**UML class diagram**

[**#**](https://www.dofactory.com/net/flyweight-design-pattern#uml)

A visualization of the classes and objects participating in this pattern.

Diagrama

Descrição gerada automaticamente

**Participants**

[**#**](https://www.dofactory.com/net/flyweight-design-pattern#participants)

The classes and objects participating in this pattern include:

* Flyweight  (Character)
  + declares an interface through which flyweights can receive and act on extrinsic state.
* ConcreteFlyweight   (CharacterA, CharacterB, ..., CharacterZ)
  + implements the Flyweight interface and adds storage for intrinsic state, if any. A ConcreteFlyweight object must be sharable. Any state it stores must be intrinsic, that is, it must be independent of the ConcreteFlyweight object's context.
* UnsharedConcreteFlyweight   ( not used )
  + not all Flyweight subclasses need to be shared. The Flyweight interface enables sharing, but it doesn't enforce it. It is common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the flyweight object structure (as the Row and Column classes have).
* FlyweightFactory   (CharacterFactory)
  + creates and manages flyweight objects
  + ensures that flyweight are shared properly. When a client requests a flyweight, the FlyweightFactory objects assets an existing instance or creates one, if none exists.
* Client   (FlyweightApp)
  + maintains a reference to flyweight(s).
  + computes or stores the extrinsic state of flyweight(s).

**Structural code in C#**

[**#**](https://www.dofactory.com/net/flyweight-design-pattern#structural)

This structural code demonstrates the Flyweight pattern in which a relatively small number of objects is shared many times by different clients.

[[](javascript:copy('flyweight-structural');) copy](javascript:copy('flyweight-structural');)

1. **using System;**
2. **using System.Collections.Generic;**
3. **namespace Flyweight.Structural**
4. **{**
5. **/// <summary>**
6. **/// Flyweight Design Pattern**
7. **/// </summary>**
8. **public class Program**
9. **{**
10. **public static void Main(string[] args)**
11. **{**
12. **// Arbitrary extrinsic state**
13. **int extrinsicstate = 22;**
14. **FlyweightFactory factory = new FlyweightFactory();**
15. **// Work with different flyweight instances**
16. **Flyweight fx = factory.GetFlyweight("X");**
17. **fx.Operation(--extrinsicstate);**
18. **Flyweight fy = factory.GetFlyweight("Y");**
19. **fy.Operation(--extrinsicstate);**
20. **Flyweight fz = factory.GetFlyweight("Z");**
21. **fz.Operation(--extrinsicstate);**
22. **UnsharedConcreteFlyweight fu = new**
23. **UnsharedConcreteFlyweight();**
24. **fu.Operation(--extrinsicstate);**
25. **// Wait for user**
26. **Console.ReadKey();**
27. **}**
28. **}**
29. **/// <summary>**
30. **/// The 'FlyweightFactory' class**
31. **/// </summary>**
32. **public class FlyweightFactory**
33. **{**
34. **private Dictionary<string, Flyweight> flyweights { get; set; } = new Dictionary<string, Flyweight>();**
35. **// Constructor**
36. **public FlyweightFactory()**
37. **{**
38. **flyweights.Add("X", new ConcreteFlyweight());**
39. **flyweights.Add("Y", new ConcreteFlyweight());**
40. **flyweights.Add("Z", new ConcreteFlyweight());**
41. **}**
42. **public Flyweight GetFlyweight(string key)**
43. **{**
44. **return ((Flyweight)flyweights[key]);**
45. **}**
46. **}**
47. **/// <summary>**
48. **/// The 'Flyweight' abstract class**
49. **/// </summary>**
50. **public abstract class Flyweight**
51. **{**
52. **public abstract void Operation(int extrinsicstate);**
53. **}**
54. **/// <summary>**
55. **/// The 'ConcreteFlyweight' class**
56. **/// </summary>**
57. **public class ConcreteFlyweight : Flyweight**
58. **{**
59. **public override void Operation(int extrinsicstate)**
60. **{**
61. **Console.WriteLine("ConcreteFlyweight: " + extrinsicstate);**
62. **}**
63. **}**
64. **/// <summary>**
65. **/// The 'UnsharedConcreteFlyweight' class**
66. **/// </summary>**
67. **public class UnsharedConcreteFlyweight : Flyweight**
68. **{**
69. **public override void Operation(int extrinsicstate)**
70. **{**
71. **Console.WriteLine("UnsharedConcreteFlyweight: " +**
72. **extrinsicstate);**
73. **}**
74. **}**
75. **}**

**Output**

ConcreteFlyweight: 21  
ConcreteFlyweight: 20  
ConcreteFlyweight: 19  
UnsharedConcreteFlyweight: 18

**Real-world code in C#**

[**#**](https://www.dofactory.com/net/flyweight-design-pattern#realworld)

This real-world code demonstrates the Flyweight pattern in which a relatively small number of Character objects is shared many times by a document that has potentially many characters.

