

**面向对象程序设计上机实验报告**

实验题目1：一个简单游戏算法的实现

**学院名称 智能与计算学部**

**专 业 软件工程**

**学生姓名 王杨森**

**学 号 3020244116**

**年 级 2020级**

**班 级 软件工程3班**

**时 间 2022年 3月14日**

1. **实验目的**

1) 理解面向对象的概念；

2) 理解继承、封装、多态的实现；

3) 练习使用DEBUG工具调试程序；

1. **实验内容**
2. 某一款游戏，其主要角色如下:
3. HeavyTank 重型坦克 初始生命值200，攻击力 20
4. Medium Tank 轻型坦克 初始生命值100，攻击力 10
5. War Factroy 兵工厂 初始生命值100，无攻击力
6. Barrack 兵营，可以训练出步枪兵、 RPG兵、军犬,初始生命值100，无攻击力
7. Rifle soldier 步枪兵 初始生命值50(对战 军犬除外)，攻击力 5（对战军犬可以一次击毙军犬)
8. Rocket soldier 火箭兵 初始生命值50(对战 军犬除外)，攻击力 10
9. Dog 军犬 ,初始生命值50，攻击力5(对战人类时候一口毙命)
10. 此外还要能通过Solider.getLivingSoilderCount/getDeadedSoilderCount 统计现在有多少个活着的和死去的士兵数量
11. 攻击，两个对象之间相互攻击，削减每个对象的健康值。
12. 请遵循以上游戏规则，设计并实现游戏代码，使其能够通过测试用例
13. **程序实现**

**public** **class** GameObject {

**protected** **int** x, y;

**protected** **int** health;

**public** GameObject(**int** x, **int** y, **int** health) {

**this**.x = x;

**this**.y = y;

**this**.health = health;

}

//获取生命值

**public** **int** getHealth() {

**return** **this**.health;

}

//判断是否被摧毁

**public** **boolean** isDestroyed(){

**if**(**this**.health <= 0) **return** **true**;

**else** **return** **false**;

}

//每个游戏对象都会有的受到伤害函数

**public** **void** getHurt(**int** attack) {

**if**(**this**.health - attack > 0) **this**.health -= attack;

**else** **this**.health = 0;

}

//直接摧毁该对象

**public** **void** beDead() {

**this**.health = 0;

}

}

**public** **class** GameBase **extends** Buildings{

**public** GameBase(**int** x, **int** y) {

**super**(x, y, 500);

}

**public** **static** GameBase createGameBase(**int** x, **int** y) {

GameBase newBase = **null**;

newBase = **new** GameBase(x, y);

**return** newBase;

}

**public** Buildings building(EnumObjectType type, **int** x, **int** y) {

Buildings newBuilding = **null**;

**switch** (type) {

**case** ***barrack***:

newBuilding = **new** Barrack(x, y);

**break**;

**case** ***warFactory***:

newBuilding = **new** WarFactory(x, y);

**break**;

**default**:

**break**;

}

**return** newBuilding;

}

}

**public** **enum** EnumObjectType {

***barrack***, ***warFactory***, ***rifleSoldier***, ***RPGSoldier***, ***mediumTank***, ***heavyTank***, ***dog***,

}

//成员移动的接口

**public** **interface** Move {

**void** move(**int** dx, **int** dy);

}

//可以移动的游戏角色

**public** **class** Roles **extends** GameObject **implements** Move{

**protected** **int** attack;

**protected** **double** range;

**public** Roles(**int** x, **int** y, **int** health, **int** attack, **int** range) {

**super**(x, y, health);

**this**.attack = attack;

**this**.range = range;

}

@Override

**public** **void** move(**int** dx, **int** dy) {

**this**.x += dx;

**this**.y += dy;

}

//每个角色可以攻击

**public** **void** attack(GameObject obj) {

**if**(obj.isDestroyed()) **return**;

**if**(Utils.*getDistance*(**this**, obj) <= **this**.range){

obj.getHurt(**this**.attack);

}

**if**(obj.isDestroyed()){

**if**(obj **instanceof** RPGSoldier || obj **instanceof** RifleSoldier) Soldier.*killSoldier*();

}

}

}

//将建筑物进行一个封装

**public** **abstract** **class** Buildings **extends** GameObject{

**public** Buildings(**int** x, **int** y, **int** health){

**super**(x, y, health);

}

}

**public** **class** Barrack **extends** Buildings{

**public** Barrack(**int** x, **int** y) {

**super**(x, y, 100);

}

**public** Roles traing(EnumObjectType type){

Roles newRole = **null**;

**switch** (type){

**case** ***dog***:

newRole = **new** Dog(**this**.x, **this**.y);

