НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КПІ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп’ютерних систем

ЛАБОРАТОРНА РОБОТА №4

з дисципліни

«Об’єктно-орієнтоване програмування»

ТЕМА: «C# .Net. Управління ресурсами. Написання чистого коду»

Підготував: студент групи КП-52

Сорочинський Владислав

Перевірила:

Заболотня Тетяна Миколаївна

Київ 2016

**Мета роботи**

Ознайомитися з правилами написання чистого коду, вивчити механізм

управління ресурсами, реалізований у .Net.

**Постановка завдання**

1. Додати до класів методи, наявність яких дозволятиме управляти

знищенням об’єктів цих класів (IDisposable) (3 бали).

2. Забезпечити виклики методів GC таким чином, щоб можна було

простежити життєвий цикл об’єктів, що обробляються (зокрема,

продемонструвати можливості роботи з WeakReference) (5 балів).

3. Привести код класів (розроблених у попередніх лабораторних

роботах) у відповідність до евристичних правил щодо написання чистого

коду (груп “Функції”, “Імена”, “Загальні” з книги Р.Мартіна “Чистий код”), а

також до правил рефакторингу. До відповідних фрагментів коду написати

коментар із вказівкою правила, якому задовольняє цей фрагмент.

Застосувати мінімум 7 окремих правил написання чистого коду (правила, які

є дуже близькими за змістом, будуть зараховані як одне правило) та 5 правил

рефакторингу. (12 балів)

