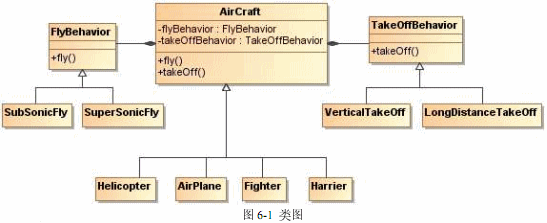


　　图6-1中，AirCraft为抽象类，描述了抽象的飞机，而类Helicopter、AirPlane、Fighter和Harrier分别描述具体的飞机种类，方法fly()和takeOff()分别表示不同飞机都具有飞行特征和起飞特征类FlyBehavior与TakeOffBehavior为抽象类，分别用于表示抽象的飞行为与起飞行为；类SubSonicFly与SuperSonicFly分别描述亚音速飞行和超音速飞行的行为；类VerticalTakeOff与LongDistanceTakeOff分别描述垂直起飞与长距离起飞的行为。

【Java 代码】

　interface FlyBehavior {  
　　public void fly();  
　};  
　class SubSonicFly implements FlyBehavior{  
　　public void fly(){ System.out.println("亚音速飞行！"); }  
　};  
　class SuperSonicFly implements FlyBehavior{  
　　public void fly(){ System.out.println("超音速飞行！" ); }  
　};  
　interface TakeOffBehavior {  
　　public void takeOff();  
　};  
　class VerticalTakeOff implements TakeOffBehavior {  
　　public void takeOff (){ System.out.println("垂直起飞！" ); }  
　};  
　class LongDistanceTakeOff implements TakeOffBehavior {  
　　public void takeOff(){ System.out.println("长距离起飞！"); }  
　};  
　　abstract class AirCraft {  
　　　　　　protected  （1）  ;  
　　　　　　protected  （2）  ;  
　　public void fly(){  （3）  ; }  
　　public void takeOff() {   （4）   ; };  
　};  
　class Helicopter（5）AirCraft{  
　　　　　　public Helicopter (){  
　　　　　　flyBehavior = new  （6）  ;  
　　　　　　takeOffBehavior = new  （7）  ;  
　　　}  
　　};  
　//其它代码省略

**试题分析**

    本题目考查了设计模式中的策略设计模式，实际上与2007年上半年Java题目的考核内容相同。     
    从本题的叙述中可以看出，存在四种不同的飞机类型，但每种飞机类型的起飞特和飞行特征并不完全相同，这就使得我们很难采用比较直接的方法来实现重用。例如，定义一个抽象的飞机类，实现飞机的起飞特征，然后四种飞机直接重用该特征。但是，我们可以观察到，尽管飞机的起飞特征和飞行特征有所不同，有一点可以肯定的是，一种飞机都具备了飞行特征和起飞特征。因此，可以抽象出一个飞机类，其中含有飞特征与起飞特征，但关于两个特征的实现要单独抽取出来，所以又形成了FIyBehavi类和TakeOffBehavior类，分别表示抽象的飞行和起飞特征，而这两个类的子类则分别现不同的起飞和飞行特征，最终转化为，在创建一个具体的飞机时，给其配上不同的飞特征和飞行特征即可。  
    本题中的空（1）和空（2）应该填写成员变量，根据类图可以得知，此处应该表示的是飞行和起飞特征变量。空（3）和空（4）处需要实现飞行与起飞特征，但AirCraft是抽象的类，所以把实现代理给指针变量。Helicopter类需要指定由父类继承而来的成员变量的初始值，因为Helicopter的特征是垂直起飞和亚音速飞行，因此生成这两个特征的对象，分别赋值给flyBehavior和takeOffBchavior变量。

**试题答案**

（6）

    （1）FlyBehavior flyBehavior   （2分）  
    （2）TakeOffBehavior takeOffBehavior    （2分）  
    （3）flyBehavior.fly() （2分）  
    （4）takeOffBehavior.takeOff()   （2分）  
    （5）extends  （3分）  
    （6）SubSonicFly()    （2分）  
    （7）VerticalTakeOff()       （2分）  
    注：空（1）与空（2）答案可互换