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:

```
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!

Systemdokumentation
Projekt Filesystem

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 12. Dezember 2025

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

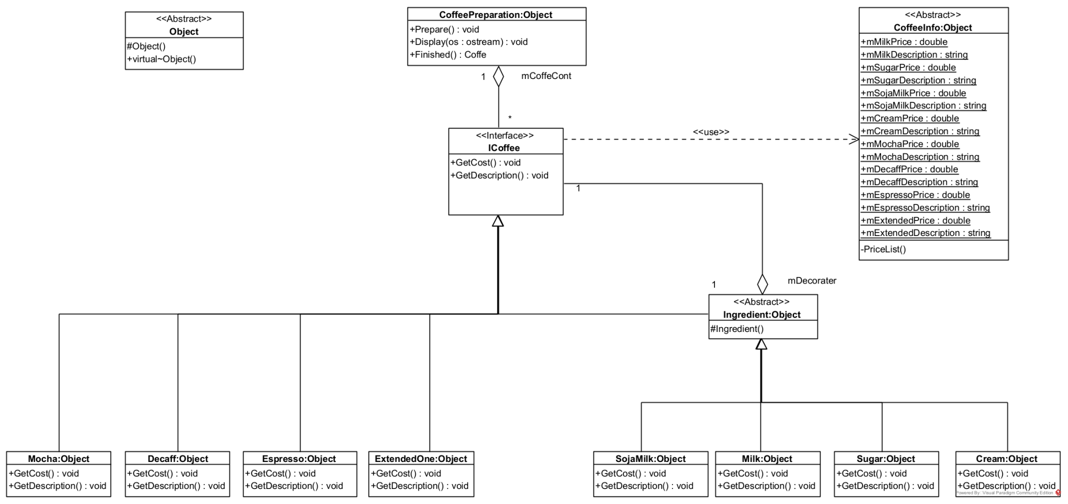
1.3 Aufwand

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

2 Anforderungsdefinition (Systemspezifikation)

3 Systementwurf

3.1 Klassendiagramm



3.2 Designentscheidungen

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
******************************************
```

```

Test Decaff

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

Test ICoffee Price
[Test OK] Result: (Expected: 2.8 == Result: 2.8)

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


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


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

Test Extended One

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

Test ICoffee Price
[Test OK] Result: (Expected: 5 == Result: 5)

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


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


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

Test Espresso with Milk

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

```

```
Test ICoffee Price
[Test OK] Result: (Expected: 5.5 == Result: 5.5)

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


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


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

Test Extended One with SojaMilk

Test ICoffee Description
[Test OK] Result: (Expected: Extended One: SojaMilk, == Result: Extended
    ↪ One: SojaMilk,)

Test ICoffee Price
[Test OK] Result: (Expected: 20 == Result: 20)

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


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


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

Test Mocha with Sugar

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

Test ICoffee Price
[Test OK] Result: (Expected: 4.2 == Result: 4.2)

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

```


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


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

Test Decaff with Cream

Test ICoffee Description
[Test OK] Result: (Expected: Decaff: Cream, == Result: Decaff: Cream,)

Test ICoffee Price
[Test OK] Result: (Expected: 4.8 == Result: 4.8)

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


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


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

Test Decaff with Cream and Cream

Test ICoffee Description
[Test OK] Result: (Expected: Decaff: Cream, Cream, == Result: Decaff: Cream, Cream,)

Test ICoffee Price
[Test OK] Result: (Expected: 6.8 == Result: 6.8)