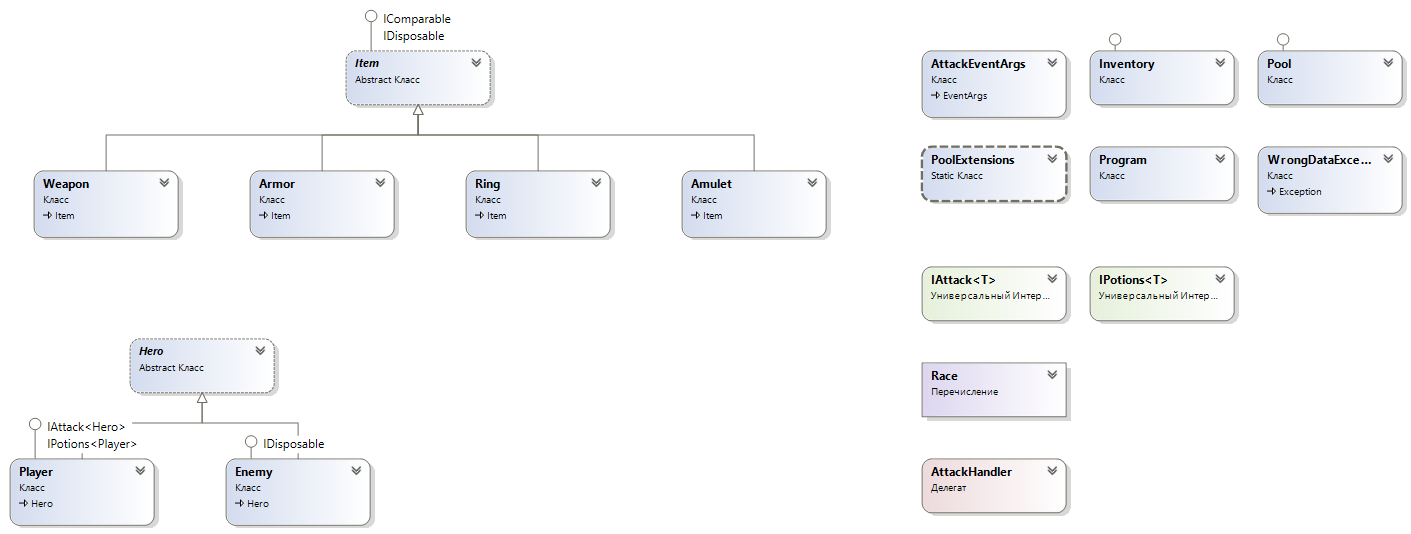
**Лістинг програми**

|  |
| --- |
| Hero.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Runtime.Serialization;  using System.Runtime.Serialization.Json;  using System.Xml.Serialization;  namespace lab4  {  interface IAttack<T> // рефакторинг - виділення інтерефейсу  {  void meleeAttack(T target); // Функції: змістовні імена  void magicAttack(T target);  }  interface IPotions<T>  {  void healthPotion();  void magickaPotion();  }  public enum Race  {  HUMAN = 0,  ORC,  ELFE  }  public delegate void AttackHandler(AttackEventArgs args);  public class AttackEventArgs : EventArgs  {  public Hero attacker;  public Hero target;  public AttackEventArgs(Hero attacker, Hero target)  {  this.attacker = attacker;  this.target = target;  }  }  [Serializable]  public abstract class Hero //Рефакторинг - уникнення дублювання коду для Player і Enemy  {  public string Name;  [XmlIgnore]  protected Race race;  [XmlIgnore]  public double health, magicka, BaseMeleeDamage, BaseMagicDamage;  public double meleeDamage, magicDamage;  protected int maxHealth, maxMagicka;  [XmlIgnore]  public Inventory inventory;  protected static int level;  static Hero()  {  level = 1;  }  public Hero()  {  }  protected Hero(string name, Race race) //Функції: один рівень абстракції, Загальні: горизонтальні розділення і стиснення  { // рефакторинг - заміщення алгоритму  this.Name = name;  this.race = race;  switch (race)  {  case Race.HUMAN:  this.maxHealth = 35;  this.maxMagicka = 30;  this.BaseMeleeDamage = 6;  this.BaseMagicDamage = 5;  break;  case Race.ORC:  this.maxHealth = 35;  this.maxMagicka = 25;  this.BaseMeleeDamage = 7;  this.BaseMagicDamage = 4;  break;  case Race.ELFE:  this.maxHealth = 30;  this.maxMagicka = 35;  this.BaseMeleeDamage = 4;  this.BaseMagicDamage = 7;  break;  default: // Функції: виключення замість коду помилки  throw new WrongDataException("Wrong race, can`t create Hero", race.ToString());  }  this.health = this.maxHealth;  this.magicka = this.maxMagicka;  }  public virtual void levelUp() { level++; }  protected void countDamage()  {  meleeDamage = BaseMeleeDamage;  magicDamage = BaseMagicDamage;  if (inventory.Weapon != null) meleeDamage = BaseMeleeDamage + inventory.Weapon.Stat;  if (inventory.Ring != null) magicDamage = BaseMagicDamage + inventory.Ring.Stat;  }  }  [Serializable]  [XmlInclude(typeof(Hero))]  [XmlInclude(typeof(Player))]  public class Player : Hero, IAttack<Hero>, IPotions<Player>  {  public int coins;  int healthPotions;  int magickaPotions;  [XmlIgnore]  public Pool pool;  public event AttackHandler MeleeAttackEvent;  public event Action<AttackEventArgs> MagicAttackEvent;  public event Func<Item, bool> BuyEvent;  public Player()  {  }  public Player(string name, Race race)  : base(name, race)  {  pool = new Pool();  inventory = new Inventory();  healthPotions = 3;  magickaPotions = 3;  coins = 500;  countDamage();  AttackHandler MeleeAttackMessage = delegate(AttackEventArgs eargs)  {  Console.WriteLine("{0} attacked {1} with {2} melee damage", eargs.attacker.Name, eargs.target.Name, eargs.attacker.meleeDamage);  };  Func<Item, bool> buyHandler = (Item item) =>  {  if (this.coins - item.Price >= 0)  {  return true;  }  return false;  };  Action<AttackEventArgs> MagicAttackMessage = eargs => Console.WriteLine("{0} attacked {1} with {2} magic damage", eargs.attacker.Name, eargs.target.Name, eargs.attacker.magicDamage);  MeleeAttackEvent += new AttackHandler(MeleeAttackMessage);  MagicAttackEvent += MagicAttackMessage;  BuyEvent += buyHandler;  }  public void wear(Item item)  {  inventory.wear(item);  countDamage();  }  public void healthPotion()  {  if (healthPotions > 0)  health += 15;  if (health > maxHealth)  health = maxHealth;  }  public void magickaPotion()  {  if (magickaPotions > 0)  magicka += 15;  if (magicka > maxMagicka)  magicka = maxMagicka;  }  public void meleeAttack(Hero target)  {  AttackEventArgs eargs = new AttackEventArgs(this, target);  target.health -= (int)meleeDamage \* target.inventory.Armor.Stat;  if (MeleeAttackEvent != null)  MeleeAttackEvent(eargs);  }  public void magicAttack(Hero target)  {  AttackEventArgs eargs = new AttackEventArgs(this, target);  target.health -= (int)magicDamage \* target.inventory.Amulet.Stat;  if (MagicAttackEvent != null)  MagicAttackEvent(eargs);  }  public void buy(Item item)  {  bool x = BuyEvent(item);  if (x)  {  pool.Add(item);  this.coins -= item.Price;  }  }  }  public class Enemy : Hero, IDisposable  {  protected bool disposed = false;  public Enemy(string name, Race race)  : base(name, race)  {  Weapon weapon = new Weapon("Sword", 50, 2);  Armor armor = new Armor("Armor", 30, 1);  Ring ring = Ring.create("Habr la bra");  Amulet amulet = new Amulet("Amulet", 45, 0.95);  inventory = new Inventory(weapon, armor, ring, amulet);  countDamage();  }  public void Dispose()  {  Dispose(true);  GC.SuppressFinalize(this);  }  protected void Dispose(bool disposing)  {  if (disposed) return;  if (disposing)  {  Name = null;  inventory.Dispose();  }  disposed = true;  }  }  } |

