

FH-OÖ Hagenberg/HSD
SDP3, WS 2025
Übung 7



Name: Simon Offenberger / Simon Vogelhuber

Aufwand in h: siehe Doku

Mat.Nr: S2410306027 / S2410306014

Punkte:

Übungsgruppe: 1

korrigiert:

Beispiel 1 (24 Punkte) Kaffeeautomat: Entwerfen Sie aus der nachfolgenden Spezifikation ein Klassendiagramm, instanzieren Sie dieses und implementieren Sie die Funktionalität entsprechend. Verwenden Sie dabei das Decorator-Pattern:

Ein Kaffeeautomat bietet verschiedene Kaffeesorten (Verlängerter, Espresso, Koffeinfrei) mit entsprechenden Zutaten (Zucker, Milch u. Schlagobers) an. Die Kaffeesorten und Zutaten haben jeweils unterschiedliche Preise und eine entsprechende Beschreibung. Eine Methode `GetCost()` liefert den Gesamtpreis des ausgewählten Kaffees und die Methode `GetDescription()` liefert dazu die entsprechende Beschreibung als `std::string` um z.B. folgende Ausgaben auf `std::cout` zu ermöglichen:

```
Espresso: Zucker, Schlagobers 2.89 Euro
Verlängerter: Zucker, Milch 2.93 Euro
Koffeinfrei: Milch, Milch, Schlagobers 3.15 Euro
```

Die Beschreibung und die Preise werden in einer separaten Preisliste (Konstanten in Header, Klasse, oder Namespace) festgelegt. Zutaten können mehrfach gewählt werden!

Achten Sie beim Design darauf, dass zusätzliche Kaffeesorten und Zutaten hinzugefügt werden können, ohne die bereits bestehenden Klassen verändern zu müssen. Beweisen Sie dies durch das Hinzufügen der Kaffeesorte "Mocca" und der Zutat "Sojamilch".

Implementieren Sie einen Testtreiber der verschiedene Kaffees mit unterschiedlichen Zutaten erzeugt, alle Methoden ausreichend testet und anschließend deren Beschreibung auf `std::cout` ausgibt.

Implementieren Sie weiters eine Klasse `CoffeePreparation` die nach dem FIFO-Prinzip arbeitet und folgende Schnittstelle aufweist:

```
1 void Prepare(/*Coffee*/);           //adds and prepares a coffee
2 void Display(std::ostream& os);    //outputs all coffees in preparation
3 /*Coffee*/ Finished();            //removes the prepared coffee
```

Testen Sie die Klasse ebenfalls ausführlich im Testtreiber!

Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen und begründen Sie diese. Verfassen Sie weiters eine Systemdokumentation (entsprechend den Vorgaben aus Übung1)!

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!



Systemdokumentation Projekt CoffeeMachine

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 29. Dezember 2025

Inhaltsverzeichnis

1 Organisatorisches	6
1.1 Team	6
1.2 Aufteilung der Verantwortlichkeitsbereiche	6
1.3 Aufwand	7
2 Anforderungsdefinition (Systemspezifikation)	8
2.1 Systemüberblick	8
2.2 Funktionale Anforderungen	8
2.2.1 Kaffeesorten (Basiskomponenten)	8
2.2.2 Zutaten (Decorator)	9
2.2.3 Preis- und Beschreibungsdaten (Preisliste)	9
2.2.4 Preis- und Beschreibungsausgabe	9
2.3 Erweiterbarkeit (Designanforderung)	10
2.4 Zusatzkomponente: coffeePreparation (FIFO)	10
2.4.1 Verhaltensanforderungen coffeePreparation	11
3 Systementwurf	12
3.1 Klassendiagramm	12
3.2 Designentscheidungen	13
4 Dokumentation der Komponenten (Klassen)	13
5 Testprotokollierung	14
6 Quellcode	20
6.1 Object.hpp	20
6.2 ICoffee.hpp	21
6.3 CoffeeInfo.hpp	22
6.4 Ingredient.hpp	23
6.5 Ingredient.cpp	24
6.6 CoffeePreparation.hpp	25
6.7 CoffeePreparation.cpp	26
6.8 SojaMilk.hpp	27
6.9 SojaMilk.cpp	28
6.10 Milk.hpp	29

6.11	Milk.cpp	30
6.12	Sugar.hpp	31
6.13	Sugar.cpp	32
6.14	Cream.hpp	33
6.15	Cream.cpp	34
6.16	ExtendedOne.hpp	35
6.17	ExtendedOne.cpp	36
6.18	Espresso.hpp	37
6.19	Espresso.cpp	38
6.20	Decaff.hpp	39
6.21	Decaff.cpp	40
6.22	Mocha.hpp	41
6.23	Mocha.cpp	42
6.24	main.cpp	43
6.25	Test.hpp	49

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@fh-hagenberg.at
- Simon Vogelhuber, Matr.-Nr.: S2410306014, E-Mail: Simon.Vogelhuber@fh-hagenberg.at

1.2 Aufteilung der Verantwortlichkeitsbereiche

- Simon Offenberger
 - Design Klassendiagramm
 - Implementierung und Test der Klassen:
 - * ICoffee,
 - * Ingredient,
 - * SojaMilk,
 - * Milk,
 - * Sugar,
 - * Cream,
 - Implementierung des Testtreibers
 - Dokumentation
- Simon Vogelhuber
 - Design Klassendiagramm

- Implementierung und Komponententest der Klassen:
 - * CoffeePreparation,
 - * ExtendedOne,
 - * Espresso,
 - * Decaff,
 - * Mocha,
 - * CoffeeInfo
- Implementierung des Testtreibers
- Dokumentation

1.3 Aufwand

- Simon Offenberger: geschätzt 4 Ph / tatsächlich 4 Ph
- Simon Vogelhuber: geschätzt 4 Ph / tatsächlich 3 Ph

2 Anforderungsdefinition (Systemspezifikation)

Das zu entwickelnde System dient der Simulation eines einfachen **Kaffeeautomaten** zur softwaretechnischen Abbildung von Kaffeevarianten und optionalen Zutaten. Ziel ist es, verschiedene Kaffeesorten dynamisch mit beliebig vielen (auch mehrfach wählbaren) Zutaten zu kombinieren und daraus **Gesamtpreis** sowie **Beschreibung** zu ermitteln. Die Implementierung hat das **Decorator-Pattern** zu verwenden und so gestaltet zu sein, dass neue Kaffeesorten und Zutaten ohne Anpassung bestehender Klassen ergänzt werden können.

2.1 Systemüberblick

Das System verwaltet zwei Elementgruppen:

- **Kaffeesorten** (Basisprodukte), z.B. *Verlängerter, Espresso, Koffeinfrei*
- **Zutaten** (Erweiterungen/Dekoratoren), z.B. *Zucker, Milch, Schlagobers*

Eine konkrete Bestellung besteht aus genau **einer Kaffeesorte** und **0..n Zutaten**. Zutaten dürfen **mehrzahl** hinzugefügt werden. Für jede Bestellung müssen eine **textuelle Beschreibung** sowie der **Gesamtpreis** ausgegeben werden können.

2.2 Funktionale Anforderungen

2.2.1 Kaffeesorten (Basiskomponenten)

- Jede Kaffeesorte besitzt eine **Beschreibung** (z.B. Name der Sorte) sowie einen **Basispreis**.
- Eine Kaffeesorte stellt mindestens folgende Operationen bereit:

- `GetCost()` liefert den Basispreis (ohne Zutaten).
- `GetDescription()` liefert die Basisbeschreibung (ohne Zutatenliste).