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


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


```

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

Test Mocha alla Diabetes

Test ICoffee Description
[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,)

Test ICoffee Price
[Test OK] Result: (Expected: 16.2 == Result: 16.2)

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


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

Test CoffeePreparation Display 1
[Test OK] Result: (Expected: Espresso: Milk 5.5 Euro
 == Result: Espresso: Milk 5.5 Euro
)

Test CoffeePreparation Display 2
[Test OK] Result: (Expected: Extended One: SojaMilk 20 Euro
 == Result: Extended One: SojaMilk 20 Euro
)

Test CoffeePreparation Display 3
[Test OK] Result: (Expected: Mocha: Sugar 4.2 Euro
 == Result: Mocha: Sugar 4.2 Euro
)

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

Test Exception Bad Ostream in CoffeePreparation
[Test OK] Result: (Expected: Error Bad Ostream == Result: Error Bad Ostream
   ↪ )

Test for Exception in Ingedient CTOR
[Test OK] Result: (Expected: Error Nullptr! == Result: Error Nullptr!)
```

```

TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```
/**
 * @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(){};
public:
   /**
    * @brief Virtual destructor to allow safe polymorphic deletion.
    */
   virtual ~Object(){}
};

#endif // OBJECT_H
```

6.2 ICoffee.hpp

```
/**
 * @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() = 0;

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

   virtual ~ICoffee() = default;
};




#endif // !ICOFFEE_HPP
```

6.3 CoffeeInfo.hpp

```
/**
 * @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";

private:
   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() override;

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

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 CoffeePreparation.hpp

```
/**
 * @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 <queue>
#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";

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

   /**
    * @brief Print the next coffee description and price to a stream.
    * @param ost Target output stream; must be valid.
    */
   void Display(std::ostream& ost);

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

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


#endif // !COFFEE_PREPARATION_HPP
```

6.6 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(move(coffee));
}

void CoffeePreparation::Display(std::ostream& ost)
{
   if (ost.bad()) throw std::invalid_argument(ERROR_BAD_OSTREAM);

   std::string description = mCoffeeQueue.front()->GetDescription();

   // discard the last "," to fullfill the requirement
   // in the excersise
   *description.rbegin() = '␣';

   ost << description;
   ost << mCoffeeQueue.front()->GetCost() << "␣Euro" << std::endl;
}

ICoffee::Uptr CoffeePreparation::Finished()
{
   ICoffee::Uptr retCoffee = move(mCoffeeQueue.front());
   mCoffeeQueue.pop();

   return move(retCoffee);
}
```

6.7 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() override;

   /**
    * @brief Append soja milk label to description.
```

```
     */
    virtual std::string GetDescription() override;

};

#endif // !SOJA_MILK_HPP
```

6.8 SojaMilk.cpp

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

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

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

6.9 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() override;

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

};

#endif // !MILK_HPP
```

6.10 Milk.cpp

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

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

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

6.11 Sugar.hpp

```
/**
 * @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() override;

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

};

#endif // !SUGAR_HPP
```

6.12 Sugar.cpp

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

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

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

6.13 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:

   /**
    * @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() override;

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

};

#endif // !CREAM_HPP
```

6.14 Cream.cpp

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

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

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

6.15 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 {

   using Sptr = std::shared_ptr<ExtendedOne>;

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

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

};

#endif // !EXTENDED_ONE_HPP
```

6.16 ExtendedOne.cpp

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


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

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

6.17 Espresso.hpp

```
/**
 * @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 {

   using Sptr = std::shared_ptr<Espresso>;

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

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

};

#endif // !ESPRESSO_HPP
```

6.18 Espresso.cpp

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


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

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

6.19 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 {

   using Sptr = std::shared_ptr<Decaff>;

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

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

};

#endif // !DECAFF_HPP
```

6.20 Decaff.cpp

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

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

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

6.21 Mocha.hpp

```
/**
 * @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 {

   using Sptr = std::shared_ptr<Mocha>;

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

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

};

