

**软件体系结构作业11**

姓 名 ： 洪伟鑫

专 业 ： 软件工程

年 级 ： 2022级

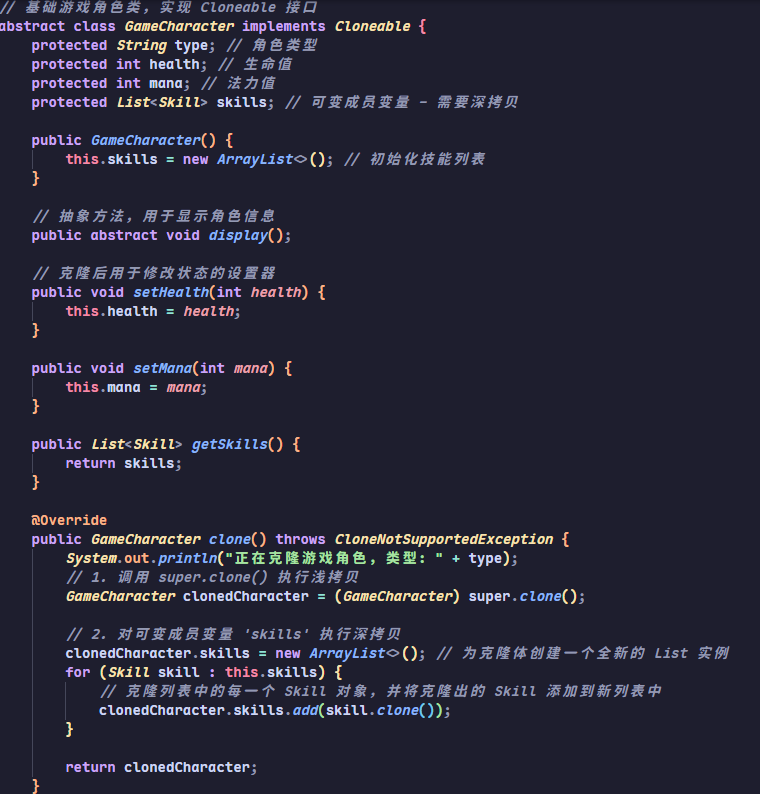
学 号 ： 37220222203612

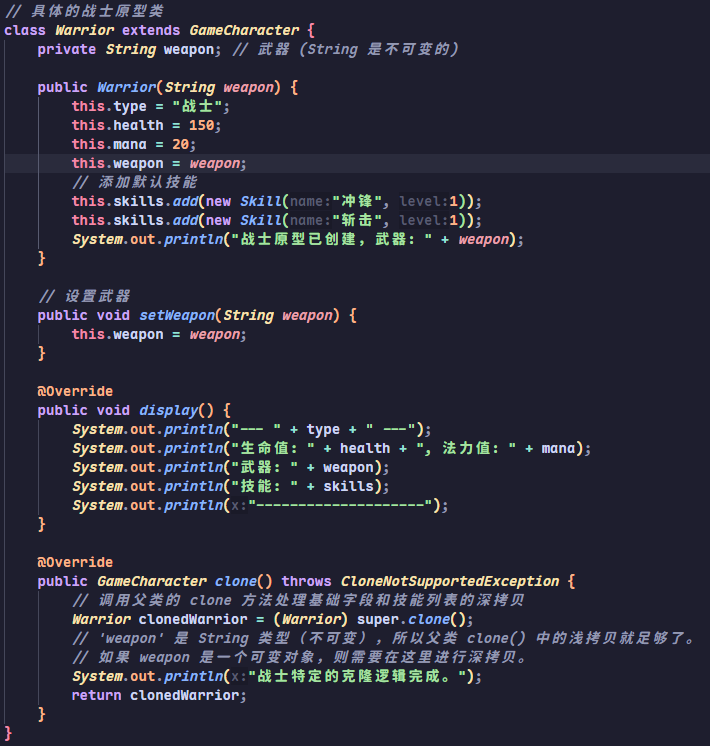
**2025年4月20日**

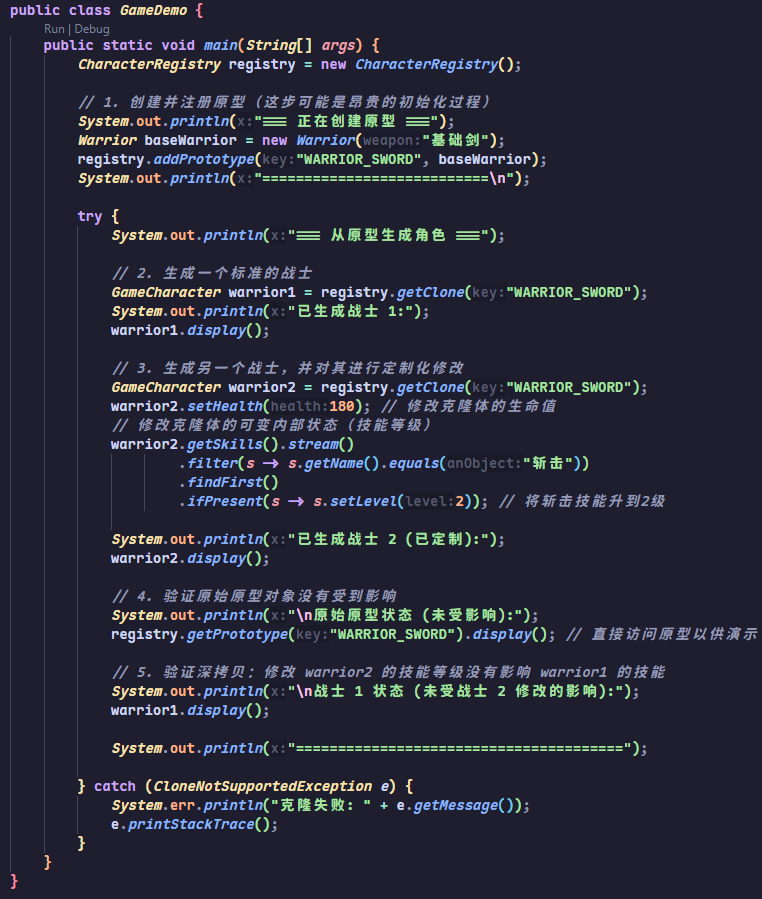
1、请举例说明克隆模式的其他应用。

**场景描述：** 在游戏中，可能存在不同类型角色的基础模板（例如：战士、法师、弓箭手）。创建一个新角色可能涉及加载资源（模型、纹理）、设置基础属性、分配默认技能等，这些操作可能非常耗费资源。使用原型模式允许我们创建预先配置好的模板，并在需要生成新实例时快速克隆它们，之后可以对克隆体进行少量修改。

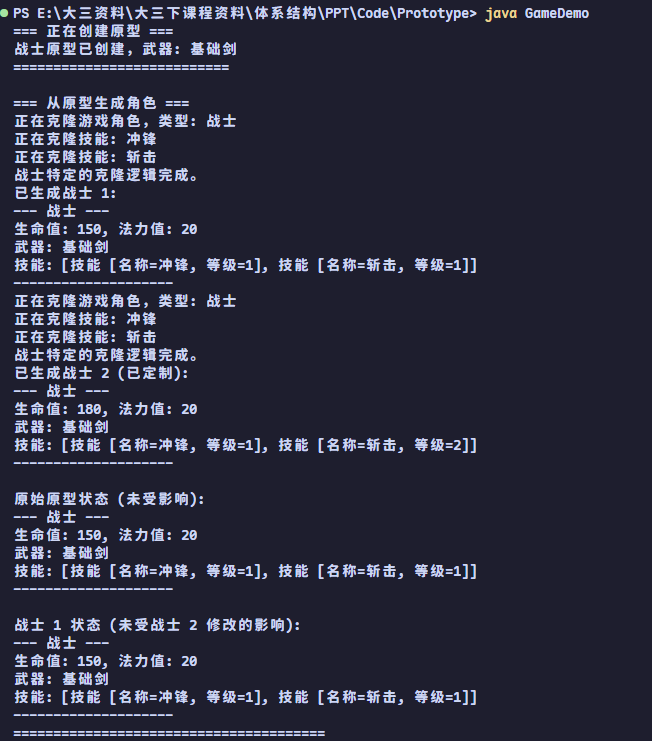








运行截图：



2、试描述浅克隆和深克隆。

|  |  |  |
| --- | --- | --- |
| 特性 | 浅克隆 (Shallow Clone) | 深克隆 (Deep Clone) |
| 基本定义 | 创建一个新对象，然后将原始对象中的**非静态字段的值**复制到新对象。 | 创建一个新对象，并将原始对象中的**非静态字段的值**复制到新对象。对于引用类型的字段，它会**递归地**复制这些字段所引用的对象。 |
| 处理基本数据类型 (如 int, float, boolean) | 直接复制值。克隆对象和原始对象中的基本类型字段是**完全独立**的。 | 直接复制值。克隆对象和原始对象中的基本类型字段也是**完全独立**的。 |
