



National Teachers College College  
Department  
J Nepomuceno, Quiapo, Manila

## **TITLE OF THE ACTIVITY**

Prelims Activity #2 in IT Elective 2 - Advanced Object-Oriented Programming and Robustness

**Submitted by:** James  
Lemit Onia

**To be submitted to:** Ms.  
Justin Louise Neypes

**Date:** February  
21, 2026

CHOSEN SCENARIO:

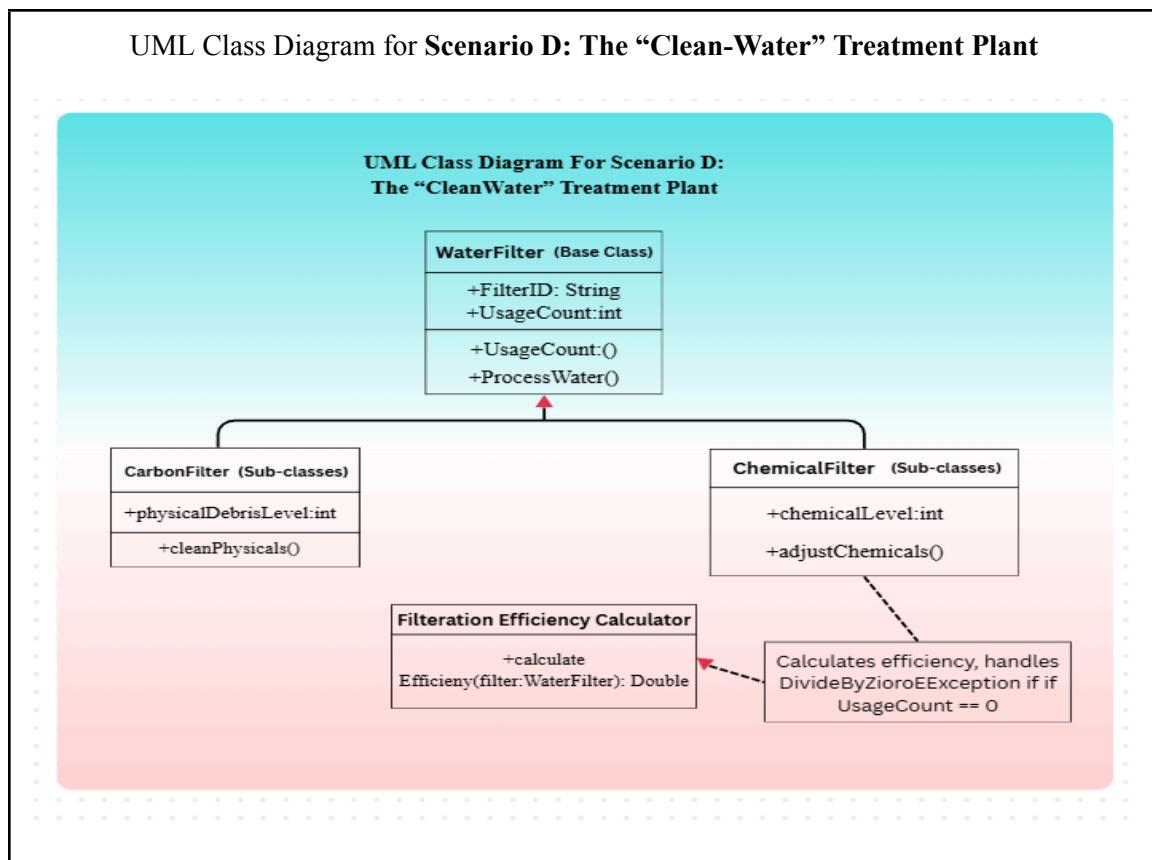
### Scenario D: The "Clean-Water" Treatment Plant

- **Base Class:** WaterFilter (Properties: FilterID, UsageCount).
- **Sub-Classes:** CarbonFilter and ChemicalFilter.
- **Logic:** An abstract method ProcessWater(). The ChemicalFilter must track chemical levels, while the Carbon tracks physical debris.
- **Exception:** Handle a DivideByZeroException when calculating the "Filtration Efficiency" if no water has passed through yet.

## Step-by-Step Instructions

### Phase 1: The Design Blueprint

1. **UML Class Diagram:** Create a visual diagram showing your Base Class, Sub-Classes, and any Interfaces. Identify which members are **private**, **protected**, or **public**.



So here are the explanation of the **UML Diagram** in Scenario D: The “Clean Water” Treatment Plant

- **WaterFilter (Base Class):** This is the "parent" class. It contains the common identity and usage tracking for all filters. The `ProcessWater()` method is abstract, meaning it provides a template that the sub-classes must fill in with their own logic.
- **Carbon & Chemical Filters (Sub-classes):** These are the "children." They inherit everything from `WaterFilter` but add their own specific responsibilities—Carbon handles physical debris, while Chemical handles chemical levels.
- **Filtration Efficiency Calculator:** This represents a utility or service class. It contains the logic to calculate performance and is specifically designed to handle the error that occurs if you try to divide by a `UsageCount` of zero.

2. **Logic Logic Flow:** Write a step-by-step description of how the pillars of OOP **are used** in your scenario (e.g., how the `CalculateRange` behaves differently for a bus vs. a van).

So here are the step-by-step description of how pillars of **OOP** are used in my chosen scenario which is the **Scenario D: The “Clean-Water” Treatment Plant**

1. **Encapsulation:** So let's start with **encapsulation**, so I used **encapsulation** in the system to protect sensitive data by setting the `chemicalLevel` to **Private (-)**. This prevents external classes from tampering with chemical balances directly. Access is only granted through controlled methods like `adjustChemicals()`, ensuring the internal state of the filter remains valid and safe.
2. **Inheritance:** So next is the **Inheritance**, So I used **Inheritance** to avoid redundant code, `CarbonFilter` and `ChemicalFilter` derive from the `WaterFilter` base class. This structure ensures that both specialized filters automatically possess the `FilterID` and `UsageCount` properties. It creates a "is-a" relationship where every specific filter is fundamentally a `WaterFilter`.
3. **Polymorphism:** So next is **Polymorphism**, So I used **Polymorphism** because Although both filters use the same method name, `ProcessWater()`, they behave differently based on their type. Because the first type is The `CarbonFilter` executes logic to trap **physical debris**. And next is the `ChemicalFilter` executes logic to neutralize **chemical levels**. This allows the plant to trigger the same command across a diverse list of filters while achieving different results.
4. **Abstraction:** And last but not the least is the **Abstraction**, So I used **Abstraction** in The system uses the `WaterFilter` **Abstract Class** to define a high-level "contract." It hides the complex internal mechanics of water purification and only exposes the essential method: `ProcessWater()`. This allows the plant to interact with any filter without needing to understand the specific science behind it.

