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:

```cpp
void Prepare(/*Coffee*/);        //adds and prepares a coffee
void Display(std::ostream& os);  //outputs all coffees in preparation
/*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!

HSD

FH-HAGENBERG

Systemdokumentation Projekt CoffeeMachine

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 3. Jänner 2026

Inhaltsverzeichnis

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 **mehrfach** 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 „Hardcoding" 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
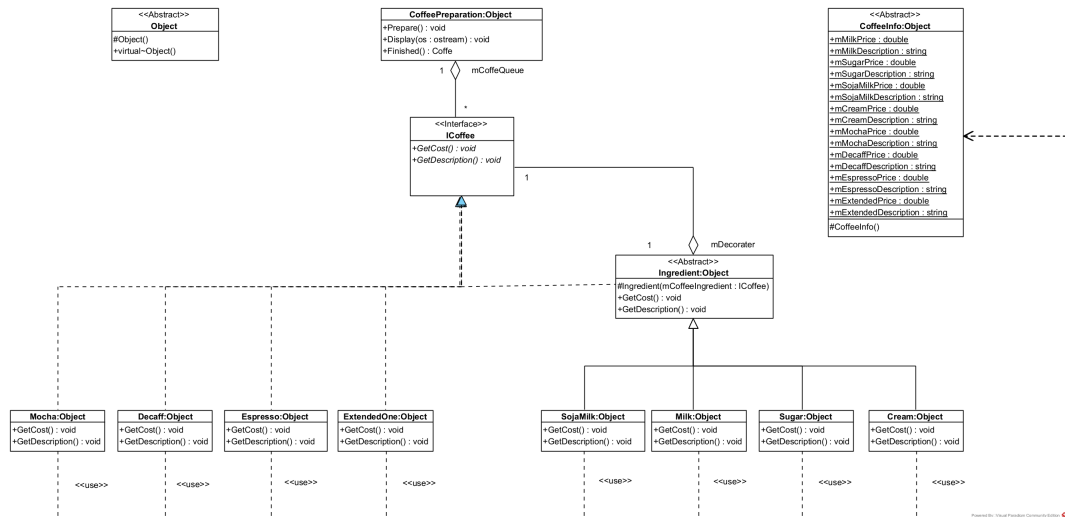
3.2 Designentscheidungen

Wie in der Aufgabenstellung gefordet 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

```
*******************************************
              TESTCASE START
*******************************************

Test Espresso

Test ICoffee Description
[Test OK] Result: (Expected: Espresso: == Result: Espresso:)

Test ICoffee Price
[Test OK] Result: (Expected: 3 == Result: 3)

Test for Exception in Testcase
[Test OK] Result: (Expected: true == Result: true)


*******************************************


*******************************************
              TESTCASE START
*******************************************

Test Mocha

Test ICoffee Description
[Test OK] Result: (Expected: Mocha: == Result: Mocha:)

Test ICoffee Price
[Test OK] Result: (Expected: 2.7 == Result: 2.7)

Test for Exception in Testcase
[Test OK] Result: (Expected: true == Result: true)


*******************************************


*******************************************
              TESTCASE START
*******************************************
```

```
43
44 Test Decaff
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,)
86
```

```
 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
     ↪ One: SojaMilk,)
105
106 Test ICoffee Price
107 [Test OK] Result: (Expected: 20 == Result: 20)
108
109 Test for Exception in Testcase
110 [Test OK] Result: (Expected: true == Result: true)
111
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
     ↪ , Cream,)
162
163 Test ICoffee Price
164 [Test OK] Result: (Expected: 6.8 == Result: 6.8)
165
166 Test for Exception in Testcase
167 [Test OK] Result: (Expected: true == Result: true)
168
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,
      ↪ Sugar, Sugar, Sugar, Sugar, == Result: Mocha: Sugar, Sugar, Sugar,
      ↪ Sugar, Sugar, Sugar, Sugar, Sugar, Sugar,)
181
182 Test ICoffee Price
183 [Test OK] Result: (Expected: 16.2 == Result: 16.2)
184
185 Test for Exception in Testcase
186 [Test OK] Result: (Expected: true == Result: true)
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 Test Exception Queue is Empty Display
210 [Test OK] Result: (Expected: Error No Coffe in the Machine! == Result:
      ↪ Error No Coffe in the Machine!)
211
212 Test Exception Queue is Empty Finished
```

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

6 Quellcode

6.1 Object.hpp

