2021 上

（1）protected
（2）abstract boolean addMenuElement(MemuComponent element)
（3）abstract List<MenuComponent> getElement()
（4）ArrayList<MenuComponent> elementList
（5）mainMenu.addMenuElement(subMenu)

本题是典型的组合模式应用。
首先根据类图中name标注的为#（+表示public，-表示private，#表示protected），第（1）空对name的修饰应该是protected。
然后根据抽象类和实现类的对应关系，可以补充第（2）（3）空，这两处缺失的方法根据下文代码进行补充，注意方法名必须用abstract修饰，并且注意抽象方法的写法没有具体方法体。其中第（2）空填写abstract boolean addMenuElement(MemuComponent element)，第（3）空填写abstract List<MenuComponent> getElement()。
第（4）空缺失了一个属性，此时根据下文的同名构造函数会发现，此处传参给了this.name以及this.elementList，name可以根据父类继承使用，而elementList需要定义，因此此处缺失的参数是elementList，类型根据后面的赋值类型进行定义，即第（4）空填写ArrayList<MenuComponent> elementList。
第（5）空是对组合模式的应用拼装，根据下文可知打印需要调用mainMenu对象，而此时该对象是独立的，需要与其他菜单进行拼装，下文中subMenu拼装了element，此处需要将subMenu拼装到mainMenu，即第（5）空填写mainMenu.addMenuElement(subMenu)。

2021 下

(1)public abstract void draw( )
(2)Piece
(3)Piece
(4)piece.draw( )
(5)piece.draw( )

对于第一空，可知该空需要填写的是 Piece类里面的方法，对于其方法在图中都无法找出，可以根据其实现类（BlackPiece和WhitePiece类）来看，对应得是方法public void draw( )，又由于其在抽象类Piece里面，所以是抽象方法，需要加上关键词abstract，则为public abstract void draw( )
对于第二空，可知该空填写的是动态数组Arraylist的泛型，<>里面填写得应该是对应的m_arrayPiece的类型，用类进行修饰，可知其属于Piece类，填写的应该是Piece
对于第三空，可知该空填写的是对象创建的声明对象过程，格式应该为类名 对象名称=null，可知该对象piece对应的类是Piece（类名字母大写）
对于第四空和第五空，根据注释来看，是放黑子和白子的过程，已知实例化该对象piece，具体的放黑子和白子过程，都需要调用draw（）方法来指向，故 第4空和第5空填写的应该都是piece.draw( )

2020

(1) void buy(double money, WebService service)
(2) WebServiceMediator
(3) abstract void buyService(double money)
(4) mediator.buy(money, this)
(5) mediator.buy(money, this)

（1）空是属于接口WebServiceMeditor内的方法，我们可以通过下文的实现类中找到ConcreteServiceMeditor可知缺少了一个buy（）方法
故第一空填写void buy(double money, WebService service)；（2）空类WebService中属性的参数类型,Colleague与Mediator之间的关联关系由属性meditor实现，所以第2空应该填写WebServiceMediator；（3）空类WebService中的抽象方法，根据其具体子类可以看到缺少的是buyService方法，书写成 abstract void buyService(double money)；（4）空和（5）空具体同事类Amazon、Ebay与中介者的通信，调用中介者之间的支付接口，所以空（4）和（5）都填写mediator.buy(money, this)。

2019 下

答案

（1）void update();

（2）Observer;

（3）obs.update();

（4）Subject;

（5）Attach(this);

试题分析

本题是对观察者模式的考查，观察者模式的意图是：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

本题根据Observer接口的实现类DocExplorer，实现类包括同名构造函数和update()方法，所以接口Observer缺失的是update()方法，并且没有方法体，（1）空填写void update()。

第2空是myObs表单类型的缺失，根据代码上下文，在构造函数中，根据this.myObs=new ArrayList<Observer>()，可以知道myObs是Observer表单，第（2）空填写Observer。

第3空是Notify()方法体的缺失，根据代码上下文，Notify传入了一个参数Observer obs，又根据代码上下文可知Observer只有一个update()方法，此时方法体调用的应该是update()方法，调用方法的对应是传入的obs，第（3）空填写obs.update()。

第4空、第5空缺失的是DocExplorer类的同名构造函数传入的参数类型以及构造方法体，这里结合根据观察者模式填空，对于实际观察者类，需要与被观察者联系起来，所以这里是与被观察者Subject联系，也就是调用Subject中的Attach()添加观察者列表。因此第（4）空需要填写参数类型Subject，形参名sub已经给出了提示；sub调用增加观察者方法，将当前观察者添加到对应列表，即第（5）空填写Attach(this)。