3. **Exception Mapping:** List at least two specific scenarios in your code where a **try-catch** block is necessary to prevent a crash.

So here are the list of specific scenarios in my code where **try-catch** block is necessary to prevent a crash:

- So this is the **First Scenario**, The “**New Filter**” Calculation Error. The `CalculateEfficiency()` is inside of method of `WaterFilter` class:
- So in this Scenario, Without the try-catch in my Main method, the program would terminate instantly when the method is called. To catch it, the program stays open and tells the operator why it failed, and moves to the `finally` block safely.
- So here is the Code Logic used in this Scenario:

```
if (UsageCount == 0)
{
    throw new DivideByZeroException("Cannot calculate efficiency: No water has passed through this filter yet.");
}
```

- So this is the **Next Scenario** which is The “**Negative Usage**” Data Error. The Code Logic that is used in this scenario is the set because Inside the set accessor for the UsageCount properly.
- So how it prevents a crash in this scenario, When this `ArgumentException` is thrown, the catch (Exception ex) blocks the Main method and intercept it. This prevents the “invalid state” from breaking the rest of the application’s logic by allowing the system to log the error and shutdown gracefully.
- So here is the Code Logic used in this Scenario:

```
set
{
    if (value < 0) throw new ArgumentException("Usage count cannot be negative.");
    _usageCount = value;
}
```

## Phase 2: The Implementation

1. **Project Setup:** Create a new C# Console Application. Use the naming convention: `Surname_ScenarioLabel.sln`.

```
using System;

namespace Onia_ScenarioD
{
```

2. **Encapsulation:** Ensure all class fields are `private`. Use Public Properties with logic inside the `set` accessor (e.g., `if (value < 0) throw new ArgumentException();`).

**Encapsulation:** All fields (like `_filterId` and `_usageCount`) are `private`. The public properties use logic to prevent invalid data (like negative usage counts).

3. **Inheritance:** Use the `: base()` constructor syntax to pass data from sub-classes to the parent class.

**Inheritance:** Both subclasses use the `: base(id, usage)` syntax to pass data up to the `WaterFilter` constructor, ensuring a centralized way to initialize IDs.

4. **Polymorphism:** Use the `virtual` keyword in the base class and the `override` keyword in sub-classes.

**Polymorphism:** The `ProcessWater()` method is `abstract` in the parent and `overridden` in the children, allowing each filter to execute its specific cleaning logic.

5. **Robustness:** Wrap your "Main" execution logic in a `try-catch-finally` block. The `finally` block should print "System Shutdown" or "Session Ended."

**Robustness:** The `Main` method is wrapped in a `try-catch-finally` block. This captures the `DivideByZeroException` when a filter's efficiency is calculated before it has been used, and the `finally` block ensures the shutdown message always appears.

## Knowledge Check Questions

1. **Encapsulation:** Why is it dangerous to make the `CurrentTemp` or `BatteryPercentage` field `public`? How does a Property solve this?

It is dangerous because making fields like `UsageCount` or `chemicalLevel` public is dangerous because it allows external code to bypass the "rules" of the object. For example, any part of the program could set a filter's usage to a negative number or a chemical level to an unsafe value without the object knowing. In addition, this **Property** solves this by acting as a "gatekeeper." Using the `set` accessor, we can include validation logic (e.g., `if (value < 0)`). This ensures the internal state of the object remains valid and protected, forcing the rest of the application to interact with the data only on the object's terms.

2. **Inheritance:** What is the benefit of using a Base Class instead of creating two entirely separate classes from scratch?

The primary benefit of using a Base Class instead of creating two entirely separate classes from scratch are **Code Reusability** and **Consistency** because without a `WaterFilter` base class, we would have to rewrite the `FilterID` and `UsageCount` logic inside every single filter type (Carbon, Chemical, UV, etc.). Besides using inheritance, we write that code once in the parent class. If we ever need to add a new feature to *all* filters—such as a "Last Inspection Date"—we only have to add it to the base class, and every subclass will automatically receive it. This reduces bugs and makes the system much easier to maintain.

3. **Polymorphism:** In your code, how did the program "know" which version of the method to run (e.g., the Solar version vs. the Wind version) at runtime?

The program "know" which version of the method to run (e.g., the Solar version vs. the Wind version at runtime is handled through **Dynamic Binding** (or Late Binding). In the code, the base class method is marked as `abstract` or `virtual`, and the subclasses use the `override` keyword.

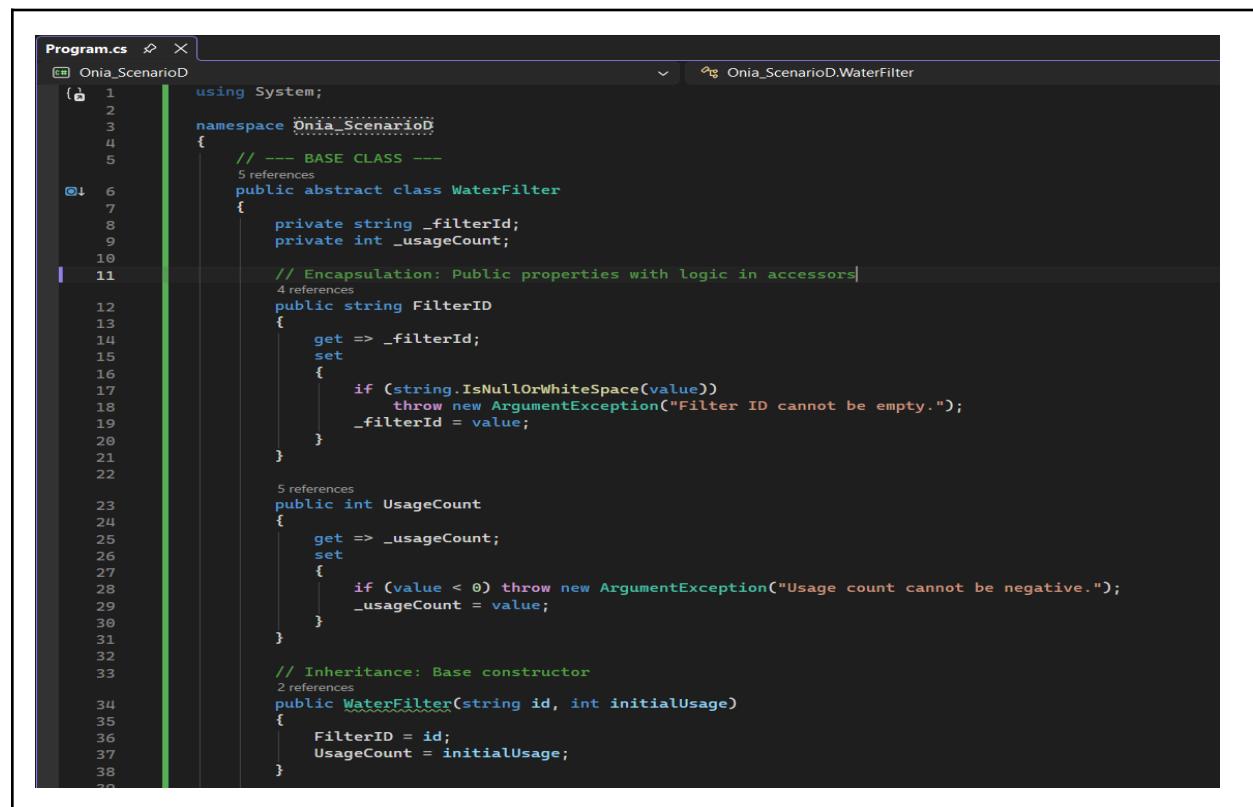
When you call `ProcessWater()`, the .NET Runtime checks the actual "type" of the object created in memory (the instance). Even if the variable is stored as a generic `WaterFilter`, the runtime sees that the object was instantiated as a `ChemicalFilter` and redirects the call to that specific version. This allows the program to be flexible and handle different behaviors using the same command.