2.2.2 Zutaten (Decorator)

- Jede Zutat referenziert **genau ein** bereits existierendes Kaffeeobjekt (Kaffeesorte oder bereits dekoriertes Objekt).
- Eine Zutat erweitert:
 - den Preis um ihren **Zutatenpreis**,
 - die Beschreibung um ihren **Zutatennamen** (als Bestandteil der Zutatenliste).
- Zutaten müssen **beliebig oft** hintereinander hinzugefügt werden können (z.B. *Milch, Milch*).

2.2.3 Preis- und Beschreibungsdaten (Preisliste)

- Preise und Beschreibungen von Kaffeesorten und Zutaten werden in einer separaten **Preisliste** (z.B. Konstanten in Header/Klasse/Namespace) definiert.
- Die fachliche Logik (Berechnung und Ausgabe) darf nicht durch „Hard-coding“ von Preisen in vielen Klassen unübersichtlich werden.

2.2.4 Preis- und Beschreibungsausgabe

- `GetCost()` liefert den **Gesamtpreis** des aktuell zusammengesetzten Kaffees (Basis + alle Zutaten).
- `GetDescription()` liefert eine **druckbare Beschreibung** zur Aus-

gabe auf `std::cout`, inkl. Zutaten in der Reihenfolge ihrer Hinzufügung.

- Die Ausgabe soll mindestens Ausgaben in folgender Form ermöglichen:
Espresso: Zucker, Schlagobers 2.89 Euro

2.3 Erweiterbarkeit (Designanforderung)

- Das Design muss sicherstellen, dass **neue Kaffeesorten** und **neue Zutaten** hinzugefügt werden können, **ohne bestehende Klassen ändern zu müssen** (Open/Closed-Prinzip).
- Diese Erweiterbarkeit ist nachzuweisen durch das Hinzufügen mindestens von:
 - Kaffeesorte „**Mocca**“
 - Zutat „**Sojamilch**“

2.4 Zusatzkomponente: `coffeePreparation (FIFO)`

Zusätzlich ist eine Klasse `coffeePreparation` zu implementieren, die Bestellungen nach dem **FIFO-Prinzip** verwaltet. Sie stellt folgende Schnittstelle bereit:

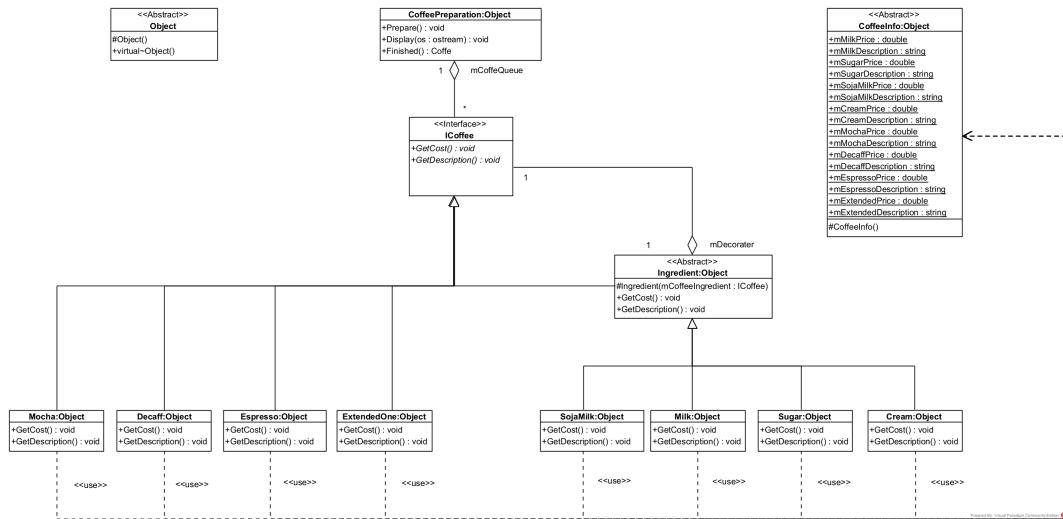
- `void Prepare(Coffee*);` // fügt einen Kaffee zur Vorbereitung hinzu
- `void Display(std::ostream& os);` // gibt alle aktuell vorbereiteten Kaffees aus
- `Coffee* Finished();` // entfernt und liefert den zuerst vorbereiteten Kaffee

2.4.1 Verhaltensanforderungen coffeePreparation

- Prepare reiht Bestellungen in eine Warteschlange ein.
- Finished liefert das **älteste** Element der Warteschlange und entfernt es aus der Struktur.
- Display gibt den aktuellen Inhalt der Warteschlange in geeigneter Form aus (z.B. je Bestellung eine Zeile mit Beschreibung und Preis).

3 Systementwurf

3.1 Klassendiagramm



3.2 Designentscheidungen

Wie in der Aufgabenstellung gefordert wurde das Decorator-Pattern verwendet, um die Kaffeesorten und Zutaten zu implementieren.

Dies ermöglicht es, neue Zutaten und Kaffeesorten hinzuzufügen, ohne bestehende Klassen ändern zu müssen.

Dies wurde durch hinzufügen der Klassen `Mocha` und `SojaMilk` demonstriert.

Die Basiskomponente ist das Interface `ICoffee`, welche die Schnittstelle für alle Kaffeesorten und Zutaten definiert. Die konkreten Kaffeesorten (`Espresso`, `Decaff`, `Mocha`, `ExtendedOne`) erben direkt von `ICoffee` und implementieren die Methoden `GetCost()` und `GetDescription()`. Die Zutatenklassen (`Ingredient` und deren Unterklassen `Milk`, `SojaMilk`, `Sugar`, `Cream`) erben ebenfalls von `ICoffee` und enthalten eine Referenz auf ein `ICoffee`-Objekt, das sie dekorieren. Die Methoden `GetCost()` und `GetDescription()` der Zutatenklassen rufen die entsprechenden Methoden des dekorierten Objekts auf und erweitern deren Ergebnisse um den Preis und die Beschreibung der Zutat.

4 Dokumentation der Komponenten (Klassen)

Die HTML-Startdatei befindet sich im Verzeichnis [./doxy/html/index.html](#)