**break**;

**case** ***RPGSoldier***:

newRole = **new** RPGSoldier(**this**.x, **this**.y);

Soldier.*addSoldier*();

**break**;

**case** ***rifleSoldier***:

newRole = **new** RifleSoldier(**this**.x, **this**.y);

Soldier.*addSoldier*();

**break**;

**default**:

**break**;

}

**return** newRole;

}

}

**public** **class** RifleSoldier **extends** Roles{

**public** RifleSoldier(**int** x, **int** y){

**super**(x, y, 50, 5, 5);

}

@Override

//重写这个函数，针对狗一击致死

**public** **void** attack(GameObject obj){

**if**(obj.isDestroyed()) **return**;

**if**(obj **instanceof** Dog){

obj.beDead();

}

**else** **super**.attack(obj);

}

}

//将获取距离的函数进行封装

**public** **class** Utils {

**public** **static** **double** getDistance(GameObject obj1, GameObject obj2) {

**return** Math.*sqrt*((obj1.x - obj2.x) \*(obj1.x - obj2.x) + (obj1.y - obj2.y) \* (obj1.y - obj2.y));

}

}

**public** **class** WarFactory **extends** Buildings{

**public** WarFactory(**int** x, **int** y) {

**super**(x, y, 100);

}

**public** Tank building(EnumObjectType type) {

Tank newTank = **null**;

**switch** (type){

**case** ***heavyTank***:

newTank = **new** HeavyTank(**this**.x, **this**.y);

**break**;

**case** ***mediumTank***:

newTank = **new** MediumTank(**this**.x, **this**.y);

**break**;

**default**:

**break**;

}

**return** newTank;

}

}

//对两种坦克的抽象

**public** **abstract** **class** Tank **extends** Roles{

**public** Tank(**int** x, **int** y, **int** health, **int** attack, **int** range){

**super**(x, y, health, attack, range);

}

}

**public** **class** Soldier {

**private** **static** **int** *livingSoldierCount* = 0;

**private** **static** **int** *deadSoldierCount* = 0;

**public** **static** **int** getLivingSoldierCount() {

**return** *livingSoldierCount*;

}

**public** **static** **int** getDeadedSoldierCount() {

**return** *deadSoldierCount*;

}

**public** **static** **void** addSoldier() {

*livingSoldierCount*++;

}

**public** **static** **void** killSoldier() {

*livingSoldierCount*--;

*deadSoldierCount*++;

}

}

**public** **class** Dog **extends** Roles{

**public** Dog(**int** x, **int** y) {

**super**(x, y, 50, 5, 5);

}

@Override

//重写攻击函数，对人有特殊情况

**public** **void** attack(GameObject obj) {

**if**(obj.isDestroyed()) **return**;

**if**(obj **instanceof** RPGSoldier || obj **instanceof** RifleSoldier) {

obj.beDead();

Soldier.*killSoldier*();

}

**else** **super**.attack(obj);

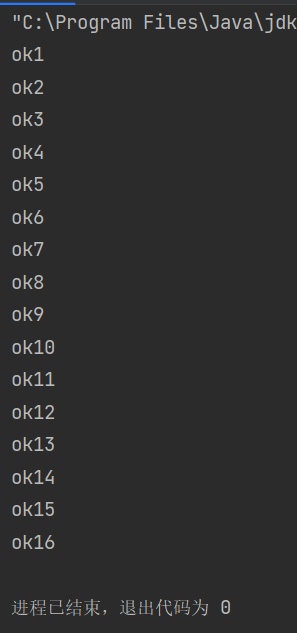
}

}

1. **实验结果**

编写测试用例并进行验证：

 程序运行结果如下图所示：



1. **实验中遇到的问题及解决方法**

在这次的实验过程中，首先我要明确如何设计游戏中使用到的类，为了更好的封装和继承，先设计一个GameObject类表示所有对象共有属性，Buildings和Roles都是对他的继承，然后对于Roles可以移动，设计一个move接口，便于程序的进一步维护，最大的困难点在于通过一个方法去创建新的类，即方法的返回类型是一个类，这就需要返回的类是父类，然后再做类型转换成为我们想要的类，在这个过程中我上网搜集了一些资料，并查看了课程示例代码学习类的继承和接口的使用，其次在构建函数时为了方便不同类型的对象的构建，我使用了多态的方法，学习了相关知识的运用。

对于Soldier这个类，由于需要全局的统计活着的和死去的士兵数，需要对这两个变量设置为static，并且将其中的方法设置为静态方法，以便于所有对象都能够共同修改这一部分的变量值。

通过这次实验，加深了对于面向对象概念的理解，复习了继承、封装、多态等相关知识的运用，同时加强了搜集资料，学习新技能的能力。