4. **Robustness:** Explain the difference between `catch (Exception ex)` and a specific catch like `catch (ArgumentException ex)`. Which is a better practice?

- The difference between `catch (Exception ex)` and a specific catch like `catch (ArgumentException ex)` is the `catch (Exception ex)` is a General Catch that can catches everything including unexpected system failures while `catch (ArgumentException ex)` is a Specific Catch that can handles only a specific types of errors like an invalid input.
- The better practice between those two is the `catch (Exception ex)` because of **three main reasons:** first is the **Precision** because it allows the program to provide accurate feedback. Second is **Debugging Efficiency** because If you catch everything as a generic error, you might accidentally mask a serious crash (like a memory failure) as a simple data entry mistake. And Lastly is the **Error Hierarchy** because you handle known issues gracefully while leaving the general `Exception` block as a final, last-resort safety net for the unexpected.

---

#### SCREENSHOTS OF CODES AND OUTPUT IN THE CHOSEN SCENARIO:



The screenshot shows a code editor window with the file `Program.cs` open. The code defines a class `WaterFilter` with properties `FilterID` and `UsageCount`, both with logic to validate their values. It also includes a constructor and an abstract method `ProcessWater()`.

```
Program.cs
1  using System;
2
3  namespace Onia_ScenarioD
4  {
5      // --- BASE CLASS ---
6      public abstract class WaterFilter
7      {
8          private string _filterId;
9          private int _usageCount;
10
11         // Encapsulation: Public properties with logic in accessors
12         public string FilterID
13         {
14             get => _filterId;
15             set
16             {
17                 if (string.IsNullOrWhiteSpace(value))
18                     throw new ArgumentException("Filter ID cannot be empty.");
19                 _filterId = value;
20             }
21
22             5 references
23             public int UsageCount
24             {
25                 get => _usageCount;
26                 set
27                 {
28                     if (value < 0) throw new ArgumentException("Usage count cannot be negative.");
29                     _usageCount = value;
30                 }
31
32             // Inheritance: Base constructor
33             2 references
34             public WaterFilter(string id, int initialUsage)
35             {
36                 FilterID = id;
37                 UsageCount = initialUsage;
38             }
39
40             // Polymorphism: Abstract method to be overridden
41             3 references
42             public abstract void ProcessWater();
43
44             // Polymorphism: Virtual method for efficiency calculation
45             1 reference
46             public virtual double CalculateEfficiency()
47             {
48                 return 1.0;
49             }
50
51             // Polymorphism: Abstract property to be overridden
52             3 references
53             public abstract string GetStatus();
54
55             // Polymorphism: Virtual property for usage history
56             1 reference
57             public virtual string GetUsageHistory()
58             {
59                 return "No history available";
60             }
61
62             // Polymorphism: Abstract method to be overridden
63             3 references
64             public abstract void RecordUsage(int amount);
65
66             // Polymorphism: Virtual method for usage tracking
67             1 reference
68             public virtual void TrackUsage(int amount)
69             {
70                 UsageCount += amount;
71             }
72
73             // Polymorphism: Abstract property to be overridden
74             3 references
75             public abstract string GetFilterType();
76
77             // Polymorphism: Virtual property for filter details
78             1 reference
79             public virtual string GetFilterDetails()
80             {
81                 return "Basic Water Filter";
82             }
83
84             // Polymorphism: Abstract method to be overridden
85             3 references
86             public abstract void Clean();
87
88             // Polymorphism: Virtual method for cleaning
89             1 reference
90             public virtual void PerformCleaning()
91             {
92                 Console.WriteLine("Filter cleaned successfully.");
93             }
94
95             // Polymorphism: Abstract property to be overridden
96             3 references
97             public abstract string GetMaintenanceStatus();
98
99             // Polymorphism: Virtual property for maintenance status
100            1 reference
101            public virtual string GetMaintenanceDetails()
102            {
103                return "No maintenance required";
104            }
105
106            // Polymorphism: Abstract method to be overridden
107            3 references
108            public abstract void ReplaceFilter();
109
110            // Polymorphism: Virtual method for filter replacement
111            1 reference
112            public virtual void PerformFilterReplacement()
113            {
114                Console.WriteLine("Filter replaced successfully.");
115            }
116
117            // Polymorphism: Abstract property to be overridden
118            3 references
119            public abstract string GetFilterStatus();
120
121            // Polymorphism: Virtual property for filter status
122            1 reference
123            public virtual string GetFilterDetails()
124            {
125                return "Filter is operational";
126            }
127
128            // Polymorphism: Abstract method to be overridden
129            3 references
130            public abstract void RecordUsage(int amount);
131
132            // Polymorphism: Virtual method for usage tracking
133            1 reference
134            public virtual void TrackUsage(int amount)
135            {
136                UsageCount += amount;
137            }
138
139            // Polymorphism: Abstract property to be overridden
140            3 references
141            public abstract string GetFilterType();
142
143            // Polymorphism: Virtual property for filter details
144            1 reference
145            public virtual string GetFilterDetails()
146            {
147                return "Basic Water Filter";
148            }
149
150            // Polymorphism: Abstract method to be overridden
151            3 references
152            public abstract void Clean();
153
154            // Polymorphism: Virtual method for cleaning
155            1 reference
156            public virtual void PerformCleaning()
157            {
158                Console.WriteLine("Filter cleaned successfully.");
159            }
160
161            // Polymorphism: Abstract property to be overridden
162            3 references
163            public abstract string GetMaintenanceStatus();
164
165            // Polymorphism: Virtual property for maintenance status
166            1 reference
167            public virtual string GetMaintenanceDetails()
168            {
169                return "No maintenance required";
170            }
171
172            // Polymorphism: Abstract method to be overridden
173            3 references
174            public abstract void ReplaceFilter();
175
176            // Polymorphism: Virtual method for filter replacement
177            1 reference
178            public virtual void PerformFilterReplacement()
179            {
180                Console.WriteLine("Filter replaced successfully.");
181            }
182
183            // Polymorphism: Abstract property to be overridden
184            3 references
185            public abstract string GetFilterStatus();
186
187            // Polymorphism: Virtual property for filter status
188            1 reference
189            public virtual string GetFilterDetails()
190            {
191                return "Filter is operational";
192            }
193
194            // Polymorphism: Abstract method to be overridden
195            3 references
196            public abstract void RecordUsage(int amount);
197
198            // Polymorphism: Virtual method for usage tracking
199            1 reference
200            public virtual void TrackUsage(int amount)
201            {
202                UsageCount += amount;
203            }
204
205            // Polymorphism: Abstract property to be overridden
206            3 references
207            public abstract string GetFilterType();
208
209            // Polymorphism: Virtual property for filter details
210            1 reference
211            public virtual string GetFilterDetails()
212            {
213                return "Basic Water Filter";
214            }
215
216            // Polymorphism: Abstract method to be overridden
217            3 references
218            public abstract void Clean();
219
220            // Polymorphism: Virtual method for cleaning
221            1 reference
222            public virtual void PerformCleaning()
223            {