5 Testprotokollierung

```
1 ****
2 **** TESTCASE START
3 ****
4 ****
5
6 Test Espresso
7
8 Test ICoffee Description
9 [Test OK] Result: (Expected: Espresso: == Result: Espresso:)
10
11 Test ICoffee Price
12 [Test OK] Result: (Expected: 3 == Result: 3)
13
14 Test for Exception in Testcase
15 [Test OK] Result: (Expected: true == Result: true)
16
17
18 ****
19
20
21 **** TESTCASE START
22 ****
23 ****
24
25 Test Mocha
26
27 Test ICoffee Description
28 [Test OK] Result: (Expected: Mocha: == Result: Mocha:)
29
30 Test ICoffee Price
31 [Test OK] Result: (Expected: 2.7 == Result: 2.7)
32
33 Test for Exception in Testcase
34 [Test OK] Result: (Expected: true == Result: true)
35
36
37 ****
38
39
40 **** TESTCASE START
41 ****
42 ****
```

```
43 Test Decaff
44
45
46 Test ICoffee Description
47 [Test OK] Result: (Expected: Decaff: == Result: Decaff:)
48
49 Test ICoffee Price
50 [Test OK] Result: (Expected: 2.8 == Result: 2.8)
51
52 Test for Exception in Testcase
53 [Test OK] Result: (Expected: true == Result: true)
54
55
56 ****
57
58
59 ****
60 TESTCASE START
61 ****
62
63 Test Extended One
64
65 Test ICoffee Description
66 [Test OK] Result: (Expected: Extended One: == Result: Extended One:)
67
68 Test ICoffee Price
69 [Test OK] Result: (Expected: 5 == Result: 5)
70
71 Test for Exception in Testcase
72 [Test OK] Result: (Expected: true == Result: true)
73
74
75 ****
76
77
78 ****
79 TESTCASE START
80 ****
81
82 Test Espresso with Milk
83
84 Test ICoffee Description
85 [Test OK] Result: (Expected: Espresso: Milk, == Result: Espresso: Milk,)
```

```
87 Test ICoffee Price
88 [Test OK] Result: (Expected: 5.5 == Result: 5.5)
89
90 Test for Exception in Testcase
91 [Test OK] Result: (Expected: true == Result: true)
92
93
94 ****
95
96
97 ****
98 TESTCASE START
99 ****
100
101 Test Extended One with SojaMilk
102
103 Test ICoffee Description
104 [Test OK] Result: (Expected: Extended One: SojaMilk, == Result: Extended
105   ↯ One: SojaMilk,)
106
107 Test ICoffee Price
108 [Test OK] Result: (Expected: 20 == Result: 20)
109
110 Test for Exception in Testcase
111 [Test OK] Result: (Expected: true == Result: true)
112
113 ****
114
115
116 ****
117 TESTCASE START
118 ****
119
120 Test Mocha with Sugar
121
122 Test ICoffee Description
123 [Test OK] Result: (Expected: Mocha: Sugar, == Result: Mocha: Sugar,)
124
125 Test ICoffee Price
126 [Test OK] Result: (Expected: 4.2 == Result: 4.2)
127
128 Test for Exception in Testcase
129 [Test OK] Result: (Expected: true == Result: true)
```

```
130
131
132 ****
133
134
135 ****
136         TESTCASE START
137 ****
138
139 Test Decaff with Cream
140
141 Test ICoffee Description
142 [Test OK] Result: (Expected: Decaff: Cream, == Result: Decaff: Cream, )
143
144 Test ICoffee Price
145 [Test OK] Result: (Expected: 4.8 == Result: 4.8)
146
147 Test for Exception in Testcase
148 [Test OK] Result: (Expected: true == Result: true)
149
150
151 ****
152
153
154 ****
155         TESTCASE START
156 ****
157
158 Test Decaff with Cream and Cream
159
160 Test ICoffee Description
161 [Test OK] Result: (Expected: Decaff: Cream, Cream, == Result: Decaff: Cream
162     ↪ , Cream, )
163
164 Test ICoffee Price
165 [Test OK] Result: (Expected: 6.8 == Result: 6.8)
166
167 Test for Exception in Testcase
168 [Test OK] Result: (Expected: true == Result: true)
169
170 ****
171
172
```

```
173 ****  
174     TESTCASE START  
175 ****  
176  
177 Test Mocha alla Diabetes  
178  
179 Test ICoffee Description  
180 [Test OK] Result: (Expected: Mocha: Sugar, Sugar, Sugar, Sugar, Sugar,  
181     ↪ Sugar, Sugar, Sugar, Sugar, == Result: Mocha: Sugar, Sugar, Sugar,  
182     ↪ Sugar, Sugar, Sugar, Sugar, Sugar, )  
183  
184 Test ICoffee Price  
185 [Test OK] Result: (Expected: 16.2 == Result: 16.2)  
186  
187  
188  
189 ****  
190  
191 Test CoffeePreparation Display 1  
192 [Test OK] Result: (Expected: Espresso: Milk 5.5 Euro  
193 Extended One: SojaMilk 20 Euro  
194     == Result: Espresso: Milk 5.5 Euro  
195 Extended One: SojaMilk 20 Euro  
196 )  
197  
198 Test CoffeePreparation Display 2  
199 [Test OK] Result: (Expected: Extended One: SojaMilk 20 Euro  
200     == Result: Extended One: SojaMilk 20 Euro  
201 )  
202  
203 Test for Exception in Testcase  
204 [Test OK] Result: (Expected: true == Result: true)  
205  
206 Test Exception Bad Ostream in CoffeePreparation  
207 [Test OK] Result: (Expected: Error Bad Ostream == Result: Error Bad Ostream  
208     ↪ )  
209  
210 Test Exception Queue is Empty Display  
211 [Test OK] Result: (Expected: Error No Coffe in the Machine! == Result:  
212     ↪ Error No Coffe in the Machine!)  
213  
214 Test Exception Queue is Empty Finished
```

```
213 [Test OK] Result: (Expected: Error No Coffe in the Machine! == Result:  
214     ↢ Error No Coffe in the Machine!)  
215 Test for Exception in Ingederal CTOR  
216 [Test OK] Result: (Expected: Error Nullptr! == Result: Error Nullptr!)  
217  
218 TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```
1  /**
2   * @file Object.h
3   * @brief Defines a minimal base object with virtual destructor support.
4   */
5 #ifndef OBJECT_H
6 #define OBJECT_H
7
8 #include <string>
9
10 class Object{
11 public:
12 protected:
13
14 /**
15  * @brief Base constructor for derived objects.
16  */
17 Object() = default;
18 public:
19 /**
20  * @brief Virtual destructor to allow safe polymorphic deletion.
21  */
22 virtual ~Object() = default;
23 };
24
25 #endif // OBJECT_H
```

6.2 ICoffee.hpp

```
1  /**
2   * @file ICoffee.hpp
3   * @brief Declares the abstract coffee interface for pricing and descriptions.
4   */
5 #ifndef ICOFFEE_HPP
6 #define ICOFFEE_HPP
7
8 #include <memory>
9 #include <string>
10
11 class ICoffee {
12 public:
13
14     using Uptr = std::unique_ptr<ICoffee>;
15
16     /**
17      * @brief Compute the total cost of the coffee including decorations.
18      * @return Final price in Euros.
19      */
20     virtual double GetCost() = 0;
21
22     /**
23      * @brief Provide a human-readable description of the coffee order.
24      * @return Description string ending with a separator.
25      */
26     virtual std::string GetDescription() = 0;
27
28     virtual ~ICoffee() = default;
29 };
30
31
32 #endif // !ICOFFEE_HPP
```

6.3 CoffeeInfo.hpp

```
1  /**
2   * @file CoffeeInfo.hpp
3   * @brief Defines static price and label constants for all coffee drinks and add-ons.
4   */
5 #ifndef COFFEE_INFO_HPP
6 #define COFFEE_INFO_HPP
7
8 #include <string>
9 #include "Object.h"
10
11 class CoffeeInfo : Object {
12 public:
13
14     inline static const double mEspressoPrice = 3;
15     inline static const std::string mEspressoInfo = "Espresso";
16
17     inline static const double mDecaffPrice = 2.8;
18     inline static const std::string mDecaffInfo = "Decaff";
19
20     inline static const double mMochaPrice = 2.7;
21     inline static const std::string mMochaInfo = "Mocha";
22
23     inline static const double mExtendedPrice = 5;
24     inline static const std::string mExtendedInfo = "Extended_One";
25
26     inline static const double mMilkPrice = 2.5;
27     inline static const std::string mMilkInfo = "Milk";
28
29     inline static const double mSojaMilkPrice = 15;
30     inline static const std::string mSojaMilkInfo = "SojaMilk";
31
32     inline static const double mSugarPrice = 1.5;
33     inline static const std::string mSugarInfo = "Sugar";
34
35     inline static const double mCreamPrice = 2;
36     inline static const std::string mCreamInfo = "Cream";
37
38 protected:
39     CoffeeInfo() = default;
40 };
41
42
43 #endif // !COFFEE_INFO_HPP
```

