

Week 1: Design Patterns and Principles

Exercise 1: Implementing the Singleton Pattern

-THEJASHRI NARAYANAN

Scenario:

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

Steps:

1. Created a New Java Project:

- Project name: SingletonPatternExample

2. Defined the Singleton Class:

- Created a class named Logger
- Made the constructor private
- Used a private static instance variable
- Provided a public static method to return the single instance

Logger.java

```
1  public class Logger {
2      private static Logger loggerInstance;
3
4      private Logger() {
5          System.out.println("Logger initialized...");
6      }
7
8      public static Logger getInstance() {
9          if (loggerInstance == null) {
10             loggerInstance = new Logger();
11          }
12          return loggerInstance;
13     }
14
15     public void log(String message) {
16         System.out.println("[LOG]: " + message);
17     }
18 }
19
```

3. Implemented Singleton Logic:

- Used lazy initialization in getInstance() to create only one instance
- Verified using object references

4. Tested Singleton Implementation:

- Wrote a Main class to test and verify that both variables point to the same Logger instance

Main.java

```
J Main.java X
J Main.java
1 public class Main {
    Run | Debug | Run main | Debug main
2     public static void main(String[] args) {
3         Logger log1 = Logger.getInstance();
4         log1.log(message:"First log message");
5
6         Logger log2 = Logger.getInstance();
7         log2.log(message:"Second log message");
8
9         if (log1 == log2) {
10             System.out.println(x:"Both loggers are the same instance (Singleton Verified)");
11         } else {
12             System.out.println(x:"Different instances (Singleton Failed)");
13         }
14     }
15 }
16
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\design nd patterns\pgm> javac Logger.java Main.java
>>
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\design nd patterns\pgm> java Main
Logger initialized...
[LOG]: First log message
[LOG]: Second log message
Both loggers are the same instance (Singleton Verified )
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\design nd patterns\pgm> █
```

Exercise 2: Implementing the Factory Method Pattern

Scenario:

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

Steps :

```
J CustomFileManager.java > Language Support for Java(TM) by Red Hat > CustomFileManager
1  import java.util.Scanner;
2
3  // STEP 1: Define the interface
4
5  interface FileHandler {
6      void open();
7      void save();
8  }
9
10 // STEP 2: Create concrete implementations
11
12 class TextFile implements FileHandler {
13     @Override
14     public void open() {
15         System.out.println(x:"Opening a plain text file...");
16     }
17
18     @Override
19     public void save() {
20         System.out.println(x:"Saving changes to the text file.");
21     }
22 }
23
24 class CsvFile implements FileHandler {
25     @Override
26     public void open() {
27         System.out.println(x:"Opening a CSV spreadsheet...");
28     }
29
30     @Override
31     public void save() {
32         System.out.println(x:"Saving CSV data...");
33     }
34 }
```

```

33     }
34 }
35 class MarkdownFile implements FileHandler {
36     @Override
37     public void open() {
38         System.out.println(x:"Opening a Markdown document...");
39     }
40
41     @Override
42     public void save() {
43         System.out.println(x:"Saving Markdown content...");
44     }
45 }
46
47 // STEP 3: Create the abstract factory
48
49 abstract class FileFactory {
50     public abstract FileHandler generateFile();
51 }
52
53 // STEP 4: Implement concrete factories
54
55 class TextFileFactory extends FileFactory {
56     @Override
57     public FileHandler generateFile() {
58         return new TextFile();
59     }
60 }
61
62 class CsvFileFactory extends FileFactory {
63     @Override

```

```

62 class CsvFileFactory extends FileFactory {
63     @Override
64     public FileHandler generateFile() {
65         return new CsvFile();
66     }
67 }
68
69 class MarkdownFileFactory extends FileFactory {
70     @Override
71     public FileHandler generateFile() {
72         return new MarkdownFile();
73     }
74 }
75
76
77 // STEP 5: Main class to test everything
78
79 public class CustomFileManager {
80     Run main | Debug main | Run | Debug
81     public static void main(String[] args) {
82
83         try (Scanner scanner = new Scanner(System.in)) {
84             System.out.println(x:"Choose file type to create: text / csv / markdown");
85             String choice = scanner.nextLine().toLowerCase();
86
87             FileFactory factory;
88
89             switch (choice) {
90                 case "text":
91                     factory = new TextFileFactory();
92                     break;

