2025 年春软件构造期末试题（回忆版）答案

编写：大半凉；审核、校对：大半凉

格式表明文本由 ChatGPT 生成；“格式”表明文本来自授课 PPT。

选择

1. C
2. B。对于 A 选项，Inheritance 明确了私有成员无法被继承。
3. D。`IImpl.counter` 为静态字段，需要通过名称限定才能访问。
4. C。其他三项分别为语句覆盖、判定覆盖、判定条件覆盖。
5. D。对于 D 选项，除了顶层容器，其他容器必须放在容器内才能使用。
6. B
7. A
8. B
9. C
10. B

判断

1. √
2. √
3. ×。逻辑覆盖由弱到强的指标分别为：语句、判定、条件、判定条件、条件组合、路径覆盖。
4. ×。迭代器模式的核心价值是统一访问方式和解耦遍历算法以及聚合对象的内部结构。
5. ×。工厂模式用来创建对象，显然是创建型模式。

填空

1. 面向对象特性：**封装**、继承、多态。
2. 通配符 **extends** 代表子类；反之，super 代表父类。
3. 模型与视图之间通常使用**观察者模式**进行交互。
4. 设计模式可以划分为：**创建型模式**、结构型模式和行为型模式。
5. TCP/IP 协议基于**字节**流传输。

（简答 1）多态

“多态是同一个行为具有多种表现形态的能力。”

多态可以通过重写（或重载，但二者并不相同）、抽象类（抽象方法）或接口实现（后二也是多继承的方法）。

多态最重要的优势在于实现了代码复用性（对外部提供了统一的代码模板），“减少耦合、增强可替换和可扩展性以及灵活性”。

（简答 2）抽象类与接口的异同

本题可答的地方有很多，考场上多数都是自由发挥。

其一，对于属性，抽象类可以提供统一的成员字段供子类继承，而接口仅能（并默认）声明公开的、静态的最终字段（也是相同点）。

其二，对于方法，抽象类可以声明非抽象方法的主体，而接口仅能声明（公开、静态）默认方法；它们都能声明抽象方法。

其三，对于继承，抽象类只能被单继承，而接口可以在继承类或抽象类外再继承多个接口。

抽象类最大的作用是确保可更改成员（属性）的一致性；而接口则强调功能可用性（方法）的一致。

（简答 3）ArrayList 与 LinkedList 比较

“ArrayList 类是一个可以动态修改的数组；LinkedList 是一种链表。”

“与 ArrayList 相比，LinkedList 的增加和删除的操作效率更高，而查找和修改的操作效率更低。”

因此，若想“频繁访问列表中的某个元素、只需要在列表末尾进行添加和删除元素操作”时，使用 ArayList；若想“需要频繁的在列表指定位置进行添加和删除元素操作”时，使用 LinkedList。

（简答 4）死锁

(1) 程序死锁。

(2) “在线程 A 持有锁 A 并想获得锁 B 的同时，线程 B 持有锁 B 并尝试获得锁 A，那么这两个线程将永远地等待下去。”

(3) “线程获取锁的顺序要一致。即严格按照先获取锁 A，再获取锁 B 的顺序，或者先获取锁 B，再获取锁 A 的顺序。”

（简答 5）super 与 this 的异同

不同点：

super 作为标识父类部分的关键字，可访问父类的公开或受保护的成员；

this 作为标识当前类实例的关键字，可访问本类的所有成员（静态字段和方法不允许实例调用）。当参数与字段同名时，可用该关键字进行区分，以防同名隐蔽的发生。

相同点：

只能出现在类的内部，用来访问成员；（不能在静态方法或类外使用）

都用来区分作用域……；

在语法上都是指针型引用……（super 其实也是通过当前对象的引用去访问“父类视角”的成员。所以两者在 JVM 层面都属于“访问当前实例的引用”的操作，只是访问入口不同。）

（简答 6）生产者 - 消费者模式

(1) “保持缓冲区数据的一致性。”

(2) 关键字 this 指代该实例。

(3) “循环会在线程睡眠前后都检查 wait 的条件，并在条件实际上并未改变的情况下处理唤醒通知”。若改成 if：

a) 不能防止伪唤醒。wait()可能无原因地返回，if 只检查一次，线程会继续执行并可能在条件仍为真的情况下进入临界区。

b) 竞态下会破坏容量约束。多个生产者被唤醒时，if 不会重新检查，可能同时 add，导致实际元素数超过逻辑上的 MAX（ArrayList 会自动扩容，程序不会抛异常但逻辑错乱）。

c) 可能引起消费者和生产者之间的错误同步、数据不一致或长期等待。

(4) 实现并发与解耦。(“生产者和消费者各司其职，生产者和消费者都只需要关心缓冲区，不需要互相关注，通过异步的方式支持高并发，将一个耗时的流程拆成生产和消费两个阶段。解耦：生产者和消费者通过缓冲区通讯进行解耦。”)