|  |
| --- |
| Inventory.cs |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Collections.ObjectModel;  using System.Runtime.Serialization;  using System.Runtime.Serialization.Json;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Xml.Serialization;  namespace lab4  {  public class Inventory: IDisposable  {  protected Weapon weapon;  protected Armor armor;  protected Ring ring;  protected Amulet amulet;  bool disposed = false;  public Weapon Weapon  {  get { return weapon; }  set  {  if (value.Name != "empty")  weapon = value;  }  }  public Armor Armor  {  get { return armor; }  set  {  if (value.Name != "empty")  armor = value;  }  }  public Ring Ring  {  get { return ring; }  set  {  if (value.Name != "empty")  ring = value;  }  }  public Amulet Amulet  {  get { return amulet; }  set  {  if (value.Name != "empty")  amulet = value;  }  }  public Inventory()  {  weapon = new Weapon("empty", 0, 1);  armor = new Armor("empty", 0, 1);  ring = Ring.create("null");  amulet = new Amulet("empty", 0, 1);  }  //Функції: аргументи речей сховані в окремих класах  public Inventory(Weapon weapon, Armor armor, Ring ring, Amulet amulet)  {  this.weapon = weapon;  this.armor = armor;  this.ring = ring;  this.amulet = amulet;  }  public void wear(Item item)  {  if (item is Weapon) weapon = (Weapon)item;  if (item is Armor) armor = (Armor)item;  if (item is Ring) ring = (Ring)item;  if (item is Amulet) amulet = (Amulet)item;  } // перейменування методу  public void Dispose()  {  Dispose(true);  GC.SuppressFinalize(this);  }  protected void Dispose(bool disposing)  {  if (disposed) return;  if (disposing)  {  weapon.Dispose();  armor.Dispose();  ring.Dispose();  amulet.Dispose();  Console.WriteLine("Pool Disposed!!!");  }  disposed = true;  }  }  [DataContract]  [Serializable]  public abstract class Item : IComparable, IDisposable  {  protected string name; // рефакторинг - інкапсуляція  protected int price;  bool disposed = false;  public Item()  {  }  protected Item(string name, int price)  {  this.name = name;  if (price >= 0)  this.price = price;  else  {  Console.WriteLine("Error, unavailable price");  return;  }  }  [DataMember]  [XmlAttribute]  public int Price  {  get { return price; }  set { price = value; }  }  [DataMember]  [XmlAttribute]  public string Name  {  get { return name; }  set { name = value; }  }  public abstract double Stat { get; }  public int CompareTo(object obj)  {  if (obj == null) return 1;  Item otherItem = obj as Item;  if (otherItem != null)  return this.Name.CompareTo(otherItem.Name);  else  throw new ArgumentException("Object is not an Item");  }  public virtual void Dispose()  {  Dispose(true);    }  protected void Dispose(bool disposing)  {  if (disposed) return;  if (disposing)  {  name = null;  price = 0;  disposed = true;  }  }  }  [DataContract]  [Serializable]  public class Weapon : Item  {  [XmlIgnore]  protected int damage;  public Weapon()  : base()  {  }  public Weapon(string name, int price, int damage)  : base(name, price)  {  this.damage = damage;  }  [XmlIgnore]  public override double Stat  {  get { return this.damage; }  }  }  [DataContract]  [Serializable]  public class Armor : Item  {  [XmlIgnore]  protected double protection;  public Armor() : base() { }  public Armor(string name, int price, double protection)  : base(name, price)  {  this.protection = protection;  }  [XmlIgnore]  public override double Stat  {  get { return this.protection; }  }  }  public class Ring : Item  {  protected int magicAttack;  Ring() : base() { }  private Ring(string name, int price, int magicAttack)  : base(name, price)  {  this.magicAttack = magicAttack;  }  public static Ring create(string spell)  {  if (spell == "null")  return new Ring("empty", 0, 1);  if (spell == "Habr la bra")  return new Ring("Aprentice ring", 20, 2);  else Console.WriteLine("You don`t know a right spell, mortal\n");  return null;  }  [XmlIgnore]  public override double Stat  {  get { return magicAttack; }  }  }  public class Amulet : Item  {  protected double magicResist;  Amulet() : base() { }  public Amulet(string name, int price, double magicResist)  : base(name, price)  {  this.magicResist = magicResist;  }  public override double Stat  {  get { return magicResist; }  }  }  [DataContract]  [KnownType(typeof(Item[]))]  [KnownType(typeof(Weapon))]  [KnownType(typeof(Armor))]  [Serializable]  [XmlInclude(typeof(Item))]  [XmlInclude(typeof(Weapon))]  [XmlInclude(typeof(Armor))]  [XmlInclude(typeof(Amulet))]  [XmlInclude(typeof(Ring))]  public class Pool : IEnumerable, IEnumerator, IDisposable  {  [DataMember]  [XmlArrayItem("Item", Type = typeof(Item))]  public List<Item> items; //Імена - назва списку позначає його вміст  bool disposed = false;  [XmlIgnore]  int index = -1;  public Pool()  {  items = new List<Item>();  }  public void Add(Item elem)  {  items.Add((Item)elem);  }  public void Add(object elem)  {  items.Add((Item)elem);  }  private Item getItem(string name)  {  for (int i = 0; i < items.Count; i++)  {  if (items[i].Name == name)  {  return items[i];  }  }  throw new System.ArgumentOutOfRangeException(name, "No such Item in your pool");  }  public Item this[string name]  {  get { return getItem(name); }  set { items.Add(value); }  }  public Item this[int ind]  {  get { return (Item) items[ind]; }  set { items[ind] = value; }  }  public IEnumerator GetEnumerator() { return (IEnumerator)this; }  public bool MoveNext()  {  if (index < items.Capacity - 1)  {  index++;  return true;  }  else  return false;  }  public void Reset()  {  index = -1;  }  public object Current  {  get { return items[index]; }  }  public void Dispose()  {  Dispose(true);  GC.SuppressFinalize(this);  }  protected void Dispose(bool disposing)  {  if (disposed) return;  if (disposing)  {  Console.WriteLine("Pool disposed");  items.Clear();  items = null;  }  disposed = true;  }  ~Pool()  {  Dispose();  }  }  public static class PoolExtensions //Імена: назва класу передає наміри програміста  {  public static bool Any(this Pool pool, Item item) //рефакторинг - видалення зайвої локальної змінної  {  for (int i = 0; i < pool.items.Capacity; i++)  {  if (pool.items[i].Name == item.Name)  {  return true;  }  }  return false;  }  }  } |

