**Programming paradigms – Inheritance & aggregation**

**LO1 – Research encapsulation & inheritance in C#**

**LO2 – Follow the control of a program and list possible outputs**

**LO3 – List the encapsulation and inheritance relationships between classes**

Below are 6 classes which work together to create a (very) simple text-based game. Study the classes, note how they link, and which methods are called within the Main class.

**Starter**

Spend 10 minutes working on your programs from last lesson.

**Evidence: paste a screenshot of your code here**

**Inheritance**

Inheritance is the practice of allowing “child” or “sub” classes to inherit the methods, variables, and sometimes constructors, of “parent” or “super” classes. Inheritance can be signified using the following syntax: Child : Parent.

**Inheritance example**

class Vehicle // base class (parent)

{

public string brand = "Ford";

public void honk()

{

Console.WriteLine("Tuut, tuut!");

}

}

class Car : Vehicle // derived class (child)

{

public string modelName = "Mustang";

}

class Program

{

static void Main(string[] args)

{

// Create a myCar object

Car myCar = new Car();

// Call the honk() method (From the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand field (from the Vehicle class) and the value of the modelName from the Car class

Console.WriteLine(myCar.brand + " " + myCar.modelName);

}

**Aggregation**

Aggregation comes under the umbrella of inheritance but is a different method for passing one classes parameters to another.

Aggregation involves having an instance of a class passed as a parameter in another class’s constructor. This can be seen in the alien and zombie classes in task 1.

**Task 1**

Each class is a different .cs file. Create a new solution in visual studio called InheritanceTest.

Create the respective class files within the solution and copy in the code.

Paste a screenshot below of the console running the code.

Comment through the code to understand each classes functionality.

**Enemy**

namespace InheritanceTest

{

public class Enemy

{

private bool holdWeapon;

private bool headshotOnly;

private bool isAlive;

public bool HoldWeapon

{

get

{

return holdWeapon;

}

set

{

holdWeapon = value;

}

}

public bool HeadshotOnly

{

get

{

return headshotOnly;

}

set

{

headshotOnly = value;

}

}

public bool IsAlive

{

get

{

return isAlive;

}

set

{

isAlive = value;

}

}

}

}

**Alien**

namespace InheritanceTest

{

public class Alien : Enemy

{

private string aRace, aComms;

private int aShield;

private Attribute aAttribute;

public Alien(string race, string comms, int shield, Attribute attribute)

{

aRace = race;

aComms = comms;

aShield = shield;

aAttribute = attribute;

}

public string Race

{

get

{

return aRace;

}

set

{

aRace = value;

}

}

public string Comms

{

get

{

return aComms;

}

set

{

aComms = value;

}

}

public int Shield

{

get

{

return aShield;

}

set

{

aShield = value;

}

}

public int Health

{

get

{

return aAttribute.Health;

}

set

{

aAttribute.Health = value;

}

}

public int Damage

{

get

{

return aAttribute.Damage;

}

set

{

aAttribute.Damage = value;

}

}

}

}

**Zombie**

amespace InheritanceTest

{

public class Zombie : Enemy

{

private string zType, zVoice;

private int zRespawns;

private Attribute zAttribute;

public Zombie(string type, string voice, int respawns, Attribute attribute)

{

zType = type;

zVoice = voice;

zRespawns = respawns;

zAttribute = attribute;

}

public string Type

{

get

{

return zType;

}

set

{

zType = value;

}

}

public string Voice

{

get

{

return zVoice;

}

set

{

zVoice = value;

}

}

public int Respawns

{

get

{

return zRespawns;

}

set

{

zRespawns = value;

}

}

public int Health

{

get

{

return zAttribute.Health;

}

set

{

zAttribute.Health = value;

}

}

public int Damage

{

get

{

return zAttribute.Damage;

}

set

{

zAttribute.Damage = value;

}

}

}

}

**Attribute**

namespace InheritanceTest

{

public class Attribute

{

private int aHealth, aStamina, aDamage;

public Attribute(int health, int stamina, int damage)

{

aHealth = health;

aStamina = stamina;

aDamage = damage;

}

public int Health

{

get

{

return aHealth;

}

set

{

aHealth = value;

}

}

public int Stamina

{

get

{

return aStamina;

}

set

{

aStamina = value;