224                Console.WriteLine("Filter cleaned successfully.");
225            }
226
227            // Polymorphism: Abstract property to be overridden
228            3 references
229            public abstract string GetMaintenanceStatus();
230
231            // Polymorphism: Virtual property for maintenance status
232            1 reference
233            public virtual string GetMaintenanceDetails()
234            {
235                return "No maintenance required";
236            }
237
238            // Polymorphism: Abstract method to be overridden
239            3 references
240            public abstract void ReplaceFilter();
241
242            // Polymorphism: Virtual method for filter replacement
243            1 reference
244            public virtual void PerformFilterReplacement()
245            {
246                Console.WriteLine("Filter replaced successfully.");
247            }
248
249            // Polymorphism: Abstract property to be overridden
250            3 references
251            public abstract string GetFilterStatus();
252
253            // Polymorphism: Virtual property for filter status
254            1 reference
255            public virtual string GetFilterDetails()
256            {
257                return "Filter is operational";
258            }
259
260            // Polymorphism: Abstract method to be overridden
261            3 references
262            public abstract void RecordUsage(int amount);
263
264            // Polymorphism: Virtual method for usage tracking
265            1 reference
266            public virtual void TrackUsage(int amount)
267            {
268                UsageCount += amount;
269            }
270
271            // Polymorphism: Abstract property to be overridden
272            3 references
273            public abstract string GetFilterType();
274
275            // Polymorphism: Virtual property for filter details
276            1 reference
277            public virtual string GetFilterDetails()
278            {
279                return "Basic Water Filter";
280            }
281
282            // Polymorphism: Abstract method to be overridden
283            3 references
284            public abstract void Clean();
285
286            // Polymorphism: Virtual method for cleaning
287            1 reference
288            public virtual void PerformCleaning()
289            {
290                Console.WriteLine("Filter cleaned successfully.");
291            }
292
293            // Polymorphism: Abstract property to be overridden
294            3 references
295            public abstract string GetMaintenanceStatus();
296
297            // Polymorphism: Virtual property for maintenance status
298            1 reference
299            public virtual string GetMaintenanceDetails()
300            {
301                return "No maintenance required";
302            }
303
304            // Polymorphism: Abstract method to be overridden
305            3 references
306            public abstract void ReplaceFilter();
307
308            // Polymorphism: Virtual method for filter replacement
309            1 reference
310            public virtual void PerformFilterReplacement()
311            {
312                Console.WriteLine("Filter replaced successfully.");
313            }
314
315            // Polymorphism: Abstract property to be overridden
316            3 references
317            public abstract string GetFilterStatus();
318
319            // Polymorphism: Virtual property for filter status
320            1 reference
321            public virtual string GetFilterDetails()
322            {
323                return "Filter is operational";
324            }
325
326            // Polymorphism: Abstract method to be overridden
327            3 references
328            public abstract void RecordUsage(int amount);
329
330            // Polymorphism: Virtual method for usage tracking
331            1 reference
332            public virtual void TrackUsage(int amount)
333            {
334                UsageCount += amount;
335            }
336
337            // Polymorphism: Abstract property to be overridden
338            3 references
339            public abstract string GetFilterType();
340
341            // Polymorphism: Virtual property for filter details
342            1 reference
343            public virtual string GetFilterDetails()
344            {
345                return "Basic Water Filter";
346            }
347
348            // Polymorphism: Abstract method to be overridden
349            3 references
350            public abstract void Clean();
351
352            // Polymorphism: Virtual method for cleaning
353            1 reference
354            public virtual void PerformCleaning()
355            {
356                Console.WriteLine("Filter cleaned successfully.");
357            }
358
359            // Polymorphism: Abstract property to be overridden
360            3 references
361            public abstract string GetMaintenanceStatus();
362
363            // Polymorphism: Virtual property for maintenance status
364            1 reference
365            public virtual string GetMaintenanceDetails()
366            {
367                return "No maintenance required";
368            }
369
370            // Polymorphism: Abstract method to be overridden
371            3 references
372            public abstract void ReplaceFilter();
373
374            // Polymorphism: Virtual method for filter replacement
375            1 reference
376            public virtual void PerformFilterReplacement()
377            {
378                Console.WriteLine("Filter replaced successfully.");
379            }
380
381            // Polymorphism: Abstract property to be overridden
382            3 references
383            public abstract string GetFilterStatus();
384
385            // Polymorphism: Virtual property for filter status
386            1 reference
387            public virtual string GetFilterDetails()
388            {
389                return "Filter is operational";
390            }
391
392            // Polymorphism: Abstract method to be overridden
393            3 references
394            public abstract void RecordUsage(int amount);
395
396            // Polymorphism: Virtual method for usage tracking
397            1 reference
398            public virtual void TrackUsage(int amount)
399            {
400                UsageCount += amount;
401            }
402
403            // Polymorphism: Abstract property to be overridden
404            3 references
405            public abstract string GetFilterType();
406
407            // Polymorphism: Virtual property for filter details
408            1 reference
409            public virtual string GetFilterDetails()
410            {
411                return "Basic Water Filter";
412            }
413
414            // Polymorphism: Abstract method to be overridden
415            3 references
416            public abstract void Clean();
417
418            // Polymorphism: Virtual method for cleaning
419            1 reference
420            public virtual void PerformCleaning()
421            {
422                Console.WriteLine("Filter cleaned successfully.");
423            }
424
425            // Polymorphism: Abstract property to be overridden
426            3 references
427            public abstract string GetMaintenanceStatus();
428
429            // Polymorphism: Virtual property for maintenance status
430            1 reference
431            public virtual string GetMaintenanceDetails()
432            {
433                return "No maintenance required";
434            }
435
436            // Polymorphism: Abstract method to be overridden
437            3 references
438            public abstract void ReplaceFilter();
439
440            // Polymorphism: Virtual method for filter replacement
441            1 reference
442            public virtual void PerformFilterReplacement()
443            {
444                Console.WriteLine("Filter replaced successfully.");
445            }
446
447            // Polymorphism: Abstract property to be overridden
448            3 references
449            public abstract string GetFilterStatus();
450
451            // Polymorphism: Virtual property for filter status
452            1 reference
453            public virtual string GetFilterDetails()
454            {
455                return "Filter is operational";
456            }
457
458            // Polymorphism: Abstract method to be overridden
459            3 references
460            public abstract void RecordUsage(int amount);
461
462            // Polymorphism: Virtual method for usage tracking
463            1 reference
464            public virtual void TrackUsage(int amount)
465            {
466                UsageCount += amount;
467            }
468
469            // Polymorphism: Abstract property to be overridden
470            3 references
471            public abstract string GetFilterType();
472
473            // Polymorphism: Virtual property for filter details
474            1 reference
475            public virtual string GetFilterDetails()
476            {
477                return "Basic Water Filter";