```cpp
/**
 * @file Object.h
 * @brief Defines a minimal base object with virtual destructor support.
 */
#ifndef OBJECT_H
#define OBJECT_H

#include <string>

class Object{
public:

protected:

    /**
     * @brief Base constructor for derived objects.
     */
    Object() = default;
public:
    /**
     * @brief Virtual destructor to allow safe polymorphic deletion.
     */
    virtual ~Object() = default;
};

#endif // OBJECT_H
```

6.2 ICoffee.hpp

```cpp
/**
 * @file ICoffee.hpp
 * @brief Declares the abstract coffee interface for pricing and descriptions.
 */
#ifndef ICOFFEE_HPP
#define  ICOFFEE_HPP

#include <memory>
#include <string>

class ICoffee {
public:

    using Uptr = std::unique_ptr<ICoffee>;

    /**
     * @brief Compute the total cost of the coffee including decorations.
     * @return Final price in Euros.
     */
    virtual double GetCost() const = 0;

    /**
     * @brief Provide a human-readable description of the coffee order.
     * @return Description string ending with a separator.
     */
    virtual std::string GetDescription() const = 0;

    virtual ~ICoffee() = default;
};


#endif // !ICOFFEE_HPP
```

6.3 CoffeeInfo.hpp

```cpp
/**
 * @file CoffeeInfo.hpp
 * @brief Defines static price and label constants for all coffee drinks and add-ons.
 */
#ifndef COFFEE_INFO_HPP
#define  COFFEE_INFO_HPP

#include <string>
#include "Object.h"

class CoffeeInfo : Object {
public:

    inline static const double mEspressoPrice = 3;
    inline static const std::string mEspressoInfo = "Espresso";

    inline static const double mDecaffPrice = 2.8;
    inline static const std::string mDecaffInfo = "Decaff";

    inline static const double mMochaPrice = 2.7;
    inline static const std::string mMochaInfo = "Mocha";

    inline static const double mExtendedPrice = 5;
    inline static const std::string mExtendedInfo = "Extended_One";

    inline static const double mMilkPrice = 2.5;
    inline static const std::string mMilkInfo = "Milk";

    inline static const double mSojaMilkPrice = 15;
    inline static const std::string mSojaMilkInfo = "SojaMilk";

    inline static const double mSugarPrice = 1.5;
    inline static const std::string mSugarInfo = "Sugar";

    inline static const double mCreamPrice = 2;
    inline static const std::string mCreamInfo = "Cream";

protected:
    CoffeeInfo() = default;
};


#endif // !COFFEE_INFO_HPP
```

6.4 Ingredient.hpp

```cpp
/**
 * @file Ingredient.hpp
 * @brief Declares the decorator base class that augments an ICoffee.
 */
#ifndef INGREDIENT_HPP
#define  INGREDIENT_HPP

#include "Object.h"
#include "ICoffee.hpp"

class Ingredient : public ICoffee , public Object {
public:
    inline static const std::string ERROR_NULLPTR = "Error_Nullptr!";

    /**
     * @brief Forward cost request to the decorated coffee.
     * @return Accumulated coffee price.
     */
    virtual double GetCost() const override;

    /**
     * @brief Forward description request to the decorated coffee.
     * @return Aggregated description string.
     */
    virtual std::string GetDescription() const override;


    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(Ingredient& ind) = delete;
    Ingredient(Ingredient& ind) = delete;


protected:

    /**
     * @brief Construct a decorator around another coffee.
     * @param mCoffeeIngredient Coffee instance to wrap; must not be null.
     */
    Ingredient(ICoffee::Uptr mCoffeeIngredient);

    ICoffee::Uptr mDecorator;
};


#endif // !INGREDIENT_HPP
```

6.5 Ingredient.cpp

```cpp
/**
 * @file Ingredient.cpp
 * @brief Implements the decorator base class used by ingredient add-ons.
 */
#include "Ingredient.hpp"
#include <stdexcept>