| 处理引用数据类型 (如 String, Object, List, 数组等) | 仅复制**引用地址**（内存地址）。这意味着克隆对象和原始对象中的引用类型字段将**指向堆内存中同一个对象实例**。 | 创建并复制引用所指向的**对象本身**。这意味着克隆对象和原始对象中的引用类型字段将指向**不同的、但内容相同**的对象实例。 |
| 对象独立性 | **不完全独立**。如果修改原始对象或克隆对象中的**引用类型成员变量所指向的对象内容**，这种修改会同时反映在另一个对象上（因为它们共享同一个引用对象）。 | **完全独立**。修改克隆对象的任何部分（包括其内部引用的对象）**不会**影响原始对象，反之亦然。 |
| 实现方式 (Java 示例) | 通常只需调用 super.clone() 方法即可（假设父类实现了正确的浅克隆逻辑）。 | 在调用 super.clone() 获得浅克隆副本后，需要**额外编写代码**，为所有**可变的引用类型**成员变量**也调用它们的** clone() **方法**（或其他复制方法），以实现递归复制。 |
| 性能开销 | 较低，速度较快，因为只复制引用地址，不复制引用对象本身。 | 较高，速度较慢，因为需要递归地创建和复制所有引用的对象。 |
| 适用场景 | 当对象只包含基本类型和不可变类型（如 String）时，或者明确希望共享引用对象状态时。 | 当需要一个完全独立的对象副本，不希望原始对象和克隆对象之间存在任何共享状态（尤其是可变状态）时。 |
| 潜在问题 | 容易因共享可变引用对象而引发意外的副作用和 bug。 | 实现相对复杂，需要仔细处理对象图中的所有引用关系，可能需要处理循环引用问题。 |

**总结来说：**

* **浅克隆**：只复制一层，对象内的引用变量还是指向原来的内存地址。像“复制了房子的地址，没复制房子本身”。
* **深克隆**：层层复制，直到所有引用的对象都被复制了一份新的。像“不仅复制了房子的地址，连房子本身和里面的家具都复制了一套全新的”。