478            }
479
480            // Polymorphism: Abstract method to be overridden
481            3 references
482            public abstract void Clean();
483
484            // Polymorphism: Virtual method for cleaning
485            1 reference
486            public virtual void PerformCleaning()
487            {
488                Console.WriteLine("Filter cleaned successfully.");
489            }
490
491            // Polymorphism: Abstract property to be overridden
492            3 references
493            public abstract string GetMaintenanceStatus();
494
495            // Polymorphism: Virtual property for maintenance status
496            1 reference
497            public virtual string GetMaintenanceDetails()
498            {
499                return "No maintenance required";
500            }
501
502            // Polymorphism: Abstract method to be overridden
503            3 references
504            public abstract void ReplaceFilter();
505
506            // Polymorphism: Virtual method for filter replacement
507            1 reference
508            public virtual void PerformFilterReplacement()
509            {
510                Console.WriteLine("Filter replaced successfully.");
511            }
512
513            // Polymorphism: Abstract property to be overridden
514            3 references
515            public abstract string GetFilterStatus();
516
517            // Polymorphism: Virtual property for filter status
518            1 reference
519            public virtual string GetFilterDetails()
520            {
521                return "Filter is operational";
522            }
523
524            // Polymorphism: Abstract method to be overridden
525            3 references
526            public abstract void RecordUsage(int amount);
527
528            // Polymorphism: Virtual method for usage tracking
529            1 reference
530            public virtual void TrackUsage(int amount)
531            {
532                UsageCount += amount;
533            }
534
535            // Polymorphism: Abstract property to be overridden
536            3 references
537            public abstract string GetFilterType();
538
539            // Polymorphism: Virtual property for filter details
540            1 reference
541            public virtual string GetFilterDetails()
542            {
543                return "Basic Water Filter";
544            }
545
546            // Polymorphism: Abstract method to be overridden
547            3 references
548            public abstract void Clean();
549
550            // Polymorphism: Virtual method for cleaning
551            1 reference
552            public virtual void PerformCleaning()
553            {
554                Console.WriteLine("Filter cleaned successfully.");
555            }
556
557            // Polymorphism: Abstract property to be overridden
558            3 references
559            public abstract string GetMaintenanceStatus();
560
561            // Polymorphism: Virtual property for maintenance status
562            1 reference
563            public virtual string GetMaintenanceDetails()
564            {
565                return "No maintenance required";
566            }
567
568            // Polymorphism: Abstract method to be overridden
569            3 references
570            public abstract void ReplaceFilter();
571
572            // Polymorphism: Virtual method for filter replacement
573            1 reference
574            public virtual void PerformFilterReplacement()
575            {
576                Console.WriteLine("Filter replaced successfully.");
577            }
578
579            // Polymorphism: Abstract property to be overridden
580            3 references
581            public abstract string GetFilterStatus();
582
583            // Polymorphism: Virtual property for filter status
584            1 reference
585            public virtual string GetFilterDetails()
586            {
587                return "Filter is operational";
588            }
589
590            // Polymorphism: Abstract method to be overridden
591            3 references
592            public abstract void RecordUsage(int amount);
593
594            // Polymorphism: Virtual method for usage tracking
595            1 reference
596            public virtual void TrackUsage(int amount)
597            {
598                UsageCount += amount;
599            }
600
601            // Polymorphism: Abstract property to be overridden
602            3 references
603            public abstract string GetFilterType();
604
605            // Polymorphism: Virtual property for filter details
606            1 reference
607            public virtual string GetFilterDetails()
608            {
609                return "Basic Water Filter";
610            }
611
612            // Polymorphism: Abstract method to be overridden
613            3 references
614            public abstract void Clean();
615
616            // Polymorphism: Virtual method for cleaning
617            1 reference
618            public virtual void PerformCleaning()
619            {
620                Console.WriteLine("Filter cleaned successfully.");
621            }
622
623            // Polymorphism: Abstract property to be overridden
624            3 references
625            public abstract string GetMaintenanceStatus();
626
627            // Polymorphism: Virtual property for maintenance status
628            1 reference
629            public virtual string GetMaintenanceDetails()
630            {
631                return "No maintenance required";
632            }
633
634            // Polymorphism: Abstract method to be overridden
635            3 references
636            public abstract void ReplaceFilter();
637
638            // Polymorphism: Virtual method for filter replacement
639            1 reference
640            public virtual void PerformFilterReplacement()
641            {
642                Console.WriteLine("Filter replaced successfully.");
643            }
644
645            // Polymorphism: Abstract property to be overridden
646            3 references
647            public abstract string GetFilterStatus();
648
649            // Polymorphism: Virtual property for filter status
650            1 reference
651            public virtual string GetFilterDetails()
652            {
653                return "Filter is operational";
654            }
655
656            // Polymorphism: Abstract method to be overridden
657            3 references
658            public abstract void RecordUsage(int amount);
659
660            // Polymorphism: Virtual method for usage tracking
661            1 reference
662            public virtual void TrackUsage(int amount)
663            {
664                UsageCount += amount;
665            }
666
667            // Polymorphism: Abstract property to be overridden
668            3 references
669            public abstract string GetFilterType();
670
671            // Polymorphism: Virtual property for filter details
672            1 reference
673            public virtual string GetFilterDetails()
674            {
675                return "Basic Water Filter";
676            }
677
678            // Polymorphism: Abstract method to be overridden
679            3 references
680            public abstract void Clean();
681
682            // Polymorphism: Virtual method for cleaning
683            1 reference
684            public virtual void PerformCleaning()
685            {
686                Console.WriteLine("Filter cleaned successfully.");
687            }
688
689            // Polymorphism: Abstract property to be overridden
690            3 references
691            public abstract string GetMaintenanceStatus();
692
693            // Polymorphism: Virtual property for maintenance status
694            1 reference
695            public virtual string GetMaintenanceDetails()
696            {
697                return "No maintenance required";
698            }
699
700            // Polymorphism: Abstract method to be overridden
701            3 references
702            public abstract void ReplaceFilter();
703
704            // Polymorphism: Virtual method for filter replacement
705            1 reference
706            public virtual void PerformFilterReplacement()
707            {
708                Console.WriteLine("Filter replaced successfully.");
709            }
710
711            // Polymorphism: Abstract property to be overridden
712            3 references
713            public abstract string GetFilterStatus();
714
715            // Polymorphism: Virtual property for filter status
716            1 reference
717            public virtual string GetFilterDetails()
718            {
719                return "Filter is operational";
720            }
721
722            // Polymorphism: Abstract method to be overridden
723            3 references
724            public abstract void RecordUsage(int amount);
725
726            // Polymorphism: Virtual method for usage tracking
727            1 reference
728            public virtual void TrackUsage(int amount)
729            {
730                UsageCount += amount;
731            }
732
733            // Polymorphism: Abstract property to be overridden