2019 上

答案

(1) void stop()

(2) BrakeBehavior

(3) wheel.stop()

(4) wheel=behavior

(5) brake()

试题分析

策略模式是定义一系列算法，把他们一个个封装起来，并且使它们之间可相互替换，从而让算法可以独立于使用它的用户而变化。

(1) 第一空接口BrakeBehavior有内容缺失，结合其实现类LongWheelBrake代码如下：

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

第一空需要补充stop()方法，即（1）void stop()

(2) (3) 第二、三空是抽象类Car缺少属性wheel的类型和brake()方法的方法体。

```
abstract class Car {
    protected        (2)     wheel;
    public  void brake() {       (3)     ; }
    /* 其余代码省略 */
};
```

根据图示策略模式，Car与BrakeBehavior 是整体与部分的关系，因此Car的属性有这一个部分，即（2）BrakeBehavior ，这里在类Car中，命名了一个与之联系的部分BrakeBehavior 类型wheel。对于BrakeBehavior 类，所包含的方法是stop，因此第（3）空填写的方法应该是wheel.stop()。这样就将Car与BrakeBehavior 联系起来了。

(4) 第四空是实现子类ShortWheelCar缺失ShortWheelCar(BrakeBehavior behavior)此带参构造方法的方法体：

```
class ShortWheelCar extends Car {
     public ShortWheelCar(BrakeBehavior behavior) {
              (4)    ;
     }
     /* 其余代码省略 */
};
```

构造方法是对类的构造，带参构造函数一般是对其属性进行参数赋值，第四空将实现子类与其父类联系起来，子类继承父类属性wheel，此处应该是对参数wheel赋值，即（4）wheel=behavior

（5）第五空是实际调用测试过程，缺失方法名：

```
class StrategyTest{
    public static void main(String[] args) {
        BrakeBehavior brake = new ShortWheelBrake();
        ShortWheelCar car1 = new ShortWheelCar(brake);
        car1.        (5)    ;
    }
}
```

由代码可知，car1是ShortWheelCar(brake)实例化的对象，类ShortWheelCar本身没有方法，只有默认继承父类的一个方法brake()，因此此处调用的是brake()，即（5）brake()。

2018 下

答案

（1）abstract double travel(int miles,FrequentFlyer context)

（2）context.setState(new CSilver())

（3）context.setState(new CGold ())

（4）context.setState(new CSilver())

（5）context.setState(new CBasic())

试题分析

由代码可知，（1）空缺少一个抽象方法，根据下面的子类可以发现，子类都有double travel(int miles, FrequentFlyer context)方法，是从该抽象类中继承而来，因此（1）空应该补充这个方法，并加上abstract修饰。

（2）（3）（4）（5）可以从状态图中根据相关状态推断出来。

首先，（2）（3）属于普卡会员CBasic，从状态图和代码可以看到，当里程>=25000且<5000时，会员等级应该从普卡会员CBasic升级到银卡会员CSilver，根据后面已有的代码，可以推断表示升级到银卡会员CSilver的表示方式为context.setState(new CSilver())；同理对于（3）空，在普卡会员CBasic状态，里程>=50000时，应该升级为金卡会员CGold，此时升级金卡CGold的表示方式为context.setState(new CGold ())，以此类推，（4）（5）分别对应金卡会员CGold状态下，不同条件，降低的不同等级。因此（4）为降级为银卡会员CSilver，（5）为降级为普卡会员CBasic，对应的表示方式分别为context.setState(new CSilver())和context.setState(new CBasic())。

2018 上

答案

（1）void buildPartA()

（2）Product getResult()

（3）product.setPartA

（4）product.setPartB

（5） builder.buildPartA();

或builder.buildPartB()

试题分析

本题考查的是面向对象程序设计，是JAVA语言与设计模式的结合考查。本题涉及的设计模式是构建器模式，将复杂类的构造过程推迟到子类实现。

对于第一空、第二空，根据实现接口的类，补充其接口缺失的方法，因此，空（1）和空（2）分别填写void buildPartA()和Product getResult()，二者可以互换；

对于第三空、第四空，是根据product类方法进行的补充，与具体产品的实现保持一致，因此，分别填写， product.setPartA， product.setPartB；

