22 春期末参考答案。标注(24)的题目标明本题与 24 试题考法相同或完全一致。
编写：大半凉；审核、校对：大半凉、鸠鸠咕咕

选择

1. B
2. C
3. C：Error 属于 JVM 层级上的严重错误。
4. A：构造方法没有返回类型，也不会返回任何对象引用。
5. A：解释见下文。
6. C：`HashSet<T>`实现了 `Iterable<T>`接口。
7. D
8. A：多数 Office 文件均能视为压缩文件。
9. C：`List.add()`只能传入泛型或其子类类型的参数。
10. A

根据 Javadoc 定义，父类的私有成员不会被继承。

Private Members in a Superclass

A subclass does not inherit the members of its parent class. Howe

A nested class has access to all the private members of its enclos

该题目描述存在问题。“属性”是一个面向对象的设计概念，通常指被 getter/setter 访问方法包装的字段。Java 并不存在语法意义上的“属性”。

属性的存在可以：

1. 令字段访问可控，即封装。我们可以通过访问方法独立读写权限、增加合法性校验、写入转换、懒加载或缓存以及只读限制等等；
2. 保持稳定的 API 接口。不修改类使用代码逻辑下，使用属性可随时引入额外的逻辑，甚至修改内部的存储结构；
3. 可以与高级访问机制连接，如通知、事件、绑定等等或实现重载（虚属性）。

字段不能完全做到。“私有属性”对任何外部都没有意义。

判断

1. ×
2. ×
3. ×：默认访问修饰符限制了类能被同一软件包内的其他类访问。
4. √
5. ×
6. √
7. ×
8. √
9. ×
10. √

填空

1. 面向对象特性：**封装**、**继承**、**多态**。
2. 设计模式：**创建型模式**、结构型模式、行为型模式。
3. 白黑测试定义了一些逻辑覆盖的指标，由逻辑从轻到强包括：语句、判定、条件、判定条件、条件组合和**路径覆盖**。
4. 黑盒测试包括：等价类划分、边界值分析、**场景法**。
5. I/O 处理方式：字节流(Input/**OutputStream**)、字符流(**Reader**/Writer)。
6. 事件处理机制：事件源、对象（事件本身）、**监听器**。
7. 通配符 **super** 代表父类；反之，extends 代表子类。

抽象类和接口异同（24）

	属性	方法	继承	作用
抽象类	没有限制（甚至可以声明构造方法）	非抽象方法必须声明方法主体	子类仅允许继承本类，无法继承其他类	描述类的共性，将一类事物抽象并提供默认实现与共享状态
接口	仅允许声明公开 public、静态 static 的 final 字段	不允许声明公开方法的主体，除被 default 或 static 关键字修饰；	在继承一个类之外，可以继承多个接口；	约定协议或定义能力
相同点	-	可声明带有主体的私有方法	必须声明接口中声明的方法（或抽象类的抽象方法）主体，除非继承类是抽象类	提高复用性（废话！）

成员：类中的所有被声明的东西。下文中 ms365、test2()等等都是成员。

字段：类中没有访问器的（不带逻辑的）变量。下文中 ms365 就是字段。

属性：具有公开访问器的字段。下文中没有属性。

方法：类中声明的函数。匿名方法：使用 lambda 表达式声明的方法。

方法主体：方法下被大括号包含的区域。

```
3   public interface ITester  1 个用法   1 个实现   新 *   1 个相关问题
4   {
5       public static final int ms365 = 666;  0 个用法
6       public abstract void test2();  0 个用法   新 *
7
8       private void helper() { }  0 个用法   新 *
9       public default void test() { }  0 个用法   新 *
10      public static void test1() { }  0 个用法   新 *
11  }
12
```

```
3   public abstract class Tester  1 个用法   1 个继承者   新 *   1 个相关问题
4   {
5       public static int ms365 = 666;  0 个用法
6       private static int windows98 = 999;  0 个用法
7
8       public int office2024 = 888;  0 个用法
9       private int windowsxp = 111;  0 个用法
10
11      public abstract void test2();  0 个用法   新 *
12      public void test1() { }  0 个用法   新 *
13
14      private void helper() { }  0 个用法   新 *
15
16      public Tester() { }  0 个用法   新 *   1 个相关问题
17  }
18
```