734            3 references
735            public abstract string GetFilterType();
736
737            // Polymorphism: Virtual property for filter details
738            1 reference
739            public virtual string GetFilterDetails()
740            {
741                return "Basic Water Filter";
742            }
743
744            // Polymorphism: Abstract method to be overridden
745            3 references
746            public abstract void Clean();
747
748            // Polymorphism: Virtual method for cleaning
749            1 reference
750            public virtual void PerformCleaning()
751            {
752                Console.WriteLine("Filter cleaned successfully.");
753            }
754
755            // Polymorphism: Abstract property to be overridden
756            3 references
757            public abstract string GetMaintenanceStatus();
758
759            // Polymorphism: Virtual property for maintenance status
760            1 reference
761            public virtual string GetMaintenanceDetails()
762            {
763                return "No maintenance required";
764            }
765
766            // Polymorphism: Abstract method to be overridden
767            3 references
768            public abstract void ReplaceFilter();
769
770            // Polymorphism: Virtual method for filter replacement
771            1 reference
772            public virtual void PerformFilterReplacement()
773            {
774                Console.WriteLine("Filter replaced successfully.");
775            }
776
777            // Polymorphism: Abstract property to be overridden
778            3 references
779            public abstract string GetFilterStatus();
780
781            // Polymorphism: Virtual property for filter status
782            1 reference
783            public virtual string GetFilterDetails()
784            {
785                return "Filter is operational";
786            }
787
788            // Polymorphism: Abstract method to be overridden
789            3 references
790            public abstract void RecordUsage(int amount);
791
792            // Polymorphism: Virtual method for usage tracking
793            1 reference
794            public virtual void TrackUsage(int amount)
795            {
796                UsageCount += amount;
797            }
798
799            // Polymorphism: Abstract property to be overridden
800            3 references
801            public abstract string GetFilterType();
802
803            // Polymorphism: Virtual property for filter details
804            1 reference
805            public virtual string GetFilterDetails()
806            {
807                return "Basic Water Filter";
808            }
809
810            // Polymorphism: Abstract method to be overridden
811            3 references
812            public abstract void Clean();
813
814            // Polymorphism: Virtual method for cleaning
815            1 reference
816            public virtual void PerformCleaning()
817            {
818                Console.WriteLine("Filter cleaned successfully.");
819            }
820
821            // Polymorphism: Abstract property to be overridden
822            3 references
823            public abstract string GetMaintenanceStatus();
824
825            // Polymorphism: Virtual property for maintenance status
826            1 reference
827            public virtual string GetMaintenanceDetails()
828            {
829                return "No maintenance required";
830            }
831
832            // Polymorphism: Abstract method to be overridden
833            3 references
834            public abstract void ReplaceFilter();
835
836            // Polymorphism: Virtual method for filter replacement
837            1 reference
838            public virtual void PerformFilterReplacement()
839            {
840                Console.WriteLine("Filter replaced successfully.");
841            }
842
843            // Polymorphism: Abstract property to be overridden
844            3 references
845            public abstract string GetFilterStatus();
846
847            // Polymorphism: Virtual property for filter status
848            1 reference
849            public virtual string GetFilterDetails()
850            {
851                return "Filter is operational";
852            }
853
854            // Polymorphism: Abstract method to be overridden
855            3 references
856            public abstract void RecordUsage(int amount);
857
858            // Polymorphism: Virtual method for usage tracking
859            1 reference
860            public virtual void TrackUsage(int amount)
861            {
862                UsageCount += amount;
863            }
864
865            // Polymorphism: Abstract property to be overridden
866            3 references
867            public abstract string GetFilterType();
868
869            // Polymorphism: Virtual property for filter details
870            1 reference
871            public virtual string GetFilterDetails()
872            {
873                return "Basic Water Filter";
874            }
875
876            // Polymorphism: Abstract method to be overridden
877            3 references
878            public abstract void Clean();
879
880            // Polymorphism: Virtual method for cleaning
881            1 reference
882            public virtual void PerformCleaning()
883            {
884                Console.WriteLine("Filter cleaned successfully.");
885            }
886
887            // Polymorphism: Abstract property to be overridden
888            3 references
889            public abstract string GetMaintenanceStatus();
890
891            // Polymorphism: Virtual property for maintenance status
892            1 reference
893            public virtual string GetMaintenanceDetails()
894            {
895                return "No maintenance required";
896            }
897
898            // Polymorphism: Abstract method to be overridden
899            3 references
900            public abstract void ReplaceFilter();
901
902            // Polymorphism: Virtual method for filter replacement
903            1 reference
904            public virtual void PerformFilterReplacement()
905            {
906                Console.WriteLine("Filter replaced successfully.");
907            }
908
909            // Polymorphism: Abstract property to be overridden
910            3 references
911            public abstract string GetFilterStatus();
912
913            // Polymorphism: Virtual property for filter status
914            1 reference
915            public virtual string GetFilterDetails()
916            {
917                return "Filter is operational";
918            }
919
920            // Polymorphism: Abstract method to be overridden
921            3 references
922            public abstract void RecordUsage(int amount);
923
924            // Polymorphism: Virtual method for usage tracking
925            1 reference
926            public virtual void TrackUsage(int amount)
927            {
928                UsageCount += amount;
929            }
930
931            // Polymorphism: Abstract property to be overridden
932            3 references
933            public abstract string GetFilterType();
934
935            // Polymorphism: Virtual property for filter details
936            1 reference
937            public virtual string GetFilterDetails()
938            {
939                return "Basic Water Filter";
940            }
941
942            // Polymorphism: Abstract method to be overridden
943            3 references
944            public abstract void Clean();
945
946            // Polymorphism: Virtual method for cleaning
947            1 reference
948            public virtual void PerformCleaning()
949            {
950                Console.WriteLine("Filter cleaned successfully.");
951            }
952
953            // Polymorphism: Abstract property to be overridden
954            3 references
955            public abstract string GetMaintenanceStatus();
956
957            // Polymorphism: Virtual property for maintenance status
958            1 reference
959            public virtual string GetMaintenanceDetails()
960            {
961                return "No maintenance required";
962            }
963
964            // Polymorphism: Abstract method to be overridden
965            3 references
966            public abstract void ReplaceFilter();
967
968            // Polymorphism: Virtual method for filter replacement
969            1 reference
970            public virtual void PerformFilterReplacement()
971            {
972                Console.WriteLine("Filter replaced successfully.");
973            }
974
975            // Polymorphism: Abstract property to be overridden
976            3 references
977            public abstract string GetFilterStatus();
978
979            // Polymorphism: Virtual property for filter status
980            1 reference
981            public virtual string GetFilterDetails()
982            {
983                return "Filter is operational";
984            }
985
986            // Polymorphism: Abstract method to be overridden
987            3 references