对于第五空，由于在填空后面跟随的是代码省略，因此题目并不严谨，缺失的语句可以有builder.buildPartA(); builder.buildPartB()。

2017 上

答案

（1）abstract void buildParts();

（2）this.pizzaBuilder=pizzaBuilder

（3）pizzaBuilder.buildParts()

（4）waiter.setPizzaBuilder(hawaiian_pizzabuilder)

（5）waiter.construct()

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。

2017 下

> 答案
>
> 1.abstract void doPaint(Matrix m)
>
> 2.imp.doPaint(m)
>
> 3.new GIFImage()
>
> 4.new LinuxImp()
>
> 5.image.setImp(imageImp)
>
> 试题分析
>
> 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)

2016 上

答案

（1）Address address;

（2）address.street();

（3）address.zip();

（4）address.city();

（5）DutchAddress addrAdapter=new DutchAddressAdapter(addr);

试题分析

本题考查的是面向对象程序设计，结合设计模式。本题涉及的设计模式是适配器。

对于代码填空，可以参照类图和代码上下文补充。

首先理清类与类之间的继承关系，再根据上下文填写。

对于第（1）空，DutchAddressAdapter继承了DutchAddress方法，根据下面的同名构造函数可知，该类定义了一个名叫address的参数，而根据代码上下文可以，address的类型为Address。本空应该填写Address address；

第（2）（3）（4）空是接口转换的具体实现，而在DutchAddressAdapter涉及的方法，可以从类图中找到，分别是straat()，postcode()，plaats()，适配器的目的是接口转换，即用这些方法分别展现原有Address中的street()、zip()、city()方法，因此这3个空分别填写address.street()、address.zip()、address.city()。

对于第（5）空，根据上下文最终调用testDutch方法的对象是addrAdapter，而此处是将原有的Address对象addr转换为接口对象，因此此处填写

DutchAddress addrAdapter=new DutchAddressAdapter(addr)。

2016 下

答案

（1） ticket.printInvoice()

（2） super.printInvoice()

（3） super.printInvoice()

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

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

试题分析

本题考查的是面向对象程序设计和设计模式。本题涉及的设计模式是装饰模式。

装饰模式（Decorator）：动态地给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活。

对于程序填空可以参照代码上下文、题干说明和设计模式综合考虑。

对于第（1）空，是对printInvoice方法的具体调用，在Decorator是装饰类，继承了Invoice发票类。此处需要填写的是printInvoice方法的方法体，根据Decorator类的上下文，已定义ticket对象，所以此处调用printInvoice方法的是ticket，第（1）空填写ticket.printInvoice()。

对于第（2）（3）空，根据类图可知，分别是HeadDecorator抬头类、FootDecorator脚注类调用printInvoice方法的方法体，由于在这两个类中并没有定义属性，只有借助其超类的构造函数，所以这两个地方调用printInvoice方法的是它们的超类，即（2）（3）填写的是super.printInvoice()。

对于第（4）（5）空，考查的是对装饰模式的调用，都是main函数中实例化的过程，根据输出结果可以看到，第（4）空实例化ticket对象，可以输出抬头、内容、脚注3个部分，因此需要调用三者的printInvoice()方法，前面已经实例化了一个Invoice对象t，可以利用t给子类实例化，因此第（4）空填写new HeadDecorator(new FootDecorator(t))；而第（5）空没有输出具体内容，只有抬头和脚注部分，可以看到这里的Invoice对象应该是空，所以第（5）空填写new HeadDecorator(new FootDecorator(null))。

2015 下

答案

（1）double acceptCash(double money)

（2）cs = new CashNormal()

（3）cs = new CashDiscount(0.8)

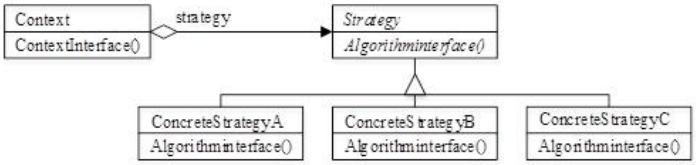
（4）cs = new CashReturn(300,100)

（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句的形式出现，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 是父类接口，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。从 3 个子类 CashNormal、CashDiscout 和 CashReturn的代码可以看出，公共操作方法为 double acceptCash(double money) 。由于不需要父类 CashSuper 提供任何的促销实现方式，在接口CashSuper中double acceptCash(double money) 方式是没有具体实现内容的。应填入空（1）处的语句是 "double acceptCash(double money)"。

