2018 年上半年软件设计师考试下午真题（参考答案）

● 阅读下列说明，回答问题 1 至问题 4，将解答填入答题纸的对应栏内。

【说明】

某医疗护理机构为老年人或有护理需求者提供专业护理，现欲开发一基于 web 的医疗管理系统，以改善医疗护理效率，该系统的主要要功能如下：

（1）通用信息查询，客户提交通用信息查询请求，查询通用信息表，返回查询结果。

（2）医生聘用。医生提出应聘 / 辞职申请，交由主管进行聘用/解聘审批，更新医生表，并给医生反馈聘用/解聘结果：删除解聘医生的出诊发排

（3）预约处理。医生安排出诊时间，存入医生出诊时间表，根据客户提交的预约查询请求，查询在职医生及其出诊时间等预约所需数据并返回：创建预约，提交预约请求，在预约表中新增预约记录，更新所约医生出诊时间并给医生发送预约通知；给客户反馈预约结果。

（4）药品管理。医生提交处方，根据药品名称从药品数据中查询相关药品库存信息，开出药品，更新对应药品的库存以及预约表中的治疗信息；给医生发送“药品已开出”反馈。

（5）报表创建。根据主管提交的报表查询请求（报表类型和时间段），从预约数据、通用信息、药品库存数据、医生以及医生出诊时间中进行查询，生成报表返回给主管。

现采用结构化方法对医疗管理系统进行分析与设计，获得如图 1-1 所示的上下文数据流图和图 1-2 所示的 0 层数据流图。

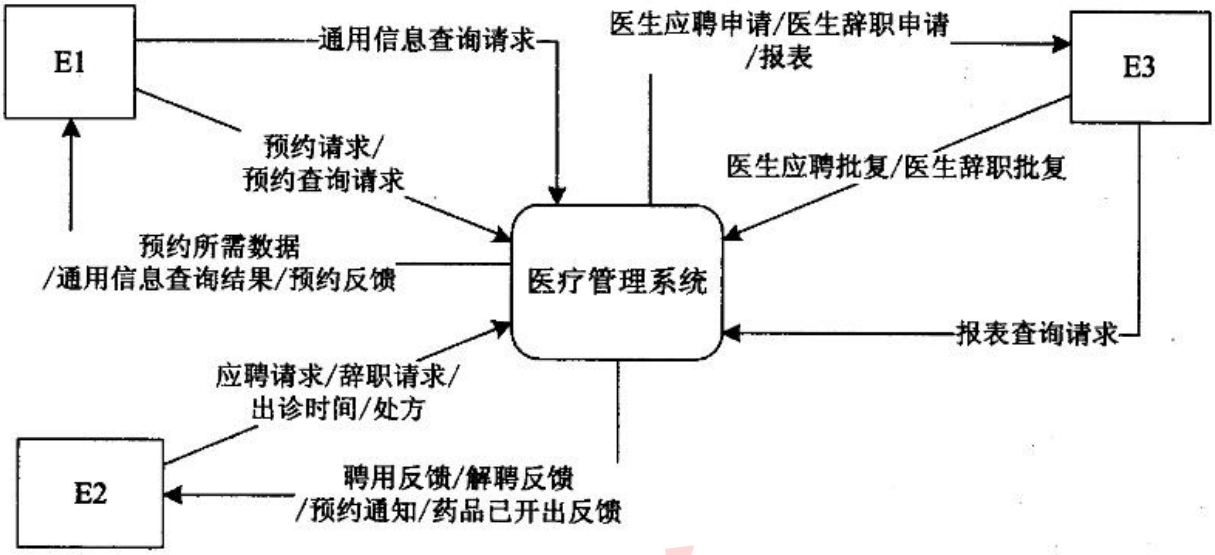
E1
通用信息查询请求
医生应聘申请/医生辞职申请
/报表
E3
预约请求/
预约查询请求
医生应聘批复/医生辞职批复
预约所需数据
/通用信息查询结果/预约反馈
医疗管理系统
报表查询请求
应聘请求/辞职请求/
出诊时间/处方
E2
聘用反馈/解聘反馈
/预约通知/药品已开出反馈