6.4 Ingredient.hpp

```
1  /**
2   * @file Ingredient.hpp
3   * @brief Declares the decorator base class that augments an ICoffee.
4   */
5 #ifndef INGREDIENT_HPP
6 #define INGREDIENT_HPP
7
8 #include "Object.h"
9 #include "ICoffee.hpp"
10
11 class Ingredient : public ICoffee , public Object {
12 public:
13     inline static const std::string ERROR_NULLPTR = "Error_Nullptr!";
14
15     /**
16      * @brief Forward cost request to the decorated coffee.
17      * @return Accumulated coffee price.
18      */
19     virtual double GetCost() override;
20
21     /**
22      * @brief Forward description request to the decorated coffee.
23      * @return Aggregated description string.
24      */
25     virtual std::string GetDescription() override;
26
27
28     // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
29     void operator=(Ingredient& ind) = delete;
30     Ingredient(Ingredient& ind) = delete;
31
32
33 protected:
34
35     /**
36      * @brief Construct a decorator around another coffee.
37      * @param mCoffeeIngredient Coffee instance to wrap; must not be null.
38      */
39     Ingredient(ICoffee::Uptr mCoffeeIngredient);
40
41     ICoffee::Uptr mDecorator;
42 };
43
44
45 #endif // !INGREDIENT_HPP
```

6.5 Ingredient.cpp

```
1 /**
2  * @file Ingredient.cpp
3  * @brief Implements the decorator base class used by ingredient add-ons.
4 */
5 #include "Ingredient.hpp"
6 #include <stdexcept>
7
8
9 Ingredient::Ingredient(ICoffee::Uptr mCoffeeIngredient)
10 {
11     if (mCoffeeIngredient == nullptr) throw std::invalid_argument(ERROR_NULLPTR);
12     mDecorator = move(mCoffeeIngredient);
13 }
14
15 double Ingredient::GetCost()
16 {
17     return mDecorator->GetCost();
18 }
19
20 std::string Ingredient::GetDescription()
21 {
22     return mDecorator->GetDescription();
23 }
```

6.6 CoffeePreparation.hpp

```
1 /**
2  * @file CoffeePreparation.hpp
3  * @brief Declares a queue-based coffee preparation pipeline with output helpers.
4 */
5 #ifndef COFFEE_PREPARATION_HPP
6 #define COFFEE_PREPARATION_HPP
7
8 #include "ICoffee.hpp"
9 #include <deque>
10 #include <string>
11 #include <iostream>
12
13 class CoffeePreparation {
14 public:
15     inline static const std::string ERROR_NULLPTR = "Error_Nullptr!";
16     inline static const std::string ERROR_BAD_OSTREAM = "Error_Bad_Ostream";
17     inline static const std::string ERROR_NO_COFFE_IN_MACHINE = "Error_No_Coffe_in_the_Machine!";
18
19     CoffeePreparation() = default;
20
21 /**
22  * @brief Enqueue a coffee for preparation.
23  * @param coffee Ownership of the coffee instance to queue.
24  */
25     void Prepare(ICoffee::Uptr coffee);
26
27 /**
28  * @brief Prints all coffees description and price to a stream.
29  * @param ost Target output stream; must be valid.
30  */
31     void Display(std::ostream& ost);
32
33 /**
34  * @brief Remove and return the next finished coffee.
35  * @return Unique pointer to the prepared coffee.
36  */
37     ICoffee::Uptr Finished();
38
39     void operator=(CoffeePreparation & prep) = delete;
40     CoffeePreparation(CoffeePreparation& prep) = delete;
41
42 private:
43     std::deque<ICoffee::Uptr> mCoffeeQueue;
44 };
45
46
47 #endif // !COFFEE_PREPARATION_HPP
```

6.7 CoffeePreparation.cpp

```
1 /**
2  * @file CoffeePreparation.cpp
3  * @brief Implements the coffee preparation queue with display and pickup helpers.
4 */
5 #include "CoffeePreparation.hpp"
6
7 void CoffeePreparation::Prepare(ICoffee::Uptr coffee)
8 {
9     if (coffee == nullptr) throw std::invalid_argument(ERROR_NULLPTR);
10
11     mCoffeeQueue.push_back(move(coffee));
12 }
13
14 void CoffeePreparation::Display(std::ostream& ost)
15 {
16     if (ost.bad()) throw std::invalid_argument(ERROR_BAD_OSTREAM);
17     if (mCoffeeQueue.empty()) throw std::runtime_error(ERROR_NO_COFFE_IN_MACHINE);
18
19     for(const auto & coffee : mCoffeeQueue) {
20         std::string description = coffee->GetDescription();
21
22         // discard the last "," to fulfill the requirement
23         // in the excercise
24         if (!description.empty()) {
25             *description.rbegin() = '\u202c';
26         }
27         ost << description;
28         ost << coffee->GetCost() << " Euro" << std::endl;
29     }
30 }
31
32
33 }
34
35 ICoffee::Uptr CoffeePreparation::Finished()
36 {
37     if (mCoffeeQueue.empty()) throw std::runtime_error(ERROR_NO_COFFE_IN_MACHINE);
38
39     ICoffee::Uptr retCoffee = move(mCoffeeQueue.front());
40     mCoffeeQueue.pop_front();
41
42     return move(retCoffee);
43 }
```

6.8 SojaMilk.hpp

```
1 /**
2  * @file SojaMilk.hpp
3  * @brief Declares the soja milk ingredient decorator for coffee orders.
4 */
5 #ifndef SOJA_MILK_HPP
6 #define SOJA_MILK_HPP
7
8 #include <string>
9
10 #include "Object.h"
11 #include "Ingredient.hpp"
12
13 class SojaMilk : public Ingredient {
14 public:
15
16 /**
17  * @brief Wrap a coffee with soja milk.
18  * @param cof Coffee to decorate.
19  */
20 SojaMilk(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}
21
22 /**
23  * @brief Return price including soja milk surcharge.
24  */
25 virtual double GetCost() override;
26
27 /**
28  * @brief Append soja milk label to description.
29  */
30 virtual std::string GetDescription() override;
31
32 // explicitly delete Assign Op and Copy Ctor to prevent untested behaviour
33 void operator=(SojaMilk& ind) = delete;
34 SojaMilk(SojaMilk& ind) = delete;
35 };
36
37 #endif // !SOJA_MILK_HPP
```

6.9 SojaMilk.cpp

```
1 /**
2  * @file SojaMilk.cpp
3  * @brief Implements the soja milk ingredient decorator behavior.
4 */
5 #include "SojaMilk.hpp"
6 #include "CoffeeInfo.hpp"
7
8 double SojaMilk::GetCost()
9 {
10     return CoffeeInfo::mSojaMilkPrice + Ingredient::GetCost();
11 }
12
13 std::string SojaMilk::GetDescription()
14 {
15     return Ingredient::GetDescription() + " „ " + CoffeeInfo::mSojaMilkInfo + ", ";
16 }
```

6.10 Milk.hpp

```
1  /**
2   * @file Milk.hpp
3   * @brief Declares the milk ingredient decorator for coffee orders.
4   */
5  #ifndef MILK_HPP
6  #define MILK_HPP
7
8  #include <string>
9
10 #include "Object.h"
11 #include "Ingredient.hpp"
12
13 class Milk : public Ingredient {
14 public:
15
16  /**
17   * @brief Wrap a coffee with milk.
18   * @param cof Coffee to decorate.
19   */
20  Milk(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}
21
22  /**
23   * @brief Return price including milk surcharge.
24   */
25  virtual double GetCost() override;
26
27  /**
28   * @brief Append milk label to description.
29   */
30  virtual std::string GetDescription() override;
31
32 // explicitly delete Assign Op and Copy Ctor to prevent untested behaviour
33 void operator=(Milk& ind) = delete;
34 Milk(Milk& ind) = delete;
35 };
36
37 #endif // !MILK_HPP
```