988            public abstract void RecordUsage(int amount);
989
990            // Polymorphism: Virtual method for usage tracking
991            1 reference
992            public virtual void TrackUsage(int amount)
993            {
994                UsageCount += amount;
995            }
996
997            // Polymorphism: Abstract property to be overridden
998            3 references
999            public abstract string GetFilterType();
1000
1001            // Polymorphism: Virtual property for filter details
1002            1 reference
1003            public virtual string GetFilterDetails()
1004            {
1005                return "Basic Water Filter";
1006            }
1007
1008            // Polymorphism: Abstract method to be overridden
1009            3 references
1010            public abstract void Clean();
1011
1012            // Polymorphism: Virtual method for cleaning
1013            1 reference
1014            public virtual void PerformCleaning()
1015            {
1016                Console.WriteLine("Filter cleaned successfully.");
1017            }
1018
1019            // Polymorphism: Abstract property to be overridden
1020            3 references
1021            public abstract string GetMaintenanceStatus();
1022
1023            // Polymorphism: Virtual property for maintenance status
1024            1 reference
1025            public virtual string GetMaintenanceDetails()
1026            {
1027                return "No maintenance required";
1028            }
1029
1030            // Polymorphism: Abstract method to be overridden
1031            3 references
1032            public abstract void ReplaceFilter();
1033
1034            // Polymorphism: Virtual method for filter replacement
1035            1 reference
1036            public virtual void PerformFilterReplacement()
1037            {
1038                Console.WriteLine("Filter replaced successfully.");
1039            }
1040
1041            // Polymorphism: Abstract property to be overridden
1042            3 references
1043            public abstract string GetFilterStatus();
1044
1045            // Polymorphism: Virtual property for filter status
1046            1 reference
1047            public virtual string GetFilterDetails()
1048            {
1049                return "Filter is operational";
1050            }
1051
1052            // Polymorphism: Abstract method to be overridden
1053            3 references
1054            public abstract void RecordUsage(int amount);
1055
1056            // Polymorphism: Virtual method for usage tracking
1057            1 reference
1058            public virtual void TrackUsage(int amount)
1059            {
1060                UsageCount += amount;
1061            }
1062
1063            // Polymorphism: Abstract property to be overridden
1064            3 references
1065            public abstract string GetFilterType();
1066
1067            // Polymorphism: Virtual property for filter details
1068            1 reference
1069            public virtual string GetFilterDetails()
1070            {
1071                return "Basic Water Filter";
1072            }
1073
1074            // Polymorphism: Abstract method to be overridden
1075            3 references
1076            public abstract void Clean();
1077
1078            // Polymorphism: Virtual method for cleaning
1079            1 reference
1080            public virtual void PerformCleaning()
1081            {
1082                Console.WriteLine("Filter cleaned successfully.");
1083            }
1084
1085            // Polymorphism: Abstract property to be overridden
1086            3 references
1087            public abstract string GetMaintenanceStatus();
1088
1089            // Polymorphism: Virtual property for maintenance status
1090            1 reference
1091            public virtual string GetMaintenanceDetails()
1092            {
1093                return "No maintenance required";
1094            }
1095
1096            // Polymorphism: Abstract method to be overridden
1097            3 references
1098            public abstract void ReplaceFilter();
1099
1100            // Polymorphism: Virtual method for filter replacement
1101            1 reference
1102            public virtual void PerformFilterReplacement()
1103            {
1104                Console.WriteLine("Filter replaced successfully.");
1105            }
1106
1107            // Polymorphism: Abstract property to be overridden
1108            3 references
1109            public abstract string GetFilterStatus();
1110
1111            // Polymorphism: Virtual property for filter status
1112            1 reference
1113            public virtual string GetFilterDetails()
1114            {
1115                return "Filter is operational";
1116            }
1117
1118            // Polymorphism: Abstract method to be overridden
1119            3 references
1120            public abstract void RecordUsage(int amount);
1121
1122            // Polymorphism: Virtual method for usage tracking
1123            1 reference
1124            public virtual void TrackUsage(int amount)
1125            {
1126                UsageCount += amount;
1127            }
1128
1129            // Polymorphism: Abstract property to be overridden
1130            3 references
1131            public abstract string GetFilterType();
1132
1133            // Polymorphism: Virtual property for filter details
1134            1 reference
1135            public virtual string GetFilterDetails()
1136            {
1137                return "Basic Water Filter";
1138            }
1139
1140            // Polymorphism: Abstract method to be overridden
1141            3 references
1142            public abstract void Clean();
1143
1144            // Polymorphism: Virtual method for cleaning
1145            1 reference
1146            public virtual void PerformCleaning()
1147            {
1148                Console.WriteLine("Filter cleaned successfully.");
1149            }
1150
1151            // Polymorphism: Abstract property to be overridden
1152            3 references
1153            public abstract string GetMaintenanceStatus();
1154
1155            // Polymorphism: Virtual property for maintenance status
1156            1 reference
1157            public virtual string GetMaintenanceDetails()
1158            {
1159                return "No maintenance required";
1160            }
1161
1162            // Polymorphism: Abstract method to be overridden
1163            3 references
1164            public abstract void ReplaceFilter();
1165
1166            // Polymorphism: Virtual method for filter replacement
1167            1 reference
1168            public virtual void PerformFilterReplacement()
1169            {
1170                Console.WriteLine("Filter replaced successfully.");
1171            }
1172
1173            // Polymorphism: Abstract property to be overridden
1174            3 references
1175            public abstract string GetFilterStatus();
1176
1177            // Polymorphism: Virtual property for filter status
1178            1 reference
1179            public virtual string GetFilterDetails()
1180            {
1181                return "Filter is operational";
1182            }
1183
1184            // Polymorphism: Abstract method to be overridden
1185            3 references
1186            public abstract void RecordUsage(int amount);
1187
1188            // Polymorphism: Virtual method for usage tracking
1189            1 reference
1190            public virtual void TrackUsage(int amount)
1191            {
1192                UsageCount += amount;
1193            }
1194
1195            // Polymorphism: Abstract property to be overridden
1196            3 references
1197            public abstract string GetFilterType();
1198
1199            // Polymorphism: Virtual property for filter details
1200            1 reference
1201            public virtual string GetFilterDetails()
1202            {
1203                return "Basic Water Filter";
1204            }
1205
1206            // Polymorphism: Abstract method to be overridden
1207            3 references
1208            public abstract void Clean();
1209
1210            // Polymorphism: Virtual method for cleaning
1211            1 reference
1212            public virtual void PerformCleaning()
1213            {
1214                Console.WriteLine("Filter cleaned successfully.");
1215            }
1216
1217            // Polymorphism: Abstract property to be overridden
1218            3 references
1219            public abstract string GetMaintenanceStatus();
1220
1221            // Polymorphism: Virtual property for maintenance status
1222            1 reference
1223            public virtual string GetMaintenanceDetails()
1224            {
1225                return "No maintenance required";
1226            }
1227
1228            // Polymorphism: Abstract method to be overridden
1229            3 references
1230            public abstract void ReplaceFilter();
1231
1232            // Polymorphism: Virtual method for filter replacement
1233            1 reference
1234            public virtual void PerformFilterReplacement()
1235           
```