图 1-1 上下文数据流图

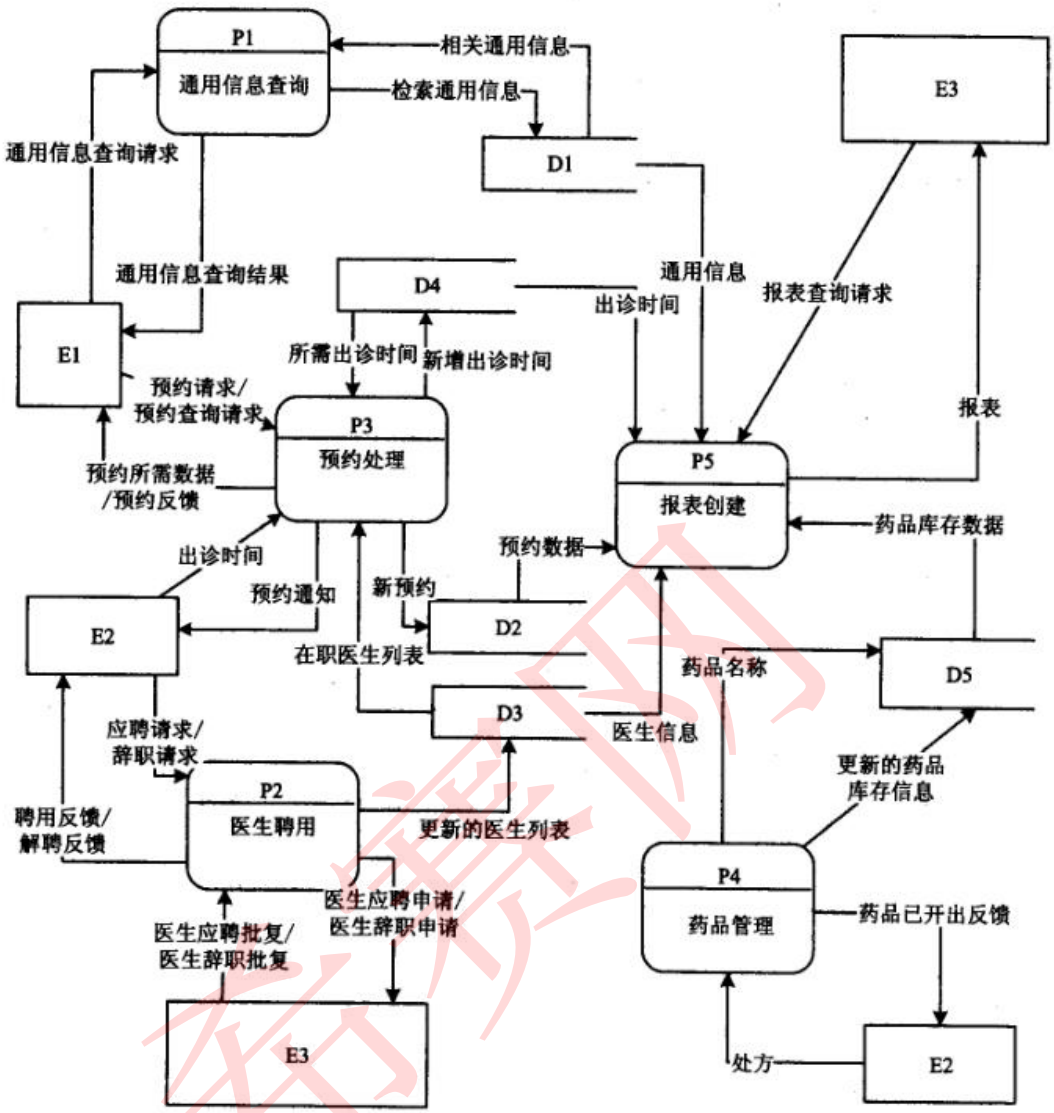

图 1-2 0 层数据流图

【问题 1】（3 分）

使用说明中的词语，给出图 1－1 中的实体 E1～E3 的名称。

【问题 2】（5 分）

使用说明中的词语，给出图 1－2 中的数据存储 D1～D5 的名称。

【问题 3）（4 分）

使用说明和图中术语，补充图 1－2 中缺失的数据流及其起点和终点。

【问题 4】（3 分）

使用说明中的词语，说明“预约处理”可以分解为哪些子加工，并说明建模图 1－1 和图 1－2 是如何保持数据流图平衡。

● 阅读下列说明，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

【说明】

某海外代购公司为扩展公司业务，需要开发一个信息化管理系统。请根据公司现有业务及需求完成该系统的数据库设计。

【需求描述】

（1）记录公司员工信息。员工信息包括工号、身份证号、姓名、性别和一个手机号，工号唯一标识每位员工，员工分为代购员和配送员。

（2）记录采购的商品信息。商品信息包括商品名称、所在超市名称、采购价格、销售价格和商品介绍，系统内部用商品条码唯一标识每种商品。一种商品只在一家超市代购。

（3）记录顾客信息。顾客信息包括顾客真实姓名、身份证号（清关缴税用）、一个手机号和一个收货地址，系统自动生成唯一的顾客编号。

（4）记录托运公司信息。托运公司信息包括托运公司名称、电话和地址，系统自动生成唯一的托运公司编号。

（5）顾客登录系统之后，可以下订单购买商品。订单支付成功后，系统记录唯一的支付凭证编号，顾客需要在订单里指定运送方式：空运或海运。

（6）代购员根据顾客的订单在超市采购对应商品，一份订单所含的多个商品可能由多名代购员从不同超市采购。