Ingredient::Ingredient(ICoffee::Uptr mCoffeeIngredient)
{
    if (mCoffeeIngredient == nullptr) throw std::invalid_argument(ERROR_NULLPTR);

    mDecorator = move(mCoffeeIngredient);
}

double Ingredient::GetCost() const
{
    return mDecorator->GetCost();
}

std::string Ingredient::GetDescription() const
{
    return mDecorator->GetDescription();
}
```

6.6 CoffeePreparation.hpp

```cpp
/**
 * @file CoffeePreparation.hpp
 * @brief Declares a queue-based coffee preparation pipeline with output helpers.
 */
#ifndef COFFEE_PREPARATION_HPP
#define  COFFEE_PREPARATION_HPP

#include "ICoffee.hpp"
#include <deque>
#include <string>
#include <iostream>

class CoffeePreparation {
public:
    inline static const std::string ERROR_NULLPTR = "Error Nullptr!";
    inline static const std::string ERROR_BAD_OSTREAM = "Error Bad Ostream";
    inline static const std::string ERROR_NO_COFFE_IN_MACHINE = "Error No Coffe in the Machine!";

    CoffeePreparation() = default;

    /**
     * @brief Enqueue a coffee for preparation.
     * @param coffee Ownership of the coffee instance to queue.
     */
    void Prepare(ICoffee::Uptr coffee);

    /**
     * @brief Prints all coffees description and price to a stream.
     * @param ost Target output stream; must be valid.
     */
    void Display(std::ostream& ost) const;

    /**
     * @brief Remove and return the next finished coffee.
     * @return Unique pointer to the prepared coffee.
     */
    ICoffee::Uptr Finished();

    void operator=(CoffeePreparation & prep) = delete;
    CoffeePreparation(CoffeePreparation& prep) = delete;

private:
    std::deque<ICoffee::Uptr> mCoffeeQueue;
};


#endif // !COFFEE_PREPARATION_HPP
```

6.7 CoffeePreparation.cpp

```cpp
/**
 * @file CoffeePreparation.cpp
 * @brief Implements the coffee preparation queue with display and pickup helpers.
 */
#include "CoffeePreparation.hpp"

void CoffeePreparation::Prepare(ICoffee::Uptr coffee)
{
    if (coffee == nullptr) throw std::invalid_argument(ERROR_NULLPTR);

    mCoffeeQueue.push_back(move(coffee));
}

void CoffeePreparation::Display(std::ostream& ost) const
{
    if (ost.bad()) throw std::invalid_argument(ERROR_BAD_OSTREAM);
    if (mCoffeeQueue.empty()) throw std::runtime_error(ERROR_NO_COFFE_IN_MACHINE);


    for(const auto & coffee : mCoffeeQueue) {
        std::string description = coffee->GetDescription();

        // discard the last "," to fullfill the requirement
        // in the excersise
        if (!description.empty()) {
            *description.rbegin() = ' ';
        }
        ost << description;
        ost << coffee->GetCost() << " Euro" << std::endl;
    }


}

ICoffee::Uptr CoffeePreparation::Finished()
{
    if (mCoffeeQueue.empty()) throw std::runtime_error(ERROR_NO_COFFE_IN_MACHINE);

    ICoffee::Uptr retCoffee = move(mCoffeeQueue.front());
    mCoffeeQueue.pop_front();

    return move(retCoffee);
}
```

6.8 SojaMilk.hpp

```cpp
/**
 * @file SojaMilk.hpp
 * @brief Declares the soja milk ingredient decorator for coffee orders.
 */
#ifndef SOJA_MILK_HPP
#define  SOJA_MILK_HPP

#include <string>

#include "Object.h"
#include "Ingredient.hpp"

class SojaMilk : public Ingredient {
public:

    /**
     * @brief Wrap a coffee with soja milk.
     * @param cof Coffee to decorate.
     */
    SojaMilk(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}

    /**
     * @brief Return price including soja milk surcharge.
     */
    virtual double GetCost() const override;

    /**
     * @brief Append soja milk label to description.
     */
    virtual std::string GetDescription() const override;

    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(SojaMilk& ind) = delete;
    SojaMilk(SojaMilk& ind) = delete;
};