}

}

public int Damage

{

get

{

return aDamage;

}

set

{

aDamage = value;

}

}

}

}

namespace InheritanceTest

{

public class Game

{

private string input;

private bool alive = true;

private void ZombieAttack(Zombie gZombie, Alien gAlien, bool block)

{

if(block == true)

{

Console.WriteLine("You blocked the attack!");

}

else if(gAlien.Shield > gZombie.Damage)

{

gAlien.Shield -= gZombie.Damage;

Console.WriteLine("Shield damaged! Health: {0} | Shield: {1}", gAlien.Health, gAlien.Shield);

}

else if(gAlien.Shield < gZombie.Damage && gAlien.Shield > 0)

{

gAlien.Shield = 0;

Console.WriteLine("Shield broken! Health: {0} | Shield: {1}", gAlien.Health, gAlien.Shield);

}

else if(gAlien.Shield <= 0 && gAlien.Health > 0)

{

gAlien.Health -= gZombie.Damage;

if(gAlien.Health < 0)

{

Console.WriteLine("You are dead");

alive = false;

}

else

{

Console.WriteLine("Health damaged! Health: {0} | Shield: {1}\n", gAlien.Health, gAlien.Shield);

}

}

else

{

Console.WriteLine("You are dead");

alive = false;

}

}

private void AlienAttack(Zombie gZombie, Alien gAlien)

{

if(gZombie.Health > 0)

{

gZombie.Health -= gAlien.Damage;

if(gZombie.Health > 0)

{

Console.WriteLine("{0} Damaged! Health: {1}\n", gZombie.Type, gZombie.Health);

}

else

{

char[] final = gZombie.Voice.ToCharArray();

foreach(char x in final)

{

Console.Write(x);

Thread.Sleep(50);

}

Console.WriteLine("\nThe {0} is dead!", gZombie.Type);

alive = false;

}

}

else

{

Console.WriteLine("The {0} is dead!", gZombie.Type);

alive = false;

}

}

public void Run(Zombie gZombie, Alien gAlien)

{

Console.WriteLine("You approach a {0}, it goes to attack you. \n\n", gZombie.Type);

while (alive == true)

{

Console.WriteLine("\nYou are dangerously close to the {0}, what do you do?", gZombie.Type);

input = Console.ReadLine();

if (input.ToUpper() == "ATTACK")

{

Console.WriteLine("The {0} hits you first", gZombie.Type);

ZombieAttack(gZombie, gAlien, false);

if (alive == true)

{

AlienAttack(gZombie, gAlien);

}

}

else if (input.ToUpper() == "BLOCK")

{

ZombieAttack(gZombie, gAlien, true);

}

else

{

Console.WriteLine("Nope, that didn't work. Zombie got a free hit\n\n");

ZombieAttack(gZombie, gAlien, false);

}

}

}

}

}

**Program**

namespace InheritanceTest

{

class Program

{

static void Main(string[] args)

{

bool menu = true;

Attribute z = new Attribute(100, 100, 20);

Attribute a = new Attribute(100, 100, 23);

Zombie walker = new Zombie("Walker", "eeuuurrghaah...", 0, z);

Alien martian = new Alien("Martian", "Merp", 32, a);

walker.HoldWeapon = false;

walker.HeadshotOnly = true;

walker.IsAlive = true;

Console.WriteLine("The Z Chronicles - ZED AGAIN");

while(menu == true){

Console.WriteLine("\*\*--- 1 : New Game ---\*\*");

Console.WriteLine("\*\*--- 2 : Quit Game ---\*\*");

int choice = int.Parse(Console.ReadLine());

if(choice == 1){

Game y = new Game();

menu = false;

y.Run(walker, martian);

}

}

Console.ReadKey();

}

}

}

**Task 2**

Answer the following questions:

Which class contains the method Run?

What is the purpose of this method?

Why are the variables in each class private, and called with get and set methods?

Where can aggregation be seen in this program?

**Modify**

1. Add a chance of a critical hit to the method ZombieAttack
2. Create new instances of the zombie and alien classes, run a new game using these instances

**Make**

Create a new class of your own enemy type e.g. goblin, soldier etc. This class should use aggregation in order to pass an attribute through it’s constructor, and inherit the properties from the Enemy class.