```

```

91         break;
92     case "csv":
93         factory = new CsvFileFactory();
94         break;
95     case "markdown":
96         factory = new MarkdownFileFactory();
97         break;
98     default:
99         System.out.println(x:"Invalid choice.");
100        return;
101    }
102    FileHandler file = factory.generateFile();
103    file.open();
104    file.save();
105    }
106    }
107    }
108

```

Output

```

PS C:\Usejavac CustomFileManager.javas\Engineering concepts\design nd p
>> erns>
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\design nd patterns> java CustomFileManager
>>
Choose file type to create: text / csv / markdown
text
Opening a plain text file...
Saving changes to the text file.
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\design nd patterns> java CustomFileManager
>>
Choose file type to create: text / csv / markdown
csv
Opening a CSV spreadsheet...
csv
Opening a CSV spreadsheet...
Saving CSV data...
csv
Opening a CSV spreadsheet...
Opening a CSV spreadsheet...
Saving CSV data...
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\design nd patterns> java CustomFileManager
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\design nd patterns> java CustomFileManager
>>
Choose file type to create: text / csv / markdown
markdown
Opening a Markdown document...
Opening a Markdown document...
Saving Markdown content...

```

Algorithms and Data Structures

Exercise: 2 E-commerce Platform Search Function

Scenario:

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

Steps:

1. Understanding Big O Notation

Big O notation is just a way to measure how fast or slow an algorithm is when the input size gets really big. It helps us know which algorithm is better when we have lots of data.

- Best case: The search finds the item right away (super fast).
- Average case: On average, how long it takes to find the item.
- Worst case: The longest it could take, like if the item isn't there or at the very end.

2. Program:

```
Program.cs X
ProductSearchApp > Program.cs > Program > Main
1  using System;
2
3  namespace ProductSearchApp
4  {
5      0 references
6      class Program
7      {
8          // STEP 2: Define Product class
9          12 references
10         public class Product
11         {
12             2 references
13             public int Id { get; }
14             4 references
15             public string Name { get; }
16             2 references
17             public string Type { get; }
18
19             5 references
20             public Product(int id, string name, string type)
21             {
22                 Id = id;
23                 Name = name;
24                 Type = type;
25             }
26
27             2 references
28             public override string ToString()
29             {
30                 return $"Product ID: {Id}, Name: {Name}, Type: {Type}";
31             }
32         }
33     }
34 }
```

```

26
27 // STEP 3a: Implement Linear Search
28 1 reference
29 public static Product FindUsingLinearSearch(Product[] items, string keyword)
30 {
31     foreach (var item in items)
32     {
33         if (string.Equals(item.Name, keyword, StringComparison.OrdinalIgnoreCase))
34         {
35             return item;
36         }
37     }
38     return null;
39 }
40
41 // STEP 3b: Implement Binary Search
42 1 reference
43 public static Product FindUsingBinarySearch(Product[] items, string keyword)
44 {
45     int low = 0;
46     int high = items.Length - 1;
47
48     while (low <= high)
49     {
50         int mid = (low + high) / 2;
51         int compare = string.Compare(items[mid].Name, keyword, StringComparison.OrdinalIgnoreCase);
52
53         if (compare == 0)
54             return items[mid];
55         else if (compare < 0)
56             low = mid + 1;
57         else
58             high = mid - 1;
59     }
60 }

```

```

61
62 return null;
63 }
64
65 // STEP 5: Main method to test both searches
66 0 references
67 static void Main(string[] args)
68 {
69     Product[] catalog = new Product[]
70     {
71         new Product(1, "Laptop", "Electronics"),
72         new Product(2, "Keyboard", "Accessories"),
73         new Product(3, "Shoes", "Fashion"),
74         new Product(4, "Book", "Education"),
75         new Product(5, "Mobile", "Electronics")
76     };
77
78     Console.WriteLine("Enter product name to search: ");
79     string inputName = Console.ReadLine();
80
81     // Linear search (no need to sort)
82     Console.WriteLine("\n🔍 Linear Search Result:");
83     var result1 = FindUsingLinearSearch(catalog, inputName);
84     Console.WriteLine(result1 != null ? result1.ToString() : "Product not found");
85
86     // Binary search (sort first)
87     Array.Sort(catalog, (p1, p2) => string.Compare(p1.Name, p2.Name, StringComparison.OrdinalIgnoreCase));
88     Console.WriteLine("\n🔍 Binary Search Result:");
89     var result2 = FindUsingBinarySearch(catalog, inputName);
90     Console.WriteLine(result2 != null ? result2.ToString() : "Product not found");
91 }
92 }
93