[[](javascript:copy('flyweight-realworld');) copy](javascript:copy('flyweight-realworld');)

1. **using System;**
2. **using System.Collections.Generic;**
3. **namespace Flyweight.RealWorld**
4. **{**
5. **/// <summary>**
6. **/// Flyweight Design Pattern**
7. **/// </summary>**
8. **public class Program**
9. **{**
10. **public static void Main(string[] args)**
11. **{**
12. **// Build a document with text**
13. **string document = "AAZZBBZB";**
14. **char[] chars = document.ToCharArray();**
15. **CharacterFactory factory = new CharacterFactory();**
16. **// extrinsic state**
17. **int pointSize = 10;**
18. **// For each character use a flyweight object**
19. **foreach (char c in chars)**
20. **{**
21. **pointSize++;**
22. **Character character = factory.GetCharacter(c);**
23. **character.Display(pointSize);**
24. **}**
25. **// Wait for user**
26. **Console.ReadKey();**
27. **}**
28. **}**
29. **/// <summary>**
30. **/// The 'FlyweightFactory' class**
31. **/// </summary>**
32. **public class CharacterFactory**
33. **{**
34. **private Dictionary<char, Character> characters = new Dictionary<char, Character>();**
35. **public Character GetCharacter(char key)**
36. **{**
37. **// Uses "lazy initialization"**
38. **Character character = null;**
39. **if (characters.ContainsKey(key))**
40. **{**
41. **character = characters[key];**
42. **}**
43. **else**
44. **{**
45. **switch (key)**
46. **{**
47. **case 'A': character = new CharacterA(); break;**
48. **case 'B': character = new CharacterB(); break;**
49. **//...**
50. **case 'Z': character = new CharacterZ(); break;**
51. **}**
52. **characters.Add(key, character);**
53. **}**
54. **return character;**
55. **}**
56. **}**
57. **/// <summary>**
58. **/// The 'Flyweight' abstract class**
59. **/// </summary>**
60. **public abstract class Character**
61. **{**
62. **protected char symbol;**
63. **protected int width;**
64. **protected int height;**
65. **protected int ascent;**
66. **protected int descent;**
67. **protected int pointSize;**
68. **public abstract void Display(int pointSize);**
69. **}**
70. **/// <summary>**
71. **/// A 'ConcreteFlyweight' class**
72. **/// </summary>**
73. **public class CharacterA : Character**
74. **{**
75. **// Constructor**
76. **public CharacterA()**
77. **{**
78. **symbol = 'A';**
79. **height = 100;**
80. **width = 120;**
81. **ascent = 70;**
82. **descent = 0;**
83. **}**
84. **public override void Display(int pointSize)**
85. **{**
86. **this.pointSize = pointSize;**
87. **Console.WriteLine(symbol +**
88. **" (pointsize " + this.pointSize + ")");**
89. **}**
90. **}**
91. **/// <summary>**
92. **/// A 'ConcreteFlyweight' class**
93. **/// </summary>**
94. **public class CharacterB : Character**
95. **{**
96. **// Constructor**
97. **public CharacterB()**
98. **{**
99. **symbol = 'B';**
100. **height = 100;**
101. **width = 140;**
102. **ascent = 72;**
103. **descent = 0;**
104. **}**
105. **public override void Display(int pointSize)**
106. **{**
107. **this.pointSize = pointSize;**
108. **Console.WriteLine(this.symbol +**
109. **" (pointsize " + this.pointSize + ")");**
110. **}**
111. **}**
112. **// ... C, D, E, etc.**
113. **/// <summary>**
114. **/// A 'ConcreteFlyweight' class**
115. **/// </summary>**
116. **public class CharacterZ : Character**
117. **{**
118. **// Constructor**
119. **public CharacterZ()**
120. **{**
121. **symbol = 'Z';**
122. **height = 100;**
123. **width = 100;**
124. **ascent = 68;**
125. **descent = 0;**
126. **}**
127. **public override void Display(int pointSize)**
128. **{**
129. **this.pointSize = pointSize;**
130. **Console.WriteLine(this.symbol +**
131. **" (pointsize " + this.pointSize + ")");**
132. **}**
133. **}**
134. **}**

**Output**

A (pointsize 11)  
A (pointsize 12)  
Z (pointsize 13)  
Z (pointsize 14)  
B (pointsize 15)  
B (pointsize 16)  
Z (pointsize 17)  
B (pointsize 18)