6.11 Milk.cpp

```
1 /**
2  * @file Milk.cpp
3  * @brief Implements the milk ingredient decorator behavior.
4 */
5 #include "Milk.hpp"
6 #include "CoffeeInfo.hpp"
7
8 double Milk::GetCost()
9 {
10     return CoffeeInfo::mMilkPrice + Ingredient::GetCost();
11 }
12
13 std::string Milk::GetDescription()
14 {
15     return Ingredient::GetDescription() + " „ " + CoffeeInfo::mMilkInfo + ",";
16 }
```

6.12 Sugar.hpp

```
1  /**
2   * @file Sugar.hpp
3   * @brief Declares the sugar ingredient decorator for coffee orders.
4   */
5  #ifndef SUGAR_HPP
6  #define SUGAR_HPP
7
8  #include <string>
9
10 #include "Object.h"
11 #include "Ingredient.hpp"
12
13 class Sugar : public Ingredient {
14 public:
15
16  /**
17   * @brief Wrap a coffee with sugar.
18   * @param cof Coffee to decorate.
19   */
20  Sugar(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}
21
22  /**
23   * @brief Return price including sugar surcharge.
24   */
25  virtual double GetCost() override;
26
27  /**
28   * @brief Append sugar label to description.
29   */
30  virtual std::string GetDescription() override;
31
32 // explicitly delete Assign Op and Copy Ctor to prevent untested behaviour
33 void operator=(Sugar& ind) = delete;
34 Sugar(Sugar& ind) = delete;
35 };
36
37 #endif // !SUGAR_HPP
```

6.13 Sugar.cpp

```
1 /**
2  * @file Sugar.cpp
3  * @brief Implements the sugar ingredient decorator behavior.
4 */
5 #include "Sugar.hpp"
6 #include "CoffeeInfo.hpp"
7
8 double Sugar::GetCost()
9 {
10     return CoffeeInfo::mSugarPrice + Ingredient::GetCost();
11 }
12
13 std::string Sugar::GetDescription()
14 {
15     return Ingredient::GetDescription() + " „ " + CoffeeInfo::mSugarInfo + ",";
16 }
```

6.14 Cream.hpp

```
1  /**
2   * @file Cream.hpp
3   * @brief Declares the cream ingredient decorator for coffee orders.
4   */
5  #ifndef CREAM_HPP
6  #define CREAM_HPP
7
8  #include <string>
9
10 #include "Object.h"
11 #include "Ingredient.hpp"
12
13 class Cream : public Ingredient {
14 public:
15     using Uptr = std::unique_ptr<Cream>;
16
17
18     /**
19      * @brief Wrap a coffee with cream.
20      * @param cof Coffee to decorate.
21      */
22     Cream(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}
23
24     /**
25      * @brief Return price including cream surcharge.
26      */
27     virtual double GetCost() override;
28
29     /**
30      * @brief Append cream label to description.
31      */
32     virtual std::string GetDescription() override;
33
34     // explicitly delete Assign Op and Copy Ctor to prevent untested behaviour
35     void operator=(Cream& ind) = delete;
36     Cream(Cream& ind) = delete;
37
38 };
39
40 #endif // !CREAM_HPP
```

6.15 Cream.cpp

```
1 /**
2  * @file Cream.cpp
3  * @brief Implements the cream ingredient decorator behavior.
4 */
5 #include "Cream.hpp"
6 #include "CoffeeInfo.hpp"
7
8 double Cream::GetCost()
9 {
10     return CoffeeInfo::mCreamPrice + Ingredient::GetCost();
11 }
12
13 std::string Cream::GetDescription()
14 {
15     return Ingredient::GetDescription() + " „ " + CoffeeInfo::mCreamInfo + ",";
16 }
```

6.16 ExtendedOne.hpp

```
1  /**
2   * @file ExtendedOne.hpp
3   * @brief Declares the extended coffee variant implementation of ICoffee.
4   */
5 #ifndef EXTENDED_ONE_HPP
6 #define EXTENDED_ONE_HPP
7
8 #include "Object.h"
9 #include "ICoffee.hpp"
10
11
12 class ExtendedOne : public ICoffee, public Object {
13 public:
14
15     using Sptr = std::shared_ptr<ExtendedOne>;
16
17     ExtendedOne() = default;
18
19
20     /**
21      * @brief Return the price of the extended variant.
22      */
23     virtual double GetCost() override;
24
25     /**
26      * @brief Provide the extended variant description label.
27      */
28     virtual std::string GetDescription() override;
29
30     // explicitly delete Assign Op and Copy Ctor to prevent untested behaviour
31     void operator=(ExtendedOne& ind) = delete;
32     ExtendedOne(ExtendedOne& ind) = delete;
33 };
34
35 #endif // !EXTENDED_ONE_HPP
```

6.17 ExtendedOne.cpp

```
1 /**
2  * @file ExtendedOne.cpp
3  * @brief Implements the extended coffee variant pricing and description.
4 */
5 #include "ExtendedOne.hpp"
6 #include "CoffeeInfo.hpp"
7
8
9 double ExtendedOne::GetCost()
10 {
11     return CoffeeInfo::mExtendedPrice;
12 }
13
14 std::string ExtendedOne::GetDescription()
15 {
16     return CoffeeInfo::mExtendedInfo + ":";  
17 }
```

6.18 Espresso.hpp

```
1 /**
2  * @file Espresso.hpp
3  * @brief Declares the espresso coffee implementation of ICoffee.
4 */
5 #ifndef ESPRESSO_HPP
6 #define ESPRESSO_HPP
7
8 #include "Object.h"
9 #include "ICoffee.hpp"
10
11
12 class Espresso : public ICoffee , public Object {
13 public:
14
15     using Sptr = std::shared_ptr<Espresso>;
16
17     Espresso() = default;
18
19 /**
20  * @brief Return the price of an espresso.
21  */
22     virtual double GetCost() override;
23
24 /**
25  * @brief Provide the espresso description label.
26  */
27     virtual std::string GetDescription() override;
28
29
30 // explicitly delete Assign Op and Copy Ctor to prevent untested behaviour
31 void operator=(Espresso& ind) = delete;
32 Espresso(Espresso& ind) = delete;
33 };
34
35 #endif // !ESPRESSO_HPP
```

6.19 Espresso.cpp

```
1 /**
2  * @file Espresso.cpp
3  * @brief Implements the espresso coffee pricing and description.
4 */
5 #include "Espresso.hpp"
6 #include "CoffeeInfo.hpp"
7
8
9 double Espresso::GetCost()
10 {
11     return CoffeeInfo::mEspressoPrice;
12 }
13
14 std::string Espresso::GetDescription()
15 {
16     return CoffeeInfo::mEspressoInfo + ":";  
17 }
```

6.20 Decaff.hpp

```
1 /**
2  * @file Decaff.hpp
3  * @brief Declares the decaffeinated coffee implementation of ICoffee.
4 */
5 #ifndef DECAFF_HPP
6 #define DECAFF_HPP
7
8 #include "Object.h"
9 #include "ICoffee.hpp"
10
11 class Decaff : public ICoffee, public Object {
12 public:
13     using Sptr = std::shared_ptr<Decaff>;
14
15     Decaff() = default;
16
17
18 /**
19  * @brief Return the price of a decaffeinated coffee.
20  */
21     virtual double GetCost() override;
22
23 /**
24  * @brief Provide the decaff description label.
25  */
26     virtual std::string GetDescription() override;
27
28 // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
29 void operator=(Decaff& ind) = delete;
30 Decaff(Decaff& ind) = delete;
31
32 };
33
34 #endif // !DECAFF_HPP
```