#endif // !SOJA_MILK_HPP
```

6.9 SojaMilk.cpp

```cpp
/**
 * @file SojaMilk.cpp
 * @brief Implements the soja milk ingredient decorator behavior.
 */
#include "SojaMilk.hpp"
#include "CoffeeInfo.hpp"

double SojaMilk::GetCost() const
{
    return CoffeeInfo::mSojaMilkPrice + Ingredient::GetCost();
}

std::string SojaMilk::GetDescription() const
{
    return Ingredient::GetDescription() + "␣" + CoffeeInfo::mSojaMilkInfo + ",";
}
```

6.10 Milk.hpp

```cpp
/**
 * @file Milk.hpp
 * @brief Declares the milk ingredient decorator for coffee orders.
 */
#ifndef MILK_HPP
#define  MILK_HPP

#include <string>

#include "Object.h"
#include "Ingredient.hpp"

class Milk : public Ingredient {
public:

    /**
     * @brief Wrap a coffee with milk.
     * @param cof Coffee to decorate.
     */
    Milk(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}

    /**
     * @brief Return price including milk surcharge.
     */
    virtual double GetCost() const override;

    /**
     * @brief Append milk label to description.
     */
    virtual std::string GetDescription() const override;

    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(Milk& ind) = delete;
    Milk(Milk& ind) = delete;
};

#endif // !MILK_HPP
```

6.11 Milk.cpp

```cpp
/**
 * @file Milk.cpp
 * @brief Implements the milk ingredient decorator behavior.
 */
#include "Milk.hpp"
#include "CoffeeInfo.hpp"

double Milk::GetCost() const
{
    return CoffeeInfo::mMilkPrice + Ingredient::GetCost();
}

std::string Milk::GetDescription() const
{
    return Ingredient::GetDescription() + "␣" + CoffeeInfo::mMilkInfo + ",";
}
```

6.12 Sugar.hpp

```cpp
/**
 * @file Sugar.hpp
 * @brief Declares the sugar ingredient decorator for coffee orders.
 */
#ifndef SUGAR_HPP
#define  SUGAR_HPP

#include <string>

#include "Object.h"
#include "Ingredient.hpp"

class Sugar : public Ingredient {
public:

    /**
     * @brief Wrap a coffee with sugar.
     * @param cof Coffee to decorate.
     */
    Sugar(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}

    /**
     * @brief Return price including sugar surcharge.
     */
    virtual double GetCost() const override;

    /**
     * @brief Append sugar label to description.
     */
    virtual std::string GetDescription() const override;

    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(Sugar& ind) = delete;
    Sugar(Sugar& ind) = delete;
};

#endif // !SUGAR_HPP
```

6.13 Sugar.cpp

```cpp
/**
 * @file Sugar.cpp
 * @brief Implements the sugar ingredient decorator behavior.
 */
#include "Sugar.hpp"
#include "CoffeeInfo.hpp"

double Sugar::GetCost() const
{
    return CoffeeInfo::mSugarPrice + Ingredient::GetCost();
}

std::string Sugar::GetDescription() const
{
    return Ingredient::GetDescription() + "␣" + CoffeeInfo::mSugarInfo + ",";
}
```

6.14 Cream.hpp

```cpp
/**
 * @file Cream.hpp
 * @brief Declares the cream ingredient decorator for coffee orders.
 */
#ifndef CREAM_HPP
#define  CREAM_HPP

#include <string>

#include "Object.h"
#include "Ingredient.hpp"

class Cream : public Ingredient {
public:

    using Uptr = std::unique_ptr<Cream>;


    /**
     * @brief Wrap a coffee with cream.
     * @param cof Coffee to decorate.
     */
    Cream(ICoffee::Uptr cof) : Ingredient{ move(cof) } {}

    /**
     * @brief Return price including cream surcharge.
     */
    virtual double GetCost() const override;

    /**
     * @brief Append cream label to description.
     */
    virtual std::string GetDescription() const override;

    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(Cream& ind) = delete;
    Cream(Cream& ind) = delete;

};

#endif // !CREAM_HPP
```

6.15 Cream.cpp

```cpp
/**
 * @file Cream.cpp
 * @brief Implements the cream ingredient decorator behavior.
 */
#include "Cream.hpp"
#include "CoffeeInfo.hpp"

double Cream::GetCost() const
{
    return CoffeeInfo::mCreamPrice + Ingredient::GetCost();
}

std::string Cream::GetDescription() const
{
    return Ingredient::GetDescription() + "␣" + CoffeeInfo::mCreamInfo + ",";
}
```

6.16 ExtendedOne.hpp

