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

Name: Simon Offenberger / Simon Vogelhuber — Aufwand in h: siehe Doku

Mat.Nr: S2410306027 / S2410306014 — Punkte:

Übungsgruppe: 1 — korrigiert:

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

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

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

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

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

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

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

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

Testen Sie die Klasse ebenfalls ausführlich im Testtreiber!

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

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

Systemdokumentation
Projekt 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

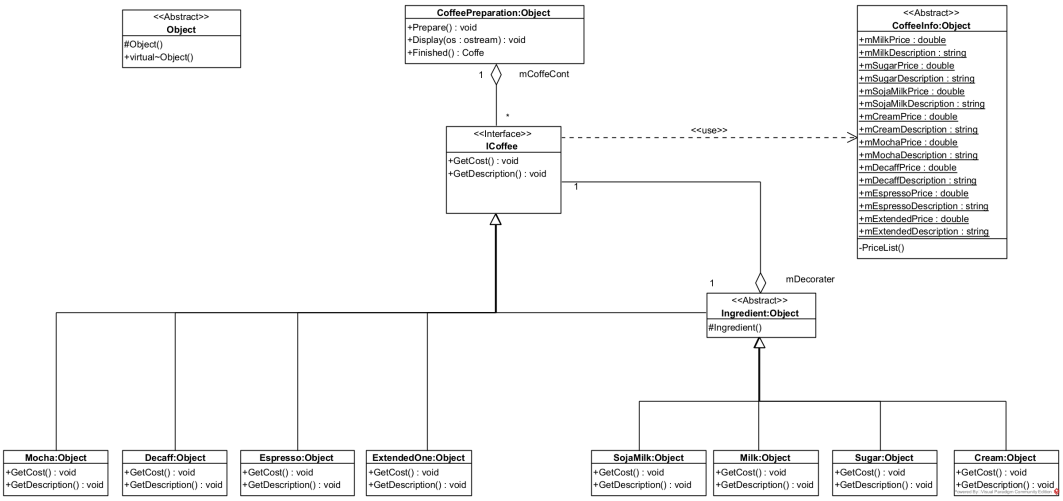
1.3 Aufwand

- 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

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

```
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
```

```
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)
```

```
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  == Result: Espresso: Milk 5.5 Euro
194 )
195
196 Test CoffeePreparation Display 2
197 [Test OK] Result: (Expected: Extended One: SojaMilk 20 Euro
198  == Result: Extended One: SojaMilk 20 Euro
199 )
200
201 Test CoffeePreparation Display 3
202 [Test OK] Result: (Expected: Mocha: Sugar 4.2 Euro
203  == Result: Mocha: Sugar 4.2 Euro
204 )
205
206 Test for Exception in Testcase
207 [Test OK] Result: (Expected: true == Result: true)
208
209 Test Exception Bad Ostream in CoffeePreparation
210 [Test OK] Result: (Expected: Error Bad Ostream == Result: Error Bad Ostream
        ↪ )
211
212 Test for Exception in Ingedient CTOR
213 [Test OK] Result: (Expected: Error Nullptr! == Result: Error Nullptr!)
```

```
214
215 TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```
1  /**
2   * @file Object.h
3   * @brief Defines a minimal base object with virtual destructor support.
4   */
5  #ifndef OBJECT_H
6  #define OBJECT_H
7
8  #include <string>
9
10 class Object{
11 public:
12
13 protected:
14
15    /**
16     * @brief Base constructor for derived objects.
17     */
18    Object(){};
19 public:
20    /**
21     * @brief Virtual destructor to allow safe polymorphic deletion.
22     */
23    virtual ~Object(){}
24 };
25
26 #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

```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";

private:
   CoffeeInfo() = default;
};


#endif // !COFFEE_INFO_HPP
```

6.4 Ingredient.hpp

```
/**
 * @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

```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 <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.
```

```
29      */
30     virtual std::string GetDescription() override;
31
32 };
33
34 #endif // !SOJA_MILK_HPP
```

6.8 SojaMilk.cpp

```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

```
/**
 * @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

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

6.11 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

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

6.13 Cream.hpp

```
/**
 * @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

```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

```
/**
 * @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

```
1  /**
2   * @file Decaff.hpp
3   * @brief Declares the decaffeinated coffee implementation of ICoffee.
4   */
5  #ifndef DECAFF_HPP
6  #define  DECAFF_HPP
7
8  #include "Object.h"
9  #include "ICoffee.hpp"
10
11 class Decaff : public ICoffee, public Object {
12
13    using Sptr = std::shared_ptr<Decaff>;
14
15    /**
16     * @brief Return the price of a decaffeinated coffee.
17     */
18    virtual double GetCost() override;
19
20    /**
21     * @brief Provide the decaff description label.
22     */
23    virtual std::string GetDescription() override;
24
25 };
26
27 #endif // !DECAFF_HPP
```

6.20 Decaff.cpp

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

6.21 Mocha.hpp