|  |
| --- |
| Program.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.IO;  using System.Text;  using System.Runtime.Serialization;  using System.Runtime.Serialization.Json;  using System.Threading.Tasks;  using System.Xml.Serialization;  using System.Runtime.Serialization.Formatters.Binary;  namespace lab4  {    class Program  {  public static void JsonSerial(Player player)  {  using (MemoryStream stream = new MemoryStream())  {  Pool pool = player.pool;  DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(Pool), new[] { typeof(Item) });  ser.WriteObject(stream, pool);  stream.Position = 0;  StreamReader streamReader = new StreamReader(stream);  Console.WriteLine(streamReader.ReadToEnd());  stream.Position = 0;  Pool result = (Pool)ser.ReadObject(stream);  }  } //рефакторинг - зміна сигнатури методу  public static void XmlSerial(Player player)  {  Pool pool = player.pool;  XmlSerializer serializer = new XmlSerializer(typeof(Pool));  string xml;  using (StringWriter stringWriter = new StringWriter())  {  serializer.Serialize(stringWriter, pool);  xml = stringWriter.ToString();  Console.WriteLine(xml);  }  using (StringReader stringReader = new StringReader(xml))  {  Pool newPool = (Pool)serializer.Deserialize(stringReader);  }  }    static void Main(string[] args)  {  Player player1 = new Player("Player\_name", Race.HUMAN);  WeakReference weak = new WeakReference(player1);  player1 = null;  Console.WriteLine((Player)weak.Target);  Console.WriteLine(weak.IsAlive);  GC.Collect(0, GCCollectionMode.Forced);  Console.WriteLine(weak.IsAlive);  Pool pool = new Pool();  Weapon weap1 = new Weapon("Item", 42, 5);  Weapon weap2 = new Weapon("Item", 42, 5);  Weapon weap3 = new Weapon("Item", 42, 5);  Weapon weap4 = new Weapon("Item", 42, 5);  Weapon weap5 = new Weapon("Item", 42, 5);  Weapon weap6 = new Weapon("Item", 42, 5);  pool.Add(weap1);  pool.Add(weap2);  pool.Add(weap3);  pool.Add(weap4);  pool.Add(weap5);  pool.Add(weap6);  pool = null;  GC.Collect();      Console.ReadKey();  }  }  } |