#endif // !MOCHA_HPP
```

6.22 Mocha.cpp

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

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

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

6.23 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 {

      ICoffee::Uptr Coff{ std::make_unique<Cream>(std::make_unique<Sugar>(std::make_unique<Milk>(std::make_unique<Espresso>()))) };

      CoffeePreparation CoffeeMachine;

      CoffeeMachine.Prepare(move(Coff));

      CoffeeMachine.Display(std::cout);

      Coff = CoffeeMachine.Finished();

      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;
        cout << "Test␣Extended␣One␣with␣SojaMilk" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<SojaMilk>(make_unique<ExtendedOne>()),
           CoffeeInfo::mExtendedInfo + ":␣" + CoffeeInfo::mSojaMilkInfo + ",",
           CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice);
        cout << TestEnd;

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

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

        cout << TestStart;
        cout << "Test␣Decaff␣with␣Cream␣and␣Cream" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<Cream>(make_unique<Cream>(make_unique<Decaff>())),
           CoffeeInfo::mDecaffInfo + ":␣" + CoffeeInfo::mCreamInfo + ",␣" + CoffeeInfo::mCreamInfo + ",",
           CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice + CoffeeInfo::mCreamPrice);
        cout << TestEnd;

        cout << TestStart;
        cout << "Test␣Mocha␣alla␣Diabetes" << endl << endl;
        TestCoffeeIngridient(std::cout, make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
           make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
              make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
                 make_unique<Mocha>())))))))))),
           CoffeeInfo::mMochaInfo + ":␣" + CoffeeInfo::mSugarInfo + ",␣"
           + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
           + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
           + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
           + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",",
           CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice * 9);
        cout << TestEnd;


        TestCoffeePreparation(std::cout);

        TestCoffeeIngridientException(std::cout);


        if (WriteOutputFile) {


           ICoffee::Uptr Coff{ std::make_unique<Cream>(std::make_unique<Sugar>(std::make_unique<Milk>(std::make_unique<Espresso>()))) };

           CoffeePreparation CoffeeMachine;

           CoffeeMachine.Prepare(move(Coff));

           CoffeeMachine.Display(std::cout);

           Coff = CoffeeMachine.Finished();

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