```
1  /**
2   * @file Mocha.hpp
3   * @brief Declares the mocha coffee implementation of ICoffee.
4   */
5  #ifndef MOCHA_HPP
6  #define  MOCHA_HPP
7
8  #include "Object.h"
9  #include "ICoffee.hpp"
10
11
12 class Mocha : public ICoffee, public Object {
13
14    using Sptr = std::shared_ptr<Mocha>;
15
16    /**
17     * @brief Return the price of a mocha.
18     */
19    virtual double GetCost() override;
20
21    /**
22     * @brief Provide the mocha description label.
23     */
24    virtual std::string GetDescription() override;
25
26 };
27
28 #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);
```

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

```
148         output << TestStart;
149         output << "Test␣Mocha" << endl << endl;
150         TestCoffeeIngridient(output, make_unique<Mocha>(), CoffeeInfo::mMochaInfo + ":", CoffeeInfo::mMochaPrice);
151         output << TestEnd;
152
153         output << TestStart;
154         output << "Test␣Decaff" << endl << endl;
155         TestCoffeeIngridient(output, make_unique<Decaff>(), CoffeeInfo::mDecaffInfo + ":", CoffeeInfo::mDecaffPrice);
156         output << TestEnd;
157
158         output << TestStart;
159         output << "Test␣Extended␣One" << endl << endl;
160         TestCoffeeIngridient(output, make_unique<ExtendedOne>(), CoffeeInfo::mExtendedInfo + ":", CoffeeInfo::mExtendedPrice);
161         output << TestEnd;
162
163         output << TestStart;
164         output << "Test␣Espresso␣with␣Milk" << endl << endl;
165         TestCoffeeIngridient(output, make_unique<Milk>(make_unique<Espresso>()),
166            CoffeeInfo::mEspressoInfo + ":␣" + CoffeeInfo::mMilkInfo + ",",
167            CoffeeInfo::mEspressoPrice + CoffeeInfo::mMilkPrice);
168         output << TestEnd;
169
170         output << TestStart;
171         output << "Test␣Extended␣One␣with␣SojaMilk" << endl << endl;
172         TestCoffeeIngridient(output, make_unique<SojaMilk>(make_unique<ExtendedOne>()),
173            CoffeeInfo::mExtendedInfo + ":␣" + CoffeeInfo::mSojaMilkInfo + ",",
174            CoffeeInfo::mExtendedPrice + CoffeeInfo::mSojaMilkPrice);
175         output << TestEnd;
176
177         output << TestStart;
178         output << "Test␣Mocha␣with␣Sugar" << endl << endl;
179         TestCoffeeIngridient(output, make_unique<Sugar>(make_unique<Mocha>()),
180            CoffeeInfo::mMochaInfo + ":␣" + CoffeeInfo::mSugarInfo + ",",
181            CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice);
182         output << TestEnd;
183
184         output << TestStart;
185         output << "Test␣Decaff␣with␣Cream" << endl << endl;
186         TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Decaff>()),
187            CoffeeInfo::mDecaffInfo + ":␣" + CoffeeInfo::mCreamInfo + ",",
188            CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice);
189         output << TestEnd;
190
191         output << TestStart;
192         output << "Test␣Decaff␣with␣Cream␣and␣Cream" << endl << endl;
193         TestCoffeeIngridient(output, make_unique<Cream>(make_unique<Cream>(make_unique<Decaff>())),
194            CoffeeInfo::mDecaffInfo + ":␣" + CoffeeInfo::mCreamInfo + ",␣" + CoffeeInfo::mCreamInfo + ",",
195            CoffeeInfo::mDecaffPrice + CoffeeInfo::mCreamPrice + CoffeeInfo::mCreamPrice);
196         output << TestEnd;
197
198         output << TestStart;
199         output << "Test␣Mocha␣alla␣Diabetes" << endl << endl;
200         TestCoffeeIngridient(output, make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
201            make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
202               make_unique<Sugar>(make_unique<Sugar>(make_unique<Sugar>(
203                  make_unique<Mocha>()))))))))),
204            CoffeeInfo::mMochaInfo + ":␣" + CoffeeInfo::mSugarInfo + ",␣"
205            + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
206            + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
207            + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",␣"
208            + CoffeeInfo::mSugarInfo + ",␣" + CoffeeInfo::mSugarInfo + ",",
209            CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPrice * 9);
210         output << TestEnd;
211
212
213         TestCoffeePreparation(output);
214
215         TestCoffeeIngridientException(output);
216
217
218
219
220         if (TestOK) {
221            output << TestCaseOK;
222         }
```

```
223          else {
224             output << TestCaseFail;
225          }
226
227          output.close();
228       }
229
230       if (TestOK) {
231          cout << TestCaseOK;
232       }
233       else {
234          cout << TestCaseFail;
235       }
236    