```

Output:

```
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS> cd ProductSearchApp
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS\ProductSearchApp> dotnet run
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS\ProductSearchApp> dotnet run
Enter product name to search: book

📦 Linear Search Result:
Enter product name to search: book
○
📦 Linear Search Result:
📦 Linear Search Result:
Product ID: 4, Name: Book, Type: Education

📦 Binary Search Result:
Product ID: 4, Name: Book, Type: Education
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS\ProductSearchApp> █
```

4. Analysis :

In my project, I compared **linear search** and **binary search** to find products faster.

What I Observed:

- **Linear Search** checks each product one by one. So if I have 1000 products, it may need to check all 1000 in the worst case. That's why it's **slow for big data** – time complexity is **$O(n)$** .
- **Binary Search** first sorts the product list, then keeps dividing the list to find the target quickly.
So it's way **faster**, especially when the list is large – time complexity is **$O(\log n)$** .

Which Is Better?

- For **small lists**, linear search is fine and easy to use.
- But for **big platforms like Amazon**, **binary search is much better** because it saves time.

Exercise 7: Financial Forecasting

Scenario:

You are developing a financial forecasting tool that predicts future values based on past data.

Steps:

1. Understand Recursive Algorithm

Recursion is when a method calls itself to solve a smaller version of the same problem.

It's ideal for problems with repetitive structure, like forecasting over months/years.

2&3. Recursive Forecasting Method:

```
FinancePredictor > Program.cs > ...
1  using System;
2  namespace FinanceForecast
3  {
4      0 references
      class Program
5      {
6          // STEP 1 & 2: Recursive method to estimate future investment value
          2 references
          public static double EstimateGrowth(double principal, double growthRate, int periods)
7          {
8              if (periods == 0)
9              {
10                 return principal;
11             }
12             double updatedAmount = principal * (1 + growthRate);
13             return EstimateGrowth(updatedAmount, growthRate, periods - 1);
14         }
15     }
16
17     // STEP 3: Main method to test the forecast
18     0 references
19     static void Main(string[] args)
20     {
21         Console.WriteLine("=== Financial Growth Forecast Tool ===");
22
23         Console.Write("Enter initial amount (₹): ");
24         if (!double.TryParse(Console.ReadLine(), out double initialAmount))
25         {
26             Console.WriteLine("Invalid input for initial amount.");
27             return;
28         }
29
30         Console.Write("Enter annual growth rate (e.g., 0.07 for 7%): ");
31         if (!double.TryParse(Console.ReadLine(), out double growthRate))
32         {
33             Console.WriteLine("Invalid input for growth rate.");
34             return;
35         }
36     }
37 }
```

```

35     }
36
37     Console.Write("Enter number of years: ");
38     if (!int.TryParse(Console.ReadLine(), out int years))
39     {
40         Console.WriteLine("Invalid input for number of years.");
41         return;
42     }
43
44     // STEP 4: Forecast using recursion
45     double estimatedValue = EstimateGrowth(initialAmount, growthRate, years);
46     Console.WriteLine($"Estimated value after {years} years: ₹{estimatedValue:F2}");
47 }
48 }
49 }
50

```

Output:

```

PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS> dotnet new console -o FinancePredictor
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS\FinancePredictor\FinancePredictor.csproj:
Restore succeeded.

PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS> cd FinancePredictor
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS\FinancePredictor> dotnet build
Restore complete (1.0s)
  FinancePredictor succeeded (4.6s) -> bin\Debug\net9.0\FinancePredictor.dll

Build succeeded in 7.1s
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS\FinancePredictor> dotnet run
=== Financial Growth Forecast Tool ===
Enter initial amount (₹): 60000
Enter annual growth rate (e.g., 0.07 for 7%): 7
Enter number of years: 8
Estimated value after 8 years: ₹100663296000.00
PS C:\Users\Hp\Desktop\cts assessments\Engineering concepts\ADS\FinancePredictor>

```

4. Analysis :

Time Complexity:

Recursive depth = years (n) → So it's $O(n)$ time

No repeated work = No exponential growth = Still efficient

Optimization Techniques (if needed for complex models):

Memoization (store already computed results)

Tail Recursion (optimize for deep stacks)

Iterative version (preferred for large-scale datasets)