（7）采购完的商品交由配送员根据顾客订单组合装箱，然后交给托运公司运送。托运公司按顾客订单核对商品名称和数量，然后按顾客的地址进行运送。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图（不完整）如图 2—1 所示。

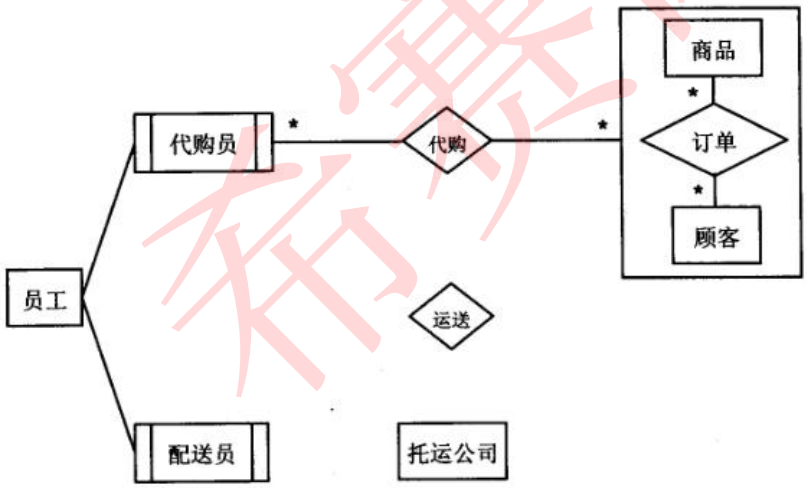

图 2-1 实体联系图

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式（不完整）：

员工（工号，身份证号，姓名，性别，手机号）

商品（条码，商品名称，所在超市名称，采购价格，销售价格，商品介绍）

顾客（编号，姓名，身份证号，手机号，收货地址）

托运公司（托运公司编号，托运公司名称，电话，地址）

订单（订单 ID，（a），商品数量，运送方式，支付凭证编号）

代购（代购 ID，代购员工号，（b））

运送（运送 ID，配送员工号，托运公司编号，订单 ID，发运时间）

【问题 1】（3 分）

根据问题描述，补充图 2－1 的实体联系图。

【问题 2】（6 分）

补充逻辑结构设计结果中的（a）、（b）两处空缺。

【问题 3】（6 分）

为方便顾客，允许顾客在系统中保存多组收货地址。请根据此需求，增加“顾客地址”弱实体，对图 2－1 进行补充，并修改“运送”关系模式。

● 阅读下列说明，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

【说明】

某 ETC（ Electronic Toll Collection，不停车收费）系统在高速公路沿线的特定位置上设置一个横跨道路上空的龙门架（ Toll gantry),龙门架下包括 6 条车道（ Traffic lanes),每条车道上安装有雷达传感器（ Radar sensor)、无线传输器（ Radio transceiver）和数码相机（ Digital Camera）等用于不停车收费的设备，以完成正常行驶速度下的收费工作。该系统的基本工作过程如下：