```cpp
/**
 * @file ExtendedOne.hpp
 * @brief Declares the extended coffee variant implementation of ICoffee.
 */
#ifndef EXTENDED_ONE_HPP
#define  EXTENDED_ONE_HPP

#include "Object.h"
#include "ICoffee.hpp"


class ExtendedOne : public ICoffee, public Object {
public:

    using Uptr = std::unique_ptr<ExtendedOne>;

    ExtendedOne() = default;

    /**
     * @brief Return the price of the extended variant.
     */
    virtual double GetCost() const override;

    /**
     * @brief Provide the extended variant description label.
     */
    virtual std::string GetDescription() const override;

    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(ExtendedOne& ind) = delete;
    ExtendedOne(ExtendedOne& ind) = delete;
};

#endif // !EXTENDED_ONE_HPP
```

6.17 ExtendedOne.cpp

```cpp
/**
 * @file ExtendedOne.cpp
 * @brief Implements the extended coffee variant pricing and description.
 */
#include "ExtendedOne.hpp"
#include "CoffeeInfo.hpp"


double ExtendedOne::GetCost() const
{
    return CoffeeInfo::mExtendedPrice;
}

std::string ExtendedOne::GetDescription() const
{
    return CoffeeInfo::mExtendedInfo + ":";
}
```

6.18 Espresso.hpp

```cpp
/**
 * @file Espresso.hpp
 * @brief Declares the espresso coffee implementation of ICoffee.
 */
#ifndef ESPRESSO_HPP
#define  ESPRESSO_HPP

#include "Object.h"
#include "ICoffee.hpp"


class Espresso : public ICoffee , public Object {
public:

    using Uptr = std::unique_ptr<Espresso>;

    Espresso() = default;

    /**
     * @brief Return the price of an espresso.
     */
    virtual double GetCost() const override;

    /**
     * @brief Provide the espresso description label.
     */
    virtual std::string GetDescription() const override;


    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(Espresso& ind) = delete;
    Espresso(Espresso& ind) = delete;
};

#endif // !ESPRESSO_HPP
```

6.19 Espresso.cpp

```cpp
/**
 * @file Espresso.cpp
 * @brief Implements the espresso coffee pricing and description.
 */
#include "Espresso.hpp"
#include "CoffeeInfo.hpp"


double Espresso::GetCost() const
{
    return CoffeeInfo::mEspressoPrice;
}

std::string Espresso::GetDescription() const
{
    return CoffeeInfo::mEspressoInfo + ":";
}
```

6.20 Decaff.hpp

```cpp
/**
 * @file Decaff.hpp
 * @brief Declares the decaffeinated coffee implementation of ICoffee.
 */
#ifndef DECAFF_HPP
#define  DECAFF_HPP

#include "Object.h"
#include "ICoffee.hpp"

class Decaff : public ICoffee, public Object {
public:

    using Uptr = std::unique_ptr<Decaff>;

    Decaff() = default;


    /**
     * @brief Return the price of a decaffeinated coffee.
     */
    virtual double GetCost() const override;

    /**
     * @brief Provide the decaff description label.
     */
    virtual std::string GetDescription() const override;

    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(Decaff& ind) = delete;
    Decaff(Decaff& ind) = delete;

};

#endif // !DECAFF_HPP
```

6.21 Decaff.cpp

```cpp
/**
 * @file Decaff.cpp
 * @brief Implements the decaffeinated coffee pricing and description.
 */
#include "Decaff.hpp"
#include "CoffeeInfo.hpp"

double Decaff::GetCost() const
{
    return CoffeeInfo::mDecaffPrice;
}

std::string Decaff::GetDescription() const
{
    return CoffeeInfo::mDecaffInfo + ":";
}
```

6.22 Mocha.hpp

```cpp
/**
 * @file Mocha.hpp
 * @brief Declares the mocha coffee implementation of ICoffee.
 */
#ifndef MOCHA_HPP
#define  MOCHA_HPP

#include "Object.h"
#include "ICoffee.hpp"


class Mocha : public ICoffee, public Object {
public:

    Mocha() = default;


    /**
     * @brief Return the price of a mocha.
     */
    virtual double GetCost() const override;

    /**
     * @brief Provide the mocha description label.
     */
    virtual std::string GetDescription() const override;

    // explicitly delete Assign Op and Copy Ctor to prevent untestet behaviour
    void operator=(Mocha& ind) = delete;
    Mocha(Mocha& ind) = delete;
};

#endif // !MOCHA_HPP
```

6.23 Mocha.cpp

```cpp
/**
 * @file Mocha.cpp
 * @brief Implements the mocha coffee pricing and description.
 */
#include "Mocha.hpp"
#include "CoffeeInfo.hpp"

double Mocha::GetCost() const
{
    return CoffeeInfo::mMochaPrice;
}

std::string Mocha::GetDescription() const
{
    return CoffeeInfo::mMochaInfo + ":";
}
```

6.24 main.cpp

```cpp
/**
 * @file main.cpp
 * @brief Runs sample preparations and tests for the coffee machine decorators.
 */
#include "vld.h"
#include "Mocha.hpp"
#include "ExtendedOne.hpp"
#include "Decaff.hpp"
#include "Espresso.hpp"
#include "Milk.hpp"
#include "Sugar.hpp"
#include "SojaMilk.hpp"
#include "Cream.hpp"
#include "CoffeePreparation.hpp"
#include "Test.hpp"
#include "CoffeeInfo.hpp"

#include <memory>
#include <iostream>
#include <cassert>
#include <sstream>
#include <fstream>

using namespace std;

static bool TestCoffeeIngridient(std::ostream& ost,ICoffee::Uptr cof, const std::string& description, const double price);
static bool TestCoffeeIngridientException(std::ostream& ost);
static bool TestCoffeePreparation(std::ostream& ost);


#define WriteOutputFile   true

int main()
{
    bool TestOK = true;
    ofstream output{ "Testoutput.txt" };

    if (!output.is_open()) {
        cerr << "Konnte Testoutput.txt nicht oeffnen" << TestCaseFail;
        return 1;
    }

    try {

        cout << TestStart;
        cout << "Test Espresso" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<Espresso>(), CoffeeInfo::mEspressoInfo + ":", CoffeeInfo::mEspressoPrice);
        cout << TestEnd;

        cout << TestStart;
        cout << "Test Mocha" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<Mocha>(), CoffeeInfo::mMochaInfo + ":", CoffeeInfo::mMochaPrice);
        cout << TestEnd;

        cout << TestStart;
        cout << "Test Decaff" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<Decaff>(), CoffeeInfo::mDecaffInfo + ":", CoffeeInfo::mDecaffPrice);
        cout << TestEnd;

        cout << TestStart;
        cout << "Test Extended One" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<ExtendedOne>(), CoffeeInfo::mExtendedInfo + ":", CoffeeInfo::mExtendedPrice);
        cout << TestEnd;

        cout << TestStart;
        cout << "Test Espresso with Milk" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<Milk>(make_unique<Espresso>()),
            CoffeeInfo::mEspressoInfo + ": " + CoffeeInfo::mMilkInfo + ",",
            CoffeeInfo::mEspressoPrice + CoffeeInfo::mMilkPrice);
        cout << TestEnd;