Program.cs

```
// --- SUB-CLASS: CHEMICAL ---
public class ChemicalFilter : WaterFilter
{
    private int _chemicalLevel;

    public int ChemicalLevel
    {
        get => _chemicalLevel;
        set => _chemicalLevel = value;
    }

    public ChemicalFilter(string id, int usage) : base(id, usage) { }

    public override void ProcessWater()
    {
        Console.WriteLine($"[Chemical Filter {FilterID}] is neutralizing chlorine levels. Usage increased.");
        UsageCount++;
    }
}
```

```
Program.cs
Onia_ScenarioD
Onia_ScenarioD.WaterFilter

94 } }
95 // --- MAIN EXECUTION ---
96 0 references
97 class Program
98 {
99 0 references
100 static void Main(string[] args)
101 {
102     // Robustness: Try-Catch-Finally block
103     try
104     {
105         Console.WriteLine("---- Clean-Water Plant Operations Starting ---\n");
106         // Scenario: A new chemical filter with 0 usage
107         ChemicalFilter chemFilter = new ChemicalFilter("CHEM-101", 0);
108         // This call will trigger the ProcessWater logic
109         chemFilter.ProcessWater();
110
111         // Intentional reset to 0 to demonstrate the DivideByZeroException requirement
112         chemFilter.UsageCount = 0;
113         Console.WriteLine($"Calculating efficiency for {chemFilter.FilterID}...");
114
115         double efficiency = chemFilter.CalculateEfficiency();
116         Console.WriteLine($"Efficiency: {efficiency}%");
117     }
118     catch (DivideByZeroException ex)
119     {
120         Console.WriteLine($"ERROR: {ex.Message}");
121     }
122     catch (Exception ex)
123     {
124         Console.WriteLine($"GENERAL ERROR: {ex.Message}");
125     }
126     finally
127     {
128         // Robustness: Final block execution
129         Console.WriteLine("\nSession Ended. System Shutdown.");
130     }
131 }
132 }
133 }
134 }
```

## OUTPUT FOR THIS CODE:

```
Microsoft Visual Studio Debug
--- Clean-Water Plant Operations Starting ---

[Chemical Filter CHEM-101] is neutralizing chlorine levels. Usage increased.
Calculating efficiency for CHEM-101...
ERROR: Cannot calculate efficiency: No water has passed through this filter yet.

Session Ended. System Shutdown.

C:\Users\james\onnia\OneDrive\Desktop\C# Folder\Onia_ScenarioD\Onia_ScenarioD\bin\Debug\net10.0\Onia_ScenarioD.exe (process 14076) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```