6.21 Decaff.cpp

```
1 /**
2  * @file Decaff.cpp
3  * @brief Implements the decaffeinated coffee pricing and description.
4 */
5 #include "Decaff.hpp"
6 #include "CoffeeInfo.hpp"
7
8 double Decaff::GetCost()
9 {
10     return CoffeeInfo::mDecaffPrice;
11 }
12
13 std::string Decaff::GetDescription()
14 {
15     return CoffeeInfo::mDecaffInfo + ":";  
16 }
```

6.22 Mocha.hpp

```
1  /**
2   * @file Mocha.hpp
3   * @brief Declares the mocha coffee implementation of ICoffee.
4   */
5 #ifndef MOCHA_HPP
6 #define MOCHA_HPP
7
8 #include "Object.h"
9 #include "ICoffee.hpp"
10
11
12 class Mocha : public ICoffee, public Object {
13 public:
14
15     using Sptr = std::shared_ptr<Mocha>;
16
17     Mocha() = default;
18
19
20     /**
21      * @brief Return the price of a mocha.
22      */
23     virtual double GetCost() override;
24
25     /**
26      * @brief Provide the mocha description label.
27      */
28     virtual std::string GetDescription() override;
29
30     // explicitly delete Assign Op and Copy Ctor to prevent untested behaviour
31     void operator=(Mocha& ind) = delete;
32     Mocha(Mocha& ind) = delete;
33 };
34
35 #endif // !MOCHA_HPP
```

6.23 Mocha.cpp

```
1 /**
2  * @file Mocha.cpp
3  * @brief Implements the mocha coffee pricing and description.
4 */
5 #include "Mocha.hpp"
6 #include "CoffeeInfo.hpp"
7
8 double Mocha::GetCost()
9 {
10     return CoffeeInfo::mMochaPrice;
11 }
12
13 std::string Mocha::GetDescription()
14 {
15     return CoffeeInfo::mMochaInfo + ":";  
16 }
```