（1）每辆汽车上安装有车载器，驾驶员（ Driver）将一张具有唯一识别码的磁卡插入车载器中。磁卡中还包含有驾驶员账户的当前信用记录。

（2）当汽车通过某条车道时，不停车收费设备识别车载器内的特有编码，判断车型，将收集到的相关信息发送到该路段所属的区域系统（ Regional center）中，计算通行费用创建收费交易（ Transaction），从驾驶员的专用账户中扣除通行费用。如果驾驶员账户透支，则记录透支账户交易信息。区域系统再将交易后的账户信息发送到维护驾驶员账户信息的中心系统（ Central system）

（3）车载器中的磁卡可以使用邮局的付款机进行充值。充值信息会传送至中心系统，以更新驾驶员账户的余额。

（4）当没有安装车载器或者车载器发生故障的车辆通过车道时，车道上的数码相机将对车辆进行拍照，并将车辆照片及拍摄时间发送到区域系统，记录失败的交易信息；并将该交易信息发送到中心系统。

（5）区域系统会获取不停车收费设备所记录的交通事件（ Traffic events）；交通广播电台（ Traffic advice center）根据这些交通事件进行路况分析并播报路况。

现采用面向对象方法对上述系统进行分析与设计，得到如表 3－1 所示的用例列表以及如图 3－1 所示的用例图和图 3－2 所示的分析类图。

表 3-1　用例列表

用例名称	说明
Create transaction	记录收费交易
Charge card	磁卡充值
Underpaid transaction	记录透支账户交易信息
Record Illegal use	记录失败交易信息
Record traffic event	记录交通事件