**Діаграма класів**

****

|  |
| --- |
| **Ім’я функції пояснює сенс функції.** |
| void meleeAttack(T target); // Функції: змістовні імена  void magicAttack(T target); |

|  |
| --- |
| **Виключення замість повернення коду помилки** |
| switch (race)  {  case Race.HUMAN:  this.maxHealth = 35;  this.maxMagicka = 30;  this.BaseMeleeDamage = 6;  this.BaseMagicDamage = 5;  break;  case Race.ORC:  this.maxHealth = 35;  this.maxMagicka = 25;  this.BaseMeleeDamage = 7;  this.BaseMagicDamage = 4;  break;  case Race.ELFE:  this.maxHealth = 30;  this.maxMagicka = 35;  this.BaseMeleeDamage = 4;  this.BaseMagicDamage = 7;  break;  default: // Функції: виключення замість коду помилки  throw new WrongDataException("Wrong race, can`t create Hero", race.ToString());  } |

|  |
| --- |
| **Функціям потрібно якнайменше аргументів. Тут аргументи сховані в речах (зброї, броні, перстні та амулеті)** |
| //Функції: аргументи речей сховані в окремих класах  public Inventory(Weapon weapon, Armor armor, Ring ring, Amulet amulet)  {  this.weapon = weapon;  this.armor = armor;  this.ring = ring;  this.amulet = amulet;  } |

|  |
| --- |
| **Змістовні імена полів** |
| public List<Item> items; //Імена - назва списку позначає його вміст |

|  |
| --- |
| **Імена передають наміри програміста** |
| public static class PoolExtensions //Імена: назва класу передає наміри програміста  {  public static bool Any(this Pool pool, Item item)  {  for (int i = 0; i < pool.items.Capacity; i++)  {  if (pool.items[i].Name == item.Name)  {  return true;  }  }  return false;  }  } |

**Застосовані правила рефакторингу**

1. Під час рефакторингу було застосовано метод зміни сигнатури деяких методів.
2. Було змінено деякі параметри методу і зменшено загальну кількість параметрів.
3. Деякі фрагменти коду були виділені в окремі методи – один із методів рефакторингу – виділення методів. Було застосовано до фрагментів коду, які потребували певних коментарів про те, що вони роблять.
4. Упакування деяких аргументів функції в окремий клас – зменшення кількості аргументів шляхом створення об’єктів.

**Висновки**

Виконавши дану лабораторну роботу, я ознайомився із правилами написання чистого коду та вивчив механізми управління ресурсами, що реалізовані у платформі .Net.