        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>(
103               make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
104                  make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
105                     make_unique<Mocha>())))))))))),
106               CoffeeInfo::mMochaInfo + ":_" + 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
149            output << TestStart;
150            output << "Test_Extended_One_with_SojaMilk" << endl << endl;
151            TestCoffeeIngridient(output, make_unique<SojaMilk>(make_unique<ExtendedOne>()),
152                CoffeeInfo::mExtendedInfo + ":_" + CoffeeInfo::mSojaMilkInfo + ",",
153                CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice);
154            output << TestEnd;
155
156            output << TestStart;
157            output << "Test_Mocha_with_Sugar" << endl << endl;
158            TestCoffeeIngridient(output, make_unique<Sugar>(make_unique<Mocha>()),
159                CoffeeInfo::mMochaInfo + ":_" + CoffeeInfo::mSugarInfo + ",",
160                CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice);
161            output << TestEnd;
162
163            output << TestStart;
164            output << "Test_Decaff_with_Cream" << endl << endl;
165            TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Decaff>()),
166                CoffeeInfo::mDecaffInfo + ":_" + CoffeeInfo::mCreamInfo + ",",
167                CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice);
168            output << TestEnd;
169
170            output << TestStart;
171            output << "Test_Decaff_with_Cream_and_Cream" << endl << endl;
172            TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Cream>(make_unique<Decaff>())),
173                CoffeeInfo::mDecaffInfo + ":_" + CoffeeInfo::mCreamInfo + ",_" + CoffeeInfo::mCreamInfo + ",",
174                CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice + CoffeeInfo::mCreamPrice);
175            output << TestEnd;
176
177            output << TestStart;
178            output << "Test_Mocha_alla_Diabetes" << endl << endl;
179            TestCoffeeIngridient(output, 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
234    bool TestCoffeeIngridient(std::ostream & ost,ICoffee::Uptr cof, const std::string & description, const double price)
235    {
236        assert(cof != nullptr);
237        assert(ost.good());
238
239        std::string error_msg;
240        bool TestOK = true;
241
242        try {
243            TestOK = TestOK && check_dump(ost, "Test ICoffee Description", cof->GetDescription(), description);
244            TestOK = TestOK && check_dump(ost, "Test ICoffee Price", cof->GetCost(), price);
245        }
246        catch (const string& err) {
247            error_msg = err;
248        }
249        catch (bad_alloc const& error) {
250            error_msg = error.what();
251        }
252        catch (const exception& err) {
253            error_msg = err.what();
254        }
255        catch (...) {
256            error_msg = "Unhandelt Exception";
257        }
258
259        TestOK = TestOK && check_dump(ost, "Test for Exception in Testcase", true, error_msg.empty());
260
261        return TestOK;
262    }
263
264    bool TestCoffeeIngridientException(std::ostream& ost)
265    {
266        assert(ost.good());
267
268        std::string error_msg;
269        bool TestOK = true;
270
271        try {
272            ICoffee::Uptr cof = make_unique<Milk>(nullptr);
273        }
274        catch (const string& err) {
275            error_msg = err;
276        }
277        catch (bad_alloc const& error) {
278            error_msg = error.what();
279        }
280        catch (const exception& err) {
281            error_msg = err.what();
282        }
283        catch (...) {
284            error_msg = "Unhandelt Exception";
285        }
286
287        TestOK = TestOK && check_dump(ost, "Test for Exception in Ingedient CTOR", Ingredient::ERROR_NULLPTR, error_msg);
288
289        return TestOK;
290    }
291
292    bool TestCoffeePreparation(std::ostream& ost) {
293
294        assert(ost.good());
295
296
297        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
310        CoffeeMachine.Display(actual_output);
311
312        expected_output << CoffeeInfo::mEspressoInfo + ": " + CoffeeInfo::mMilkInfo + " " << CoffeeInfo::mEspressoPrice + CoffeeInfo::mMi
313        expected_output << CoffeeInfo::mExtendedInfo + ": " + CoffeeInfo::mSojaMilkInfo + " " << CoffeeInfo::mExtendedPrice + CoffeeInfo:
314
315        TestOK = TestOK && check_dump(ost, "Test CoffeePreparation Display 1", actual_output.str(), expected_output.str());
316
317        ICoffee::Uptr cof = CoffeeMachine.Finished();
318
319        actual_output.str("");
320        expected_output.str("");
321
322        CoffeeMachine.Display(actual_output);
323
324        expected_output << CoffeeInfo::mExtendedInfo + ": " + CoffeeInfo::mSojaMilkInfo + " " << CoffeeInfo::mExtendedPrice + CoffeeInfo:
325
326        TestOK = TestOK && check_dump(ost, "Test CoffeePreparation Display 2", actual_output.str(), expected_output.str());
327
328        cof = CoffeeMachine.Finished();
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
375        CoffeeMachine.Display(ost);
376     }
377     catch (const string& err) {
378        error_msg = err;
379     }
380     catch (bad_alloc const& error) {
381        error_msg = error.what();
382     }
383     catch (const exception& err) {
384        error_msg = err.what();
385     }
386     catch (...) {
387        error_msg = "Unhandelt Exception";
388     }
389
390     TestOK = TestOK && check_dump(ost, "Test Exception Queue is Empty Display", CoffeePreparation::ERROR_NO_COFFE_IN_MACHINE, error_msg)
391
392     try {
393
394        CoffeePreparation CoffeeMachine;
395
396        CoffeeMachine.Finished();
397     }
398     catch (const string& err) {
399        error_msg = err;
400     }
401     catch (bad_alloc const& error) {
402        error_msg = error.what();
403     }
404     catch (const exception& err) {
405        error_msg = err.what();
406     }
407     catch (...) {
408        error_msg = "Unhandelt Exception";
409     }
410
411     TestOK = TestOK && check_dump(ost, "Test Exception Queue is Empty Finished", CoffeePreparation::ERROR_NO_COFFE_IN_MACHINE, error_msg
412
413
414     return TestOK;
415 }
```

6.25 Test.hpp