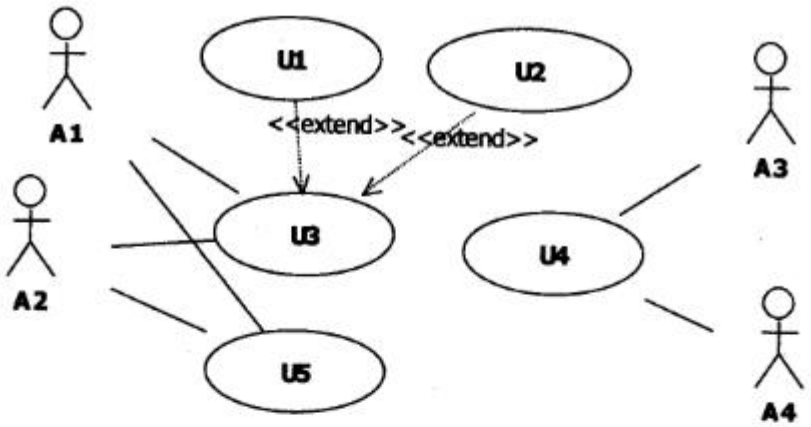

图 3-1　用例图

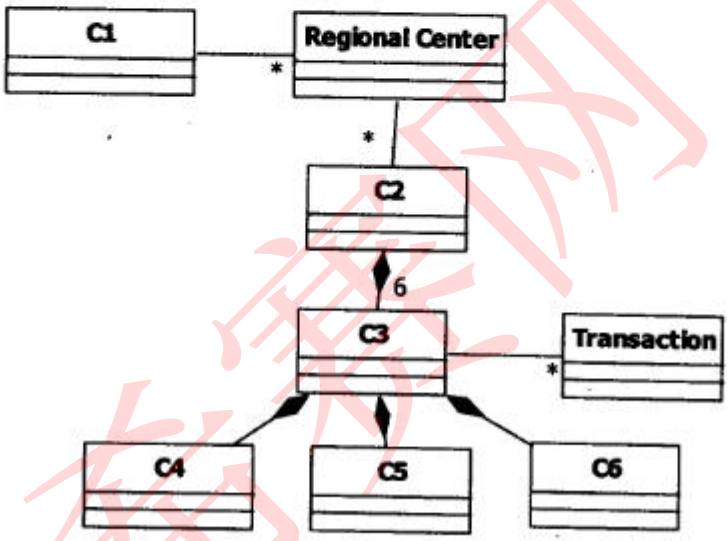

图 3-2　分析类图

【问题 1】（4 分）

根据说明中的描述，给出图 3－1 中 A1～A4 所对应的参与者名称。

【问题 2）（5 分）

根据说明中的描述及表 3－1，给出图 3－1 中 U1－U5 所对应的用例名称。

【问题 3】（6 分）

根据说明中的描述，给出图 3－2 中 C1~C6 所对应的类名。

● 阅读下列说明和 C 代码，回答问题 1 和问题 2，将解答填入答题纸的对应栏内。

【说明】

某公司购买长钢条，将其切割后进行出售。切割钢条的成本可以忽略不计，钢条的长度为整英寸。已知价格表 P，其中中 P_i（i＝1，2，...，m）表示长度为 i 英寸的钢条的价格。现要求解使销售收益最大的切割方案。

求解此切割方案的算法基本思想如下：

假设长钢条的长度为 n 英寸，最佳切割方案的最左边切割段长度为 i 英寸，则继续求解剩余长度为 n－i 英寸钢条的最佳切割方案。考虑所有可能的 i，得到的最大收益 r_n 对应的切割方案即为最佳切割方案。r_n 的递归定义如下：

$r_n=\max_{1\leqslant i\leqslant n}(p_i+r_{n-i})$

对此递归式，给出自顶向下和自底向上两种实现方式

【C 代码】

```
/*常量和变量说明
n：长钢条的长度
P[]：价格数组
*/
#define LEN 100

int Top_Down_ Cut_Rod(int P[], int n) {/*自顶向下*/
        int r=0;
        int i;
        if(n==0){
               retum 0;
        }
        for(i=1; (1) ;i++){
                    int tmp=p[i]+Top_Down_ Cut_Rod(p,n-i);
                    r=(r>=tmp)?r: tmp;
        }
        return r;
}


int Bottom_Up_Cut_Rod(int p[],int n){          /*自底向上*/
                int r[LEN]={0};
                int temp=0;
                int i,j;
                for(j=1;j<=n;j++){
                                    temp=0;
                                    for(i=1;(2);i++){
                                                        temp=(3);
                                      }
                                (4)
                }
                return r[n];
}
```