解答题见仁见智，默认情况下思路相同即可，无需咬文嚼字。某些答案由 ChatGPT 生成（由下划线标记）并已通过作者检查。

等价类测试

等价类测试类似于抽样调查，它把所有可能的输入划分成若干具有相同性质的输入子集，然后从每个子集中挑选少量代表性数据进行测试。

测试的理论基础是假设同一个等价类中的所有输入，对程序的逻辑应该产生相同的结果。因此不必把所有输入都测试一遍，只要测试每个等价类的代表值，就能有效覆盖输入空间。

等价类划分的原则有多个，但都与不等式类似。

1. 区间内部的 1 个有效等价类，和大于/小于本区间的 2 个无效等价类；
2. 有限个（n）枚举的 n 个有效等价类，和除此之外的 1 个无效等价类；

以下原则归属 2：

3. 决定性约束的 1 个有效等价类，和除此之外的 1 个无效等价类；
4. 布尔表达式成功的 1 个有效等价类，和失败的 1 个无效等价类。

List 的不同适用场景

24 出过。

ArrayList 以索引为优先访问器，其本质是（内部逻辑实现的）可变长度的原始类型数组。因此，对其的随机访问性能更好；

LinkedList 以迭代为优先访问器，其本质是双向链表。因此，对其进行队列操作、for-each 索引、插入或删除操作时性能更好，但内存开销更大。

工厂模式

简单工厂只是为产品新增了一个类，所有型号的产品（以及决断产品线的逻辑）均在此类构建；工厂模式则将构建方法下放到各个子类中，令用户决断生产什么产品。

MVC 模式

Model（模型）只负责逻辑；View（视图）只负责渲染；Controller（控制器）负责二者的协调。

Swing ~~纯构式~~一定程度上参考了 MVC，比如：控件继承了某些模型（它的逻辑因此定死）；控件本身就是视图；事件处理逻辑由 UI 组件内部完成（你自己写的，比如按钮点击事件，等等）。

多线程实现

一直推荐使用 Runnable 接口的原因，其一就是因为它是接口，同时可继承其他接口或类；此外，线程只能启动一次，Thread 类无法被多个线程重用，而 Runnable 可以包装在 Thread 中。

其他的，更现代的并发框架（Executors、ThreadPool 等等）均基于 Runnable 的设计理念。

智能家居系统

观察者模式定义了一种对象间的一对多的关系：当观察对象被触发时，它会自动通知已订阅的对象并作出响应，无需主动调用或轮询。

观察者模式降低了对象之间的耦合性，也容易扩展：仅需增加订阅。此外，它的自动更新机制也降低了部分性能开销，这是它的优势。缺点在于调试时极其复杂。当通知链过长时，错误来源不易定位（多个观察者嵌套），这也会带来性能上的问题（链式触发），同时顺序难以控制（多个订阅者的通知顺序需要在代码内写死）。

而且会有内存泄漏风险，观察者未解除订阅时，Subject 始终持有引用，这尤其在事件系统中常见。

UML 图略。

策略计算器(24)

(1) 策略模式是用来将各种具体的方法抽象出一个行为的方案。

好比商城的促销：原价、打折、买一送一等等，若将在原始代码中新建 if/else 语句体，会显得十分臃肿，并且不利于代码维护，也不符合设计原则。

业务逻辑经常变化、算法需要动态替换、可扩展性强的对战平台或模拟系统、条件判断容易膨胀的地方均适用于策略模式。这些场景均与一种设计类似，即可插拔，或即插即用（Plug-and-Play）。

(2)

各策略先继承接口：`public class OperationAdd implements Strategy {`

实现接口方法：`(public) double Computing(double num1, double num2)`

实现方法主体：`{ return num1 + num2; } }`

(3) 11.1；8.9。

(4) 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：

`Context context = new Context(new OpreationAdd());` 一次多态：参数的类型是接口。

ExecuteStrategy 实现了第二次多态，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**。*

单例模式与反射攻击

(1) 单例模式保证了实例的唯一性。

(2) 饿汉式实现重点在于饿，它会不计代价地生成。因此，饿汉式实例在类中声明并初始化：

```
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
```

相反，懒汉式实现由于懒，只有在使用时才会生成。

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， 获取反射类：objectClass = Singleton.class;

注意Class.class 与Object.getClass()的区别：前者使用类，后者使用实例。

第二， 设定构造方法：constructor = objectClass.getDeclaredConstructor();

第三， 设置可见性：constructor.setAccesible(true);

第四， 创建新实例：newInstance = constructor.newInstance();

(4) 你可以使用条件判断抵抗反射。反射编译通过，但会反馈运行时错误。

```
3   public class Singleton  2 个用法   新 *
4   {
5       public static final Singleton Instance = new Singleton(10);  1 个用法
6
7       private final int Id;  3 个用法
8       public int getId() { return Id; }  0 个用法   新 *
9
10      private Singleton(int id)  1 个用法   新 *
11      {
12          if (Instance != null) throw new IllegalStateException("别想拿反射搞我。");
13          Id = id;
14      }
15
16      public void Display() { System.out.println("Singleton ID: " + Id); }  0 个用法   新 *
17  }
18
```