ГОУ ВПО

«Дальневосточный государственный университет

путей сообщения»

Лабораторная работа №2  
Наследование, абстрактные классы, члены класса. Интерфейсы.

Выполнил: Точилкин М.А 230 группа

Проверил: Латышевич Е.О.

Хабаровск, 2016

Цель: изучить наследование, абстрактные классы, члены класса, интерфейсы.

Задание: Предметная область: Royal Playing Game (RPG). Компьютерная игра. Реализовать абстрактный класс TargetObject (объект цель позволяющий оперировать своим здоровьем) Имеющий следующие члены:

Абстрактное свойство Life (жизни).

Реализовать абстрактный класс Player (представляет собой сущность игрока), унаследовавший абстрактный класс TargetObject. Имеющий следующие члены:

Констркутор по умолчанию, конструктор с инициализацией полей.

Реализовать свойство Life (жизни героя).

Реализовать свойство Strange (Сила героя).

Реализовать свойство Dexterity (ловкость героя).

Реализовать свойство Intelligence (интеллект героя).

Реализовать свойство Effects (возвращает полный список эффектов наложенных на героя (открытый блок get, закрытый блок set)).

Реализовать свойство Level (текущий уровень героя (открытый блок get, закрытый блок set)).

Реализовать индексатор для получения эффекта по его имени или уникальному номеру.

Объявить абстрактный метод Attack принимающий в себя объект типа TargetObject.

Объявить абстрактный метод EquipWeapon принимающий в себя объект типа Weapon.

Объявить абстрактный метод EquipOutfit принимающий в себя объект типа Armor.

Объявить абстрактный метод AddEffect принимающий в себя объект типа Effect.

Реализовать класс Wizard, унаследовавший абстрактный класс Player (класс Чародей, маг, волшебник, может носить только тканевую броню, а в качестве оружия только посохи и жезлы). Имеющий следующие члены:

Конструктор по умолчанию. Конструктор с инициализацией полей.

Реализовать все абстрактные методы и свойства класса родителя.

Реализовать свойство Mana (мана героя).

Реализовать реализовать следующие константы: тип возможного оружия, тип возможной брони. (Подумайте как их можно реализовать? Какую структуру необходимо выбрать?)

Реализовать метод LevelUp. Увеличивающий уровень героя и его параметры.

Реализовать открытый метод ChanceToDodgeAttack. Возвращает шанс уворота от атаки противника. Расчитывается от показателя ловкость.

В методе Attack предусмотреть проверку на возможность атаки героя. Закрытый метод CanAttack для проверки наложенного эффекта оглушения и экипированного оружия.

В методе AddEffect добавить проверку на уже наложенный эффект на героя.

В методе EquipWeapon и EquipOutfit добавить проверку на возможность экипировки оружия. Если экипировать оружия невозможно, выбросить исключение типа PlayerOutfitException.

Реализовать закрытое свойство возвращающие величину на которое увеличивается атака героя. Рассчитывается от показателя интеллект.

Реализовать класс Effect (эффект наложенный на игрока). Имеющие следующие члены:

Тип накладываемого эффекта. EffectType.

Время действия эффекта. Duration.

Свойство возвращающие значение отвечающие на вопрос постоянный это эффект или временный?

Унаследовать интерфейс IEquatable. Переопределить методы GetHashCode и Equals. Переопределить операторы != и ==

Реализовать исключение типа PlayerOutfitException (для сообщения об исключительной ситуации с экипированием оружия или брони).

Реализовать класс Weapon (оружие):

Реализовать свойство базовый урон. Damage.

Реализовать свойство тип оружия. WeaponType.

Реализовать свойство требуемая сила, ловкость, или интеллект. Requirements.

Реализовать класс Armor (броня):

Реализовать свойство базовая защита. Defense.

Реализовать свойство тип брони. ArmorType.

Реализовать свойство требуемая сила, ловкость, или интеллект. Requirements (подумайте как здесь избавиться от дублирующегося кода?).

using System;

namespace Laba2

{

class Wizard : Player

{

public int Mana { get; set; }

public Wizard()

{

Life = 100;

Strange = 10;

Dexterity = 10;

Intelligence = 10;

}

public Wizard(int Life, int Strange, int Dexterity, int Intelligence, int Mana)

{

this.Life = Life;

this.Strange = Strange;

this.Dexterity = Dexterity;

this.Intelligence = Intelligence;

this.Mana = Mana;

}

public void LevelUp()

{

Level += 1;

Strange += 3;

Dexterity += 3;

Intelligence += 3;

}

protected int AttackUp

{

get { return Intelligence \* 5; }

}

public int ChanceToDodgeAttack(int attack)

{

return Dexterity / attack / 100;

}

public bool CanAttack(Effect effect)

{

bool status = true;

foreach (string name in Enum.GetValues(typeof(Effect.EffectType)))

{

if (effect.EffectType == EffectType.stun)

{

status = false;

}

}

return status;

}

public override void Attack(TargetObject targetObject)

{

if (CanAttack())

{

targetObject.Life = targetObject.Life - weapon.Damage;

}

}

public override void EquipWeapon(Weapon weapon)

{

if (weapon.Requirements != new Requirements(Strange, Dexterity, Intelligence))

{

new PlayerOutfitException("НЕДОСТАТОЧНО СТАТОВ");

}

if (weapon.WeaponType != weapon.WeaponType.staff)

{

new PlayerOutfitException("ТИП ОРУЖИЯ НЕ ПОДХОДИТ");

}

}

public override void EquipOutfit(Armor armor)

{

if (armor.Requirements != new Requirements(Strange, Dexterity, Intelligence))

{

new PlayerOutfitException("НЕДОСТАТОЧНО СТАТОВ");

}

if (armor.ArmorType != armor.ArmorType.coat)

{

new PlayerOutfitException("ТИП БРОНИ НЕ ПОДХОДИТ");

}

}

public override void AddEffect(Effect effect)

{

if (effect.Duration == -1)

{

effect = new Effect(effect.EffectType, effect.Duration);

}

}

}

}

using System;

namespace Laba2

{

public class Weapon

{

public int Damage { get; set; }

public Requirements Requirements { get; set; }

public enum WeaponType { bones, arms, stick, rock, staff }

public Weapon(int Damage, enum WeaponType, Requirements Requirements)

{

this.Damage = Damage;

foreach (string name in Enum.GetValues(typeof(WeaponType)))

{

if (WeaponType == (string)name)

{

this.WeaponType = name;

}

else

{

this.WeaponType = WeaponType.bones;

}

}

this.Requirements = Requirements;

}

}

}

namespace Laba2

{

abstract class TargetObject

{

public abstract int Life { get; set; }

}

}

namespace Laba2

{

public class Requirements

{

public int Strange { get; set; }

public int Dexterity { get; set; }

public int Intelligence { get; set; }

public Requirements()

{

Strange = 15;

Dexterity = 15;

Intelligence = 15;

}

public Requirements(int Strange, int Dexterity, int Intelligence)

{

this.Strange = Strange;

this.Dexterity = Dexterity;

this.Intelligence = Intelligence;

}

}

}

using System;

namespace Laba2

{

public class PlayerOutfitException : ApplicationException

{

public PlayerOutfitException() { }

public PlayerOutfitException(string message) : base(message) { }

protected PlayerOutfitException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext contex)

: base(info, contex) { }

}

}

namespace Laba2

{

abstract class Player: TargetObject

{

public override int Life { get; set; }

public int Strange { get; set; }

public int Dexterity { get; set; }

public int Intelligence { get; set; }

public int Effects { get; protected set; }

public int Level { get; protected set; }

public Weapon Weapon { get; set; }

public Armor Armor { get; set; }

public Player()

{

Life = 100;

Strange = 10;

Dexterity = 10;

Intelligence = 10;

Effects = 0;

Level = 1;

Armor = new Armor(20, Armor.ArmorType.sportivki, new Requirements(1,1,1));

Weapon = new Weapon(10, Weapon.WeaponType.arms, new Requirements(1,1,1));

}

public Player(int Life, int Strange, int Dexterity, int Intelligence, int Effects, int Level, Armor Armor, Weapon Weapon)

{

this.Life = Life;

this.Strange = Strange;

this.Dexterity = Dexterity;

this.Intelligence = Intelligence;

this.Effects = Effects;

this.Level = Level;

this.Armor = Armor;

this.Weapon = Weapon;

}

public abstract void Attack(TargetObject targetObject, Effect effect, Weapon weapon);

public abstract void EquipWeapon(Weapon weapon);

public abstract void EquipOutfit(Armor armor);

public abstract void AddEffect(Effect effect);

}

}

using System;

namespace Laba2

{

class Effect : IEquatable<Object>

{

public enum EffectType { stun, poison, bite, saliva, mucus, webs, frost }

public double Duration { get; set; }

public Effect(string EffectType, double Duration)

{

this.EffectType = EffectType;

this.Duration = Duration;

}

public bool EffectDuration

{

get

{

if (Duration == -1)

{

return true;

}

else

{

return false;

}

}

}

public bool Equals(Effect obj)

{

if (obj == null)

{

return false;

}

Effect p = obj as Effect;

if ((Object)p == null)

{

return false;

}

return (EffectType == p.EffectType) && (Duration == p.Duration);

}

public int GetHashCode()

{

return EffectType.GetHashCode();

}

public static bool operator ==(Effect eff1, Effect eff2)

{

if ((eff1.EffectType == eff2.EffectType) && (eff1.Duration == eff2.Duration) && (eff1.EffectDo == eff2.EffectDo))

return true;

return false;

}

public static bool operator !=(Effect eff1, Effect eff2)

{

if ((eff1.EffectType != eff2.EffectType) || (eff1.Duration != eff2.Duration) || (eff1.EffectDo != eff2.EffectDo))

return true;

return false;

}

}

}

using System;

namespace Laba2

{

public class Armor

{

public int Defense { get; set; }

public enum ArmorType { leather, jacket, coat, sportivki, underpants, briefs }

public Requirements Requirements { get; set; }

public Armor(int Defense, enum ArmorType, Requirements Requirements)

{

this.Defense = Defense;

foreach (string name in Enum.GetValues(typeof(ArmorType)))

{

if (ArmorType == (string)name)

{

this.ArmorType = name;

}

else

{

this.ArmorType = ArmorType.underpants;

}

}

this.Requirements = Requirements;

}

}

}