【问题 1】（8 分）

根据说明，填充 C 代码中的空（1）～（4）。

【问题 2】（7 分）

根据说明和 C 代码，算法采用的设计策略为（5）。

求解 r_n 时，自顶向下方法的时间复杂度为（6）；自底向上方法的时间复杂度为（7）（用 O 表示）。

● 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】

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 5-1 所示为其类图。

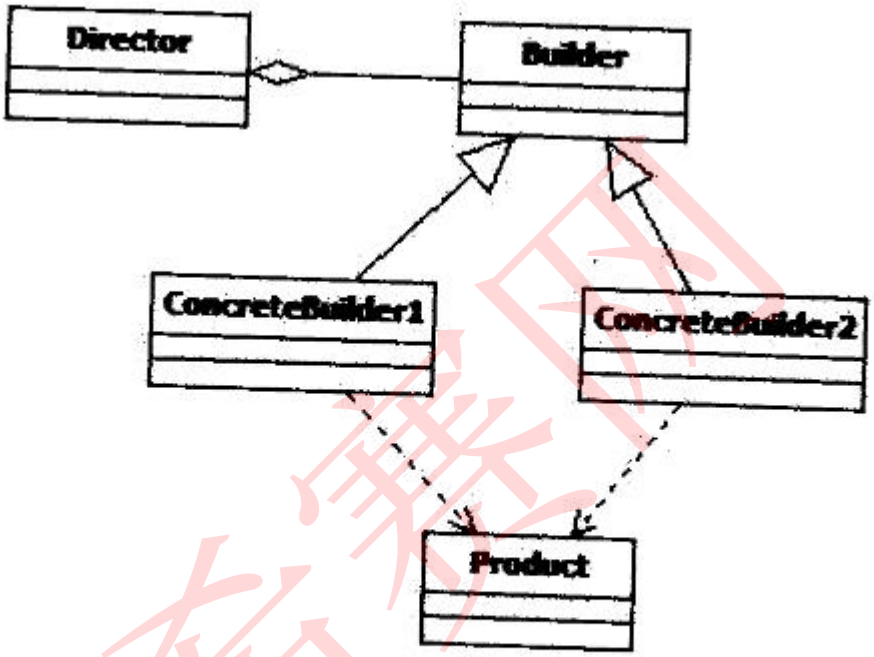

图 5-1 生成器模式类图

【C++代码】

```
#include <iostream>
#include <string>
using namespace std;

class Product {
private:
       string partA, partB;
public:
       Product__(5)__{    }
      void setPartA(const string&s){ PartA=s;}
       void setPartB(const string&s){ PartB=s;}
      //其余代码省略
```

```
};
class Builder{
public:
       (1);
       virtual void buildPartB__(6)__=0;
       (2);
};
class ConcreteBuilder1: public Builder{
private:
         Product*     product;
public:
       ConcreteBuilder1__(7)__{product=new Product__(8)__; }
     void buildPartA__(9)__{  (3) ("Component A");}
     void buildPartB__(10)__{  (4) ("Component B");}
     Product*getResult__(11)__{ return product;}
     //其余代码省略
};
class ConcreteBuilder2: public Builder{

                /*代码省略*/
};
class Director {
private:
         Builder* builder;
public:
       Director(Builder*pBuilder){builder*pBuilder;}
       void construct__(12)__{
                  (5)
                  //其余代码省略
       }
     //其余代码省略

};
int main__(13)__{
           Director* director1=new Director(new ConcreteBuilder1__(14)__);
           director1->construct__(15)__;
           delete director 1;
         return 0;
}
```

● 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。

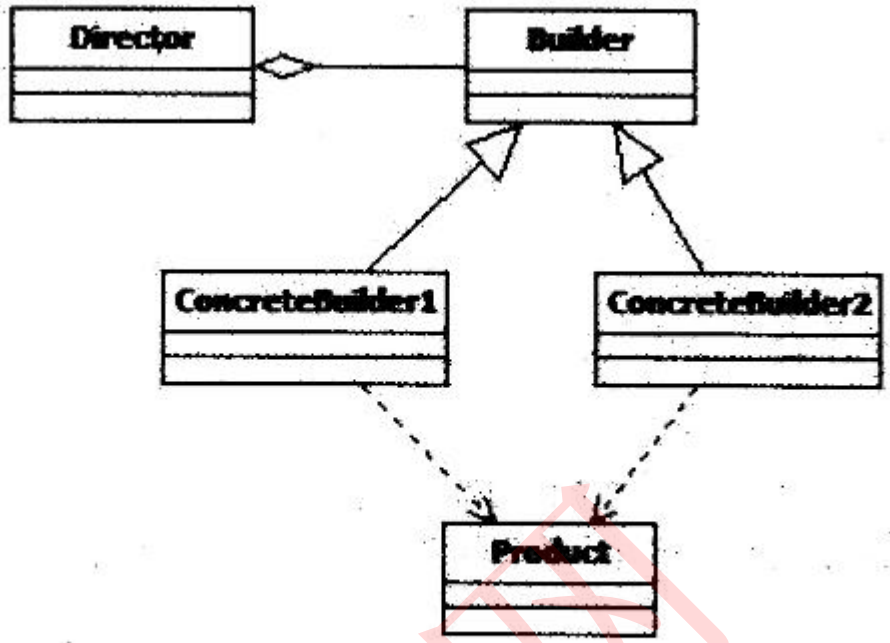

图 6-1 生成器模式类图

【Java 代码】

```
import java.util.*;

class Product {
          private String part A;
          private String part B;
          public Product__(6)__{}
          public void setPartA(String s){ partA=s;}
          public void setPartB(String s){ partB=s;}
}
interface Builder{
         public (1);
         public void buildPartB__(7)__;
         public (2);
}
class ConcreteBuilder1: implements Builder{
          private Product product;
          public ConcreteBuilder1__(8)__{product=new Product__(9)__; }
          public void buildPartA__(10)__{  (3) ("Component A");}
          public void buildPartB__(11)__{  (4) ("Component B");}
          public Product getResult__(12)__{ 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__(13)__{
                (5)
                //代码省略
        }
}

class Test{
        public static void main(String[]args){
                                        Director   director1=new   Director(new
ConcreteBuilder1__(14)__);
                director1.construct__(15)__;
        }
}
```