|  |  |
| --- | --- |
| **SOFTWARE DESIGN PATTERNS**  Diploma in IT  Year 2 (2024/25) Semester 4 | Week **5** |
| **2** hours |
| **Exercise 5 – Decorator** | |

**OBJECTIVES**

* Understand and implement the Decorator design pattern

**ACTIVITY**

A video game has randomly generated weapons. Each weapon deals a fixed base damage. It can have multiple prefixes and one suffix that add to the base damage.

Weapons:

|  |  |
| --- | --- |
| **Weapon name** | **Damage** |
| Dagger | 4 |
| Sword | 6 |
| Mace | 5 |

Prefixes:

|  |  |
| --- | --- |
| **Prefix name** | **Damage increase** |
| Burning | +3 |
| Icy | +3 |
| Blessed | +3 |

Suffixes:

|  |  |
| --- | --- |
| **Suffix name** | **Damage increase** |
| Pain | +2 |
| Justice | +4 |
| Vengeance | +6 |

* Draw the class diagram for the above scenario using the Decorator design pattern.

A diagram of a weapon

Description automatically generated

* Implement the Decorator design pattern such that running the following program will produce the desired output.

Program:

Weapon w1 = new Dagger();

Console.WriteLine($"{w1.getDescription()} deals {w1.getDamage()} damage.");

w1 = new Burning(w1);

Console.WriteLine($"{w1.getDescription()} deals {w1.getDamage()} damage.");

w1 = new Pain(w1);

Console.WriteLine($"{w1.getDescription()} deals {w1.getDamage()} damage.");

w1 = new Icy(w1);

Console.WriteLine($"{w1.getDescription()} deals {w1.getDamage()} damage.");

Output:

Dagger deals 4 damage.

Burning Dagger deals 7 damage.

Burning Dagger of Pain deals 9 damage.

Icy Burning Dagger of Pain deals 12 damage.

**Discussion Notes:**

* What if you want to have multiple suffixes and have the description be grammatically correct, such as “Burning Sword of Pain, Justice and Vengeance”? One solution is to have another decorator, e.g., NiceNameDecorator, whose job it is to generate the name correctly. In this case, it may be more convenient for getDescription() to return a list of strings to make generating the correct name easier.
* What if Burning causes fire damage, Icy causes ice damage, etc, and this has an in-game effect? Then getDamage() cannot just return an integer. Instead, it may be best to create a Damage object that contains all of the damage information such as damage dealt, damage type, elemental type or other effects. But as long as we can create an updated Damage object each time we add a decorator, we can use the decorator pattern.
* Creating objects using the decorator pattern can seem troublesome, but we can use the Factory or Builder pattern to help us create objects in a better way.