```

```
      output << TestStart;
      output << "Test␣Mocha" << endl << endl;
      TestCoffeeIngridient(output, make_unique<Mocha>(), CoffeeInfo::mMochaInfo + ":", CoffeeInfo::mMochaPrice);
      output << TestEnd;

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

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

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

      output << TestStart;
      output << "Test␣Extended␣One␣with␣SojaMilk" << endl << endl;
      TestCoffeeIngridient(output, make_unique<SojaMilk>(make_unique<ExtendedOne>()),
         CoffeeInfo::mExtendedInfo + ":␣" + CoffeeInfo::mSojaMilkInfo + ",",
         CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice);
      output << TestEnd;

      output << TestStart;
      output << "Test␣Mocha␣with␣Sugar" << endl << endl;
      TestCoffeeIngridient(output, make_unique<Sugar>(make_unique<Mocha>()),
         CoffeeInfo::mMochaInfo + ":␣" + CoffeeInfo::mSugarInfo + ",",
         CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice);
      output << TestEnd;

      output << TestStart;
      output << "Test␣Decaff␣with␣Cream" << endl << endl;
      TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Decaff>()),
         CoffeeInfo::mDecaffInfo + ":␣" + CoffeeInfo::mCreamInfo + ",",
         CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice);
      output << TestEnd;

      output << TestStart;
      output << "Test␣Decaff␣with␣Cream␣and␣Cream" << endl << endl;
      TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Cream>(make_unique<Decaff>())),
         CoffeeInfo::mDecaffInfo + ":␣" + CoffeeInfo::mCreamInfo + ",␣" + CoffeeInfo::mCreamInfo + ",",
         CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice + CoffeeInfo::mCreamPrice);
      output << TestEnd;

      output << TestStart;
      output << "Test␣Mocha␣alla␣Diabetes" << endl << endl;
      TestCoffeeIngridient(output, make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
         make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
            make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
               make_unique<Mocha>())))))))))),
         CoffeeInfo::mMochaInfo + ":␣" + CoffeeInfo::mSugarInfo + ",␣"
         + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
         + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
         + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
         + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",",
         CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice * 9);
      output << TestEnd;


      TestCoffeePreparation(output);

      TestCoffeeIngridientException(output);




      if (TestOK) {
         output << TestCaseOK;
      }
```

```
         else {
            output << TestCaseFail;
         }

         output.close();
      }

      if (TestOK) {
         cout << TestCaseOK;
      }
      else {
         cout << TestCaseFail;
      }
   }
   catch (const string& err) {
      cerr << err << TestCaseFail;
   }
   catch (bad_alloc const& error) {
      cerr << error.what() << TestCaseFail;
   }
   catch (const exception& err) {
      cerr << err.what() << TestCaseFail;
   }
   catch (...) {
      cerr << "Unhandelt␣Exception" << TestCaseFail;
   }

   if (output.is_open()) output.close();

   return 0;


}

bool TestCoffeeIngridient(std::ostream & ost,ICoffee::Uptr cof, const std::string & description, const double price)
{
   assert(cof != nullptr);
   assert(ost.good());

   std::string error_msg;
   bool TestOK = true;

   try {
      TestOK = TestOK && check_dump(ost, "Test␣ICoffee␣Description", cof->GetDescription(), description);
      TestOK = TestOK && check_dump(ost, "Test␣ICoffee␣Price", cof->GetCost(), price);
   }
   catch (const string& err) {
      error_msg = err;
   }
   catch (bad_alloc const& error) {
      error_msg = error.what();
   }
   catch (const exception& err) {
      error_msg = err.what();
   }
   catch (...) {
      error_msg = "Unhandelt␣Exception";
   }

   TestOK = TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Testcase", true, error_msg.empty());


   return TestOK;
}

bool TestCoffeeIngridientException(std::ostream& ost)
{
   assert(ost.good());

   std::string error_msg;
   bool TestOK = true;

   try {
      ICoffee::Uptr cof = make_unique<Milk>(nullptr);
   }
   catch (const string& err) {
      error_msg = err;
   }
   catch (bad_alloc const& error) {
      error_msg = error.what();
   }
   catch (const exception& err) {
      error_msg = err.what();
   }
   catch (...) {
      error_msg = "Unhandelt␣Exception";
   }

   TestOK = TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Ingedient␣CTOR", Ingredient::ERROR_NULLPTR, error_msg);

   return TestOK;
}

bool TestCoffeePreparation(std::ostream& ost) {

   assert(ost.good());


   std::string error_msg;
   bool TestOK = true;

   try {
      CoffeePreparation CoffeeMachine;

      CoffeeMachine.Prepare(make_unique<Milk>(make_unique<Espresso>()));
      CoffeeMachine.Prepare(make_unique<SojaMilk>(make_unique<ExtendedOne>()));
      CoffeeMachine.Prepare(make_unique<Sugar>(make_unique<Mocha>()));

      stringstream expected_output;
      stringstream actual_output;


      CoffeeMachine.Display(actual_output);

      expected_output << CoffeeInfo::mEspressoInfo + ":␣" + CoffeeInfo::mMilkInfo + "␣" << CoffeeInfo::mEspressoPrice + CoffeeInfo::mMi

      TestOK = TestOK && check_dump(ost, "Test␣CoffeePreparation␣Display␣1", actual_output.str(), expected_output.str());

      ICoffee::Uptr cof = CoffeeMachine.Finished();

      actual_output.str("");
      expected_output.str("");

      CoffeeMachine.Display(actual_output);

      expected_output << CoffeeInfo::mExtendedInfo + ":␣" + CoffeeInfo::mSojaMilkInfo + "␣" << CoffeeInfo::mExtendedPrice + CoffeeInfo:

      TestOK = TestOK && check_dump(ost, "Test␣CoffeePreparation␣Display␣2", actual_output.str(), expected_output.str());

      cof = CoffeeMachine.Finished();

      actual_output.str("");
      expected_output.str("");

      CoffeeMachine.Display(actual_output);

      expected_output << CoffeeInfo::mMochaInfo + ":␣" + CoffeeInfo::mSugarInfo + "␣" << CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPr

      TestOK = TestOK && check_dump(ost, "Test␣CoffeePreparation␣Display␣3", actual_output.str(), expected_output.str());

      cof = CoffeeMachine.Finished();

   }
   catch (const string& err) {
      error_msg = err;
   }
   catch (bad_alloc const& error) {
      error_msg = error.what();
   }
   catch (const exception& err) {
      error_msg = err.what();
   }
   catch (...) {
      error_msg = "Unhandelt␣Exception";
   }

   TestOK = TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Testcase", true, error_msg.empty());

   try {

      CoffeePreparation CoffeeMachine;

      stringstream badstream;

      badstream.setstate(ios::badbit);

      CoffeeMachine.Display(badstream);
   }
   catch (const string& err) {
      error_msg = err;
   }
   catch (bad_alloc const& error) {
      error_msg = error.what();
   }
   catch (const exception& err) {
      error_msg = err.what();
   }
   catch (...) {
      error_msg = "Unhandelt␣Exception";
   }

   TestOK = TestOK && check_dump(ost, "Test␣Exception␣Bad␣Ostream␣in␣CoffeePreparation", CoffeePreparation::ERROR_BAD_OSTREAM, error_ms


   return TestOK;
}
```