```cpp
/******************************************************//**
 * \file   Test.hpp
 * \brief  File that provides a Test Function with a formated output
 *
 * \author Simon
 * \date   April 2025
 ******************************************************/
#ifndef TEST_HPP
#define TEST_HPP

#include <string>
#include <iostream>
#include <vector>
#include <list>
#include <queue>
#include <forward_list>

#define ON 1
#define OFF 0
#define COLOR_OUTPUT OFF

// Definitions of colors in order to change the color of the output stream.
const std::string colorRed = "\x1B[31m";
const std::string colorGreen = "\x1B[32m";
const std::string colorWhite = "\x1B[37m";

inline std::ostream& RED(std::ostream& ost) {
    if (ost.good()) {
        ost << colorRed;
    }
    return ost;
}
inline std::ostream& GREEN(std::ostream& ost) {
    if (ost.good()) {
        ost << colorGreen;
    }
    return ost;
}
inline std::ostream& WHITE(std::ostream& ost) {
    if (ost.good()) {
        ost << colorWhite;
    }
    return ost;
}

inline std::ostream& TestStart(std::ostream& ost) {
    if (ost.good()) {
        ost << std::endl;
        ost << "*****************************************" << std::endl;
        ost << "_____TESTCASE_START_____" << std::endl;
        ost << "*****************************************" << std::endl;
        ost << std::endl;
    }
    return ost;
}

inline std::ostream& TestEnd(std::ostream& ost) {
    if (ost.good()) {
        ost << std::endl;
        ost << "*****************************************" << std::endl;
        ost << std::endl;
    }
    return ost;
}

inline std::ostream& TestCaseOK(std::ostream& ost) {

#if COLOR_OUTPUT
    if (ost.good()) {
        ost << colorGreen << "TEST_OK!!" << colorWhite << std::endl;
    }
#else
```

```cpp
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      }
88  #else
89      if (ost.good()) {
90          ost << "TEST_FAILED_!!" << std::endl;
91
92      }
93  #endif // COLOR_OUTPUT
94
95      return ost;
96  }
97
98  /**
99    * \brief function that reports if the testcase was successful.
100   *
101   * \param testcase String that indicates the testcase
102   * \param succsessful true -> reports to cout test OK
103   * \param succsessful false -> reports test failed
104   */
105 template <typename T>
106 bool check_dump(std::ostream& ostr, const std::string& testcase, const T& expected, const T& result) {
107     if (ostr.good()) {
108 #if COLOR_OUTPUT
109         if (expected == result) {
110             ostr << testcase << std::endl <<  colorGreen << "[Test_OK]_" << colorWhite <<"Result:_(Expected:_" << std::boolalpha << expect
            << std::noboolalpha << std::endl << std::endl;
111         }
112         else {
113             ostr << testcase << std::endl << colorRed << "[Test_FAILED]_" << colorWhite << "Result:_(Expected:_" << std::boolalpha << expe
            << std::noboolalpha << std::endl << std::endl;
114         }
115 #else
116         if (expected == result) {
117             ostr << testcase << std::endl << "[Test_OK]_"  << "Result:_(Expected:_" << std::boolalpha << expected << "_==" << "_Result:_
118         }
119         else {
120             ostr << testcase << std::endl  << "[Test_FAILED]_"  << "Result:_(Expected:_" << std::boolalpha << expected << "_!=" << "_Resul
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
```