经典并发同步模式：生产者-消费者设计模式 - 知乎

（综合 1）多态

(1) 关键字 super 为了调用父抽象类的构造方法。

(2) 语句不正确。

尽管抽象类具有构造方法，但由于抽象类是不具体的，抽象类通常包含抽象方法，这些方法没有具体的实现，只是定义了方法的签名。由于抽象类不包含所有必要的实现细节，因此无法创建一个具体的对象。

其次，内存分配问题。当我们实例化一个对象时，关键字 new 会向 JVM 申请内存，并将类的成员保存到内存中。然而，抽象类没有具体的成员，因此无法准确分配内存。

(3) 不违反标准。尽管多数面向对象的语言禁止了继承多个类，但允许继承单个类的同时继承多个接口。

（综合 2）模板模式 new

(1) 模板模式“定义一个操作中的算法骨架，将一些步骤延迟到子类中使得子类可以不改变一个算法的结构即可重新定义该算法的某些特定步骤。”

(2) 一个抽象模板的顶层逻辑可继承至多个具体的逻辑，这些逻辑给出了抽象方法的不同实现，这也就实现了多态。

(3) UML 图略。

(4) 好处是把不变行为搬到超类，去除子类中的重复代码，提高了代码复用。

（综合 3）观察者模式设计智能家居系统

观察者模式“定义对象间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。”

好处在于：

- “可以实现表示层和数据逻辑层的分离”；
- “在观察目标和观察者之间建立一个抽象的耦合”；
- “支持广播通信，简化了一对多系统设计的难度”；
- “符合开闭原则，增加新的具体观察者无须修改原有系统代码，在具体观察者与观察目标之间不存在关联关系的情况下，增加新的观察目标也很方便”。

缺点在于：

- “将所有的观察者都通知到会花费很多时间”；
- “如果存在循环依赖时可能导致系统崩溃”；
- “没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而只是知道观察目标发生了变化。”

UML 图略。

（综合 4）策略模式算运费

(1) “策略模式封装了每一个算法，并且使得它们能够相互替换，从而使得算法可以独立于使用它的客户变化。”

(2) 先继承：`public class BzStrategy implements Strategy {`
再重载抽象方法：`(public) double Computing(double originalPrice)`
方法主体：`{ return originalPrice * 0.05 < 10 ? 10 : originalPrice * 0.05; } }`
其他策略类似，在此略。

(3) 表面上，多态发生在重载 Computing 方法中。实际上，当使用策略时，也会发生两次多态：
`Strategy ps = new PcStrategy();` 一次多态：ps 的类型是接口。
`var price = ps.Computing(10);` 第二次多态，调用时，JVM 知晓到底是什么类型（动态绑定）。
*策略的替换体现运行时行为的多态，其最大的价值并不在于方法不一样，而是**对象可替换**。*

（综合 5）单例模式与反射攻击

(1) 饿汉式实现重点在于饿，它会不计代价地生成。*相反，懒汉式实现由于懒，只有在使用时才会生成。*

```
3     public class Singleton  2 个用法
4     {
5         public static final Singleton Instance = new Singleton(10);  0 个用法
6
7         private int Id;  4 个用法
8         public int GetId() { return Id; }  0 个用法
9         public void SetId(int id) { Id = id; }  0 个用法
10
11        private Singleton(int id) { Id = id; }  1 个用法
12
13        public void Display() { System.out.println("Singleton ID: " + Id); }  0 个用法
14    }
15
```

(2) 饿汉式简单、稳定、线程安全，其在类加载时就已经创建了唯一实例。但这也会带来困扰：可能会浪费资源、无法延迟初始化（根据运行时参数动态配置）、可能会引发连锁类加载等。

(3) **默认情况下**，饿汉式实现无法抵抗反射。准确地说，私有构造方法无法抵抗反射。更准确地说，除被 final 修饰的成员均可被反射修改。

第一， 设定构造方法：constructor = objectClass.getDeclaredConstructor();

第二， 设置可见性：constructor.setAccesible(true);

第三， 创建新实例：newInstance = constructor.newInstance();

你可以在构造方法中使用条件判断抵抗反射。