6.24 main.cpp

```
1 /**
2  * @file main.cpp
3  * @brief Runs sample preparations and tests for the coffee machine decorators.
4 */
5 #include "vld.h"
6 #include "Mocha.hpp"
7 #include "ExtendedOne.hpp"
8 #include "Decaff.hpp"
9 #include "Espresso.hpp"
10 #include "Milk.hpp"
11 #include "Sugar.hpp"
12 #include "SojaMilk.hpp"
13 #include "Cream.hpp"
14 #include "CoffeePreparation.hpp"
15 #include "Test.hpp"
16 #include "CoffeeInfo.hpp"
17
18 #include <memory>
19 #include <iostream>
20 #include <cassert>
21 #include <sstream>
22 #include <fstream>
23
24 using namespace std;
25
26 static bool TestCoffeeIngridient(std::ostream& ost, ICoffee::Uptr cof, const std::string& description, const double price);
27 static bool TestCoffeeIngridientException(std::ostream& ost);
28 static bool TestCoffeePreparation(std::ostream& ost);
29
30
31 #define WriteOutputFile true
32
33 int main()
34 {
35     bool TestOK = true;
36     ofstream output( "Testoutput.txt" );
37
38     if (!output.is_open()) {
39         cerr << "Konnte Testoutput.txt nicht öffnen" << TestCaseFail;
40         return 1;
41     }
42
43     try {
44
45         cout << TestStart;
46         cout << "Test_Espresso" << endl << endl;
47         TestCoffeeIngridient(std::cout, make_unique<Espresso>(), CoffeeInfo::mEspressoInfo + ":", CoffeeInfo::mEspressoPrice);
48         cout << TestEnd;
49
50         cout << TestStart;
51         cout << "Test_Mocha" << endl << endl;
52         TestCoffeeIngridient(std::cout, make_unique<Mocha>(), CoffeeInfo::mMochaInfo + ":", CoffeeInfo::mMochaPrice);
53         cout << TestEnd;
54
55         cout << TestStart;
56         cout << "Test_Decaff" << endl << endl;
57         TestCoffeeIngridient(std::cout, make_unique<Decaff>(), CoffeeInfo::mDecaffInfo + ":", CoffeeInfo::mDecaffPrice);
58         cout << TestEnd;
59
60         cout << TestStart;
61         cout << "Test_Extended_One" << endl << endl;
62         TestCoffeeIngridient(std::cout, make_unique<ExtendedOne>(), CoffeeInfo::mExtendedInfo + ":", CoffeeInfo::mExtendedPrice);
63         cout << TestEnd;
64
65         cout << TestStart;
66         cout << "Test_Espresso_with_Milk" << endl << endl;
67         TestCoffeeIngridient(std::cout, make_unique<Milk>(make_unique<Espresso>()),
68             CoffeeInfo::mEspressoInfo + ":" + CoffeeInfo::mMilkInfo + ",",
69             CoffeeInfo::mEspressoPrice + CoffeeInfo::mMilkPrice);
70         cout << TestEnd;
71
72         cout << TestStart;
```

```
73     cout << "Test_Extended_One_with_SojaMilk" << endl << endl;
74     TestCoffeeIngridient(std::cout, make_unique<SojaMilk>(make_unique<ExtendedOne>()),
75         CoffeeInfo::mExtendedInfo + ":" + CoffeeInfo::mSojaMilkInfo + ",",
76         CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice);
77     cout << TestEnd;
78
79     cout << TestStart;
80     cout << "Test_Mocha_with_Sugar" << endl << endl;
81     TestCoffeeIngridient(std::cout, make_unique<Sugar>(make_unique<Mocha>()),
82         CoffeeInfo::mMochaInfo + ":" + CoffeeInfo::mSugarInfo + ",",
83         CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice);
84     cout << TestEnd;
85
86     cout << TestStart;
87     cout << "Test_Decaff_with_Cream" << endl << endl;
88     TestCoffeeIngridient(std::cout, make_unique<Cream>(make_unique<Decaff>()),
89         CoffeeInfo::mDecaffInfo + ":" + CoffeeInfo::mCreamInfo + ",",
90         CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice);
91     cout << TestEnd;
92
93     cout << TestStart;
94     cout << "Test_Decaff_with_Cream_and_Cream" << endl << endl;
95     TestCoffeeIngridient(std::cout, make_unique<Cream>(make_unique<Cream>(make_unique<Decaff>())),
96         CoffeeInfo::mDecaffInfo + ":" + CoffeeInfo::mCreamInfo + "," + CoffeeInfo::mCreamInfo + ",",
97         CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice + CoffeeInfo::mCreamPrice);
98     cout << TestEnd;
99
100    cout << TestStart;
101    cout << "Test_Mocha_alla_Diabetes" << endl << endl;
102    TestCoffeeIngridient(std::cout, make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>()))))))),
103        CoffeeInfo::mMochaInfo + ":" + CoffeeInfo::mSugarInfo + ",",
104        + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",",
105        + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",",
106        + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",",
107        + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",",
108        + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",",
109        + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",",
110        + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",",
111        CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice * 9);
112    cout << TestEnd;
113
114
115    TestCoffeePreparation(std::cout);
116
117    TestCoffeeIngridientException(std::cout);
118
119
120    if (WriteOutputFile) {
121
122        output << TestStart;
123        output << "Test_Espresso" << endl << endl;
124        TestCoffeeIngridient(output, make_unique<Espresso>(), CoffeeInfo::mEspressoInfo + ":" , CoffeeInfo::mEspressoPrice);
125        output << TestEnd;
126
127        output << TestStart;
128        output << "Test_Mocha" << endl << endl;
129        TestCoffeeIngridient(output, make_unique<Mocha>(), CoffeeInfo::mMochaInfo + ":" , CoffeeInfo::mMochaPrice);
130        output << TestEnd;
131
132        output << TestStart;
133        output << "Test_Decaff" << endl << endl;
134        TestCoffeeIngridient(output, make_unique<Decaff>(), CoffeeInfo::mDecaffInfo + ":" , CoffeeInfo::mDecaffPrice);
135        output << TestEnd;
136
137        output << TestStart;
138        output << "Test_Extended_One" << endl << endl;
139        TestCoffeeIngridient(output, make_unique<ExtendedOne>(), CoffeeInfo::mExtendedInfo + ":" , CoffeeInfo::mExtendedPrice);
140        output << TestEnd;
141
142        output << TestStart;
143        output << "Test_Espresso_with_Milk" << endl << endl;
144        TestCoffeeIngridient(output, make_unique<Milk>(make_unique<Espresso>())),
145            CoffeeInfo::mEspressoInfo + ":" + CoffeeInfo::mMilkInfo + ",",
146            CoffeeInfo::mEspressoPrice + CoffeeInfo::mMilkPrice);
147        output << TestEnd;
```

```
148     output << TestStart;
149     output << "Test_Extended_One_with_SojaMilk" << endl << endl;
150     TestCoffeeIngridient(output, make_unique<SojaMilk>(make_unique<ExtendedOne>()),
151         CoffeeInfo::mExtendedInfo + ":" + CoffeeInfo::mSojaMilkInfo + ",",
152         CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice);
153     output << TestEnd;
154
155     output << TestStart;
156     output << "Test_Mocha_with_Sugar" << endl << endl;
157     TestCoffeeIngridient(output, make_unique<Sugar>(make_unique<Mocha>()),
158         CoffeeInfo::mMochaInfo + ":" + CoffeeInfo::mSugarInfo + ",",
159         CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice);
160     output << TestEnd;
161
162     output << TestStart;
163     output << "Test_Decaff_with_Cream" << endl << endl;
164     TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Decaff>()),
165         CoffeeInfo::mDecaffInfo + ":" + CoffeeInfo::mCreamInfo + ",",
166         CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice);
167     output << TestEnd;
168
169     output << TestStart;
170     output << "Test_Decaff_with_Cream_and_Cream" << endl << endl;
171     TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Decaff>(),
172         CoffeeInfo::mDecaffInfo + ":" + CoffeeInfo::mCreamInfo + ",," +
173         CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice + CoffeeInfo::mCreamPrice));
174     output << TestEnd;
175
176     output << TestStart;
177     output << "Test_Mocha_alla_Diabetes" << endl << endl;
178     TestCoffeeIngridient(output, make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
179         make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
180             make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
181                 make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
182                     make_unique<Mocha>()))))))),
183             CoffeeInfo::mMochaInfo + ":" + CoffeeInfo::mSugarInfo + ",,"
184             + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",,"
185             + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",,"
186             + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",,"
187             + CoffeeInfo::mSugarInfo + "," + CoffeeInfo::mSugarInfo + ",,"
188             CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice * 9));
189     output << TestEnd;
190
191
192     TestCoffeePreparation(output);
193
194     TestCoffeeIngridientException(output);
195
196
197     if (TestOK) {
198         output << TestCaseOK;
199     }
200     else {
201         output << TestCaseFail;
202     }
203
204     output.close();
205 }
206
207     if (TestOK) {
208         cout << TestCaseOK;
209     }
210     else {
211         cout << TestCaseFail;
212     }
213 }
214 catch (const string& err) {
215     cerr << err << TestCaseFail;
216 }
217 catch (bad_alloc const& error) {
218     cerr << error.what() << TestCaseFail;
219 }
220 catch (const exception& err) {
221     cerr << err.what() << TestCaseFail;
222 }
```

```
223     catch (...) {
224         cerr << "Unhandelt_Exception" << TestCaseFail;
225     }
226
227     if (output.is_open()) output.close();
228
229     return 0;
230
231 }
232
233 bool TestCoffeeIngridient(std::ostream & ost, ICoffee::Uptr cof, const std::string & description, const double price)
234 {
235     assert(cof != nullptr);
236     assert(ost.good());
237
238     std::string error_msg;
239     bool TestOK = true;
240
241     try {
242         TestOK = TestOK && check_dump(ost, "Test_ICoffee_Description", cof->GetDescription(), description);
243         TestOK = TestOK && check_dump(ost, "Test_ICoffee_Price", cof->GetCost(), price);
244     }
245     catch (const string& err) {
246         error_msg = err;
247     }
248     catch (bad_alloc const& error) {
249         error_msg = error.what();
250     }
251     catch (const exception& err) {
252         error_msg = err.what();
253     }
254     catch (...) {
255         error_msg = "Unhandelt_Exception";
256     }
257
258     TestOK = TestOK && check_dump(ost, "Test_for_Exception_in_Testcase", true, error_msg.empty());
259
260     return TestOK;
261 }
262
263 bool TestCoffeeIngridientException(std::ostream& ost)
264 {
265     assert(ost.good());
266
267     std::string error_msg;
268     bool TestOK = true;
269
270     try {
271         ICoffee::Uptr cof = make_unique<Milk>(nullptr);
272     }
273     catch (const string& err) {
274         error_msg = err;
275     }
276     catch (bad_alloc const& error) {
277         error_msg = error.what();
278     }
279     catch (const exception& err) {
280         error_msg = err.what();
281     }
282     catch (...) {
283         error_msg = "Unhandelt_Exception";
284     }
285
286     TestOK = TestOK && check_dump(ost, "Test_for_Exception_in_IngedientCTOR", Ingredient::ERROR_NULLPTR, error_msg);
287
288     return TestOK;
289 }
290
291 bool TestCoffeePreparation(std::ostream& ost) {
292
293     assert(ost.good());
294
295     std::string error_msg;
```

```
298     bool TestOK = true;
299
300     try {
301         CoffeePreparation CoffeeMachine;
302
303         CoffeeMachine.Prepare(make_unique<Milk>(make_unique<Espresso>()));
304         CoffeeMachine.Prepare(make_unique<SojaMilk>(make_unique<ExtendedOne>()));
305
306         stringstream expected_output;
307         stringstream actual_output;
308
309         CoffeeMachine.Display(actual_output);
310
311         expected_output << CoffeeInfo::mEspressoInfo + ":"_ + CoffeeInfo::mMilkInfo + ":"_ << CoffeeInfo::mEspressoPrice + CoffeeInfo::mMilkPrice;
312         expected_output << CoffeeInfo::mExtendedInfo + ":"_ + CoffeeInfo::mSojaMilkInfo + ":"_ << CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice;
313
314         TestOK = TestOK && check_dump(ost, "Test_CoffeePreparation_Display_1", actual_output.str(), expected_output.str());
315
316         ICoffee::Uptr cof = CoffeeMachine.Finished();
317
318         actual_output.str("");
319         expected_output.str("");
320
321         CoffeeMachine.Display(actual_output);
322
323         expected_output << CoffeeInfo::mExtendedInfo + ":"_ + CoffeeInfo::mSojaMilkInfo + ":"_ << CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice;
324
325         TestOK = TestOK && check_dump(ost, "Test_CoffeePreparation_Display_2", actual_output.str(), expected_output.str());
326
327         cof = CoffeeMachine.Finished();
328
329     }
330
331     catch (const string& err) {
332         error_msg = err;
333     }
334     catch (bad_alloc const& error) {
335         error_msg = error.what();
336     }
337     catch (const exception& err) {
338         error_msg = err.what();
339     }
340     catch (...) {
341         error_msg = "Unhandelt_Exception";
342     }
343
344     TestOK = TestOK && check_dump(ost, "Test_for_Exception_in_Testcase", true, error_msg.empty());
345
346     try {
347
348         CoffeePreparation CoffeeMachine;
349
350         stringstream badstream;
351
352         badstream.setstate(ios::badbit);
353
354         CoffeeMachine.Display(badstream);
355     }
356     catch (const string& err) {
357         error_msg = err;
358     }
359     catch (bad_alloc const& error) {
360         error_msg = error.what();
361     }
362     catch (const exception& err) {
363         error_msg = err.what();
364     }
365     catch (...) {
366         error_msg = "Unhandelt_Exception";
367     }
368
369     TestOK = TestOK && check_dump(ost, "Test_Exception_Bad_Ostream_in_CoffeePreparation", CoffeePreparation::ERROR_BAD_OSTREAM, error_ms
370
371     try {
372 }
```

```
373     CoffeePreparation CoffeeMachine;
374     CoffeeMachine.Display(ost);
375 }
376 catch (const string& err) {
377     error_msg = err;
378 }
379 catch (bad_alloc const& error) {
380     error_msg = error.what();
381 }
382 catch (const exception& err) {
383     error_msg = err.what();
384 }
385 }
386 catch (...) {
387     error_msg = "Unhandelt_Exception";
388 }
389 TestOK = TestOK && check_dump(ost, "Test_Exception_Queue_is_Empty_Display", CoffeePreparation::ERROR_NO_COFFE_IN_MACHINE, error_msg);
390
391 try {
392     CoffeePreparation CoffeeMachine;
393     CoffeeMachine.Finished();
394 }
395 catch (const string& err) {
396     error_msg = err;
397 }
398 catch (bad_alloc const& error) {
399     error_msg = error.what();
400 }
401 catch (const exception& err) {
402     error_msg = err.what();
403 }
404 catch (...) {
405     error_msg = "Unhandelt_Exception";
406 }
407
408 return TestOK;
409 }
410
411 TestOK = TestOK && check_dump(ost, "Test_Exception_Queue_is_Finished", CoffeePreparation::ERROR_NO_COFFE_IN_MACHINE, error_msg);
412
413
414
415 }
```