6.24 Test.hpp

```
/******************************************************************//**
 * \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
```

```
   if (ost.good()) {
      ost <<  "TEST␣OK!!" <<  std::endl;
   }
#endif // COLOR_OUTPUT

   return ost;
}

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

#if COLOR_OUTPUT
   if (ost.good()) {
      ost << colorRed << "TEST␣FAILED␣!!" << colorWhite << std::endl;

   }
#else
   if (ost.good()) {
      ost << "TEST␣FAILED␣!!" << std::endl;

   }
#endif // COLOR_OUTPUT

   return ost;
}

/**
   * \brief function that reports if the testcase was successful.
   *
   * \param testcase String that indicates the testcase
   * \param succsessful true -> reports to cout test OK
   * \param succsessful false -> reports test failed
   */
template <typename T>
bool check_dump(std::ostream& ostr, const std::string& testcase, const T& expected, const T& result) {
   if (ostr.good()) {
#if COLOR_OUTPUT
      if (expected == result) {
         ostr << testcase << std::endl <<  colorGreen << "[Test␣OK]␣" << colorWhite <<"Result:␣(Expected:␣" << std::boolalpha << expect
<< std::noboolalpha << std::endl << std::endl;
      }
      else {
         ostr << testcase << std::endl << colorRed << "[Test␣FAILED]␣" << colorWhite << "Result:␣(Expected:␣" << std::boolalpha << expe
<< std::noboolalpha << std::endl << std::endl;
      }
#else
      if (expected == result) {
         ostr << testcase << std::endl << "[Test␣OK]␣"  << "Result:␣(Expected:␣" << std::boolalpha << expected << "␣==" << "␣Result:␣"
      }
      else {
         ostr << testcase << std::endl  << "[Test␣FAILED]␣"  << "Result:␣(Expected:␣" << std::boolalpha << expected << "␣!=" << "␣Resul
      }
#endif
      if (ostr.fail()) {
         std::cerr << "Error:␣Write␣Ostream" << std::endl;
      }
   }
   else {
      std::cerr << "Error:␣Bad␣Ostream" << std::endl;
   }
   return expected == result;
}

template <typename T1, typename T2>
std::ostream& operator<< (std::ostream& ost,const std::pair<T1,T2> & p) {
   if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
   ost << "(" << p.first << "," << p.second << ")";
   return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::vector<T> & cont) {
   if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
   std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
   return ost;
}
```

```

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::list<T> & cont) {
   if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
   std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
   return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::deque<T> & cont) {
   if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
   std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
   return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::forward_list<T> & cont) {
   if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
   std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
   return ost;
}


#endif // !TEST_HPP
```