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)"。

2015 上

答案

（1）void visit(Book p_book)

（2）void visit(Article p_article)

（3）void accept(LibraryVisitor visitor)

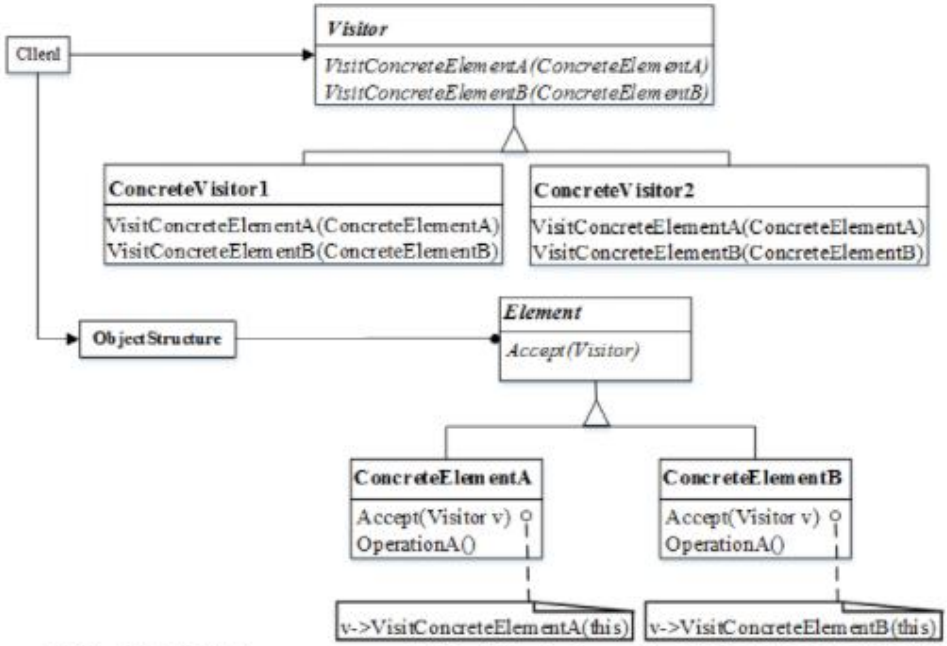
（4）visitor.visit(this)

（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。由前文分析已知， 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）。

2014 上

答案

（1）Subject

（2）observer.update(temperature,humidity,cleanness)

（3）notifyObservers()

（4）measurementsChanged()

（5）Observer

（6）envData.registerObserver(this)

试题分析

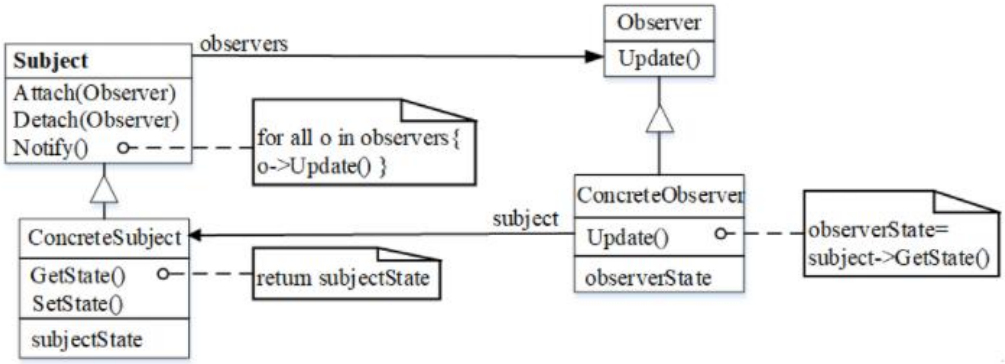
EnvironmentData是环境数据，也就是我们要监测的对象，即主题（Subject），因此（1）处为Subject。

（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。



2014 下

答案

（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.

（1）是Command接口的实现，应该填写interface Command；

（2）和（3）定义了开灯、关灯action，因此，分别填写（2）light->on()（3）light->off()；

（4）（5）分别设置“开灯”命令对象、“关灯”命令对象，因此分别填写（4）onCommands[slot]（5）offCommands[slot]；

（6）（7）分别完成对开灯、关灯命令对象的execute方法的调用，因此分别填写（6）onCommands[slot].execute()

（7）offCommands[slot].execute()。