6.25 Test.hpp

```

1 /*****
2 * \file Test.hpp
3 * \brief File that provides a Test Function with a formated output
4 *
5 * \author Simon
6 * \date April 2025
7 *****/
8 #ifndef TEST_HPP
9 #define TEST_HPP
10
11 #include <string>
12 #include <iostream>
13 #include <vector>
14 #include <list>
15 #include <queue>
16 #include <forward_list>
17
18 #define ON 1
19 #define OFF 0
20 #define COLOR_OUTPUT OFF
21
22 // Definitions of colors in order to change the color of the output stream.
23 const std::string colorRed = "\x1B[31m";
24 const std::string colorGreen = "\x1B[32m";
25 const std::string colorWhite = "\x1B[37m";
26
27 inline std::ostream& RED(std::ostream& ost) {
28     if (ost.good()) {
29         ost << colorRed;
30     }
31     return ost;
32 }
33 inline std::ostream& GREEN(std::ostream& ost) {
34     if (ost.good()) {
35         ost << colorGreen;
36     }
37     return ost;
38 }
39 inline std::ostream& WHITE(std::ostream& ost) {
40     if (ost.good()) {
41         ost << colorWhite;
42     }
43     return ost;
44 }
45
46 inline std::ostream& TestStart(std::ostream& ost) {
47     if (ost.good()) {
48         ost << std::endl;
49         ost << "*****" << std::endl;
50         ost << "TESTCASE_<START>" << std::endl;
51         ost << "*****" << std::endl;
52         ost << std::endl;
53     }
54     return ost;
55 }
56
57 inline std::ostream& TestEnd(std::ostream& ost) {
58     if (ost.good()) {
59         ost << std::endl;
60         ost << "*****" << std::endl;
61         ost << std::endl;
62     }
63     return ost;
64 }
65
66 inline std::ostream& TestCaseOK(std::ostream& ost) {
67
68 #if COLOR_OUTPUT
69     if (ost.good()) {
70         ost << colorGreen << "TEST_OK!!" << colorWhite << std::endl;
71     }
72 #else

```

```
73     if (ost.good()) {
74         ost << "TEST_OK!!" << std::endl;
75     }
76 #endif // COLOR_OUTPUT
77
78     return ost;
79 }
80
81 inline std::ostream& TestCaseFail(std::ostream& ost) {
82
83 #if COLOR_OUTPUT
84     if (ost.good()) {
85         ost << colorRed << "TEST_FAILED!!" << colorWhite << std::endl;
86     }
87 #else
88     if (ost.good()) {
89         ost << "TEST_FAILED!!" << std::endl;
90     }
91 #endif // COLOR_OUTPUT
92
93     return ost;
94 }
95
96 /**
97 * \brief function that reports if the testcase was successful.
98 *
99 * \param testcase String that indicates the testcase
100 * \param successful true -> reports to cout test OK
101 * \param successful false -> reports test failed
102 */
103 template <typename T>
104 bool check_dump(std::ostream& ostr, const std::string& testcase, const T& expected, const T& result) {
105     if (ostr.good()) {
106 #if COLOR_OUTPUT
107         if (expected == result) {
108             ostr << testcase << std::endl << colorGreen << "[Test_OK]" << colorWhite << "Result:" << std::boolalpha << expected
109             << std::noboolalpha << std::endl << std::endl;
110         }
111         else {
112             ostr << testcase << std::endl << colorRed << "[Test_FAILED]" << colorWhite << "Result:" << std::boolalpha << expected
113             << std::noboolalpha << std::endl << std::endl;
114         }
115 #else
116         if (expected == result) {
117             ostr << testcase << std::endl << "[Test_OK]" << "Result:" << std::boolalpha << expected << "==" << "Result:" <<
118         }
119         else {
120             ostr << testcase << std::endl << "[Test_FAILED]" << "Result:" << std::boolalpha << expected << "!=" << "Result:" <<
121         }
122 #endif
123         if (ostr.fail()) {
124             std::cerr << "Error: Write_Ostream" << std::endl;
125         }
126     }
127     else {
128         std::cerr << "Error: Bad_Ostream" << std::endl;
129     }
130     return expected == result;
131 }
132
133 template <typename T1, typename T2>
134 std::ostream& operator<< (std::ostream& ost, const std::pair<T1,T2> & p) {
135     if (!ost.good()) throw std::exception{ "Error_bad_Ostream!" };
136     ost << "(" << p.first << "," << p.second << ")";
137     return ost;
138 }
139
140 template <typename T>
141 std::ostream& operator<< (std::ostream& ost, const std::vector<T> & cont) {
142     if (!ost.good()) throw std::exception{ "Error_bad_Ostream!" };
143     std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>(ost, " "));
144     return ost;
145 }
```

```
146
147 template <typename T>
148 std::ostream& operator<< (std::ostream& ost, const std::list<T> & cont) {
149     if (!ost.good()) throw std::exception( "Error_bad_Ostream! " );
150     std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>(ost, " "));
151     return ost;
152 }
153
154 template <typename T>
155 std::ostream& operator<< (std::ostream& ost, const std::deque<T> & cont) {
156     if (!ost.good()) throw std::exception( "Error_bad_Ostream! " );
157     std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>(ost, " "));
158     return ost;
159 }
160
161 template <typename T>
162 std::ostream& operator<< (std::ostream& ost, const std::forward_list<T> & cont) {
163     if (!ost.good()) throw std::exception( "Error_bad_Ostream! " );
164     std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>(ost, " "));
165     return ost;
166 }
167
168
169 #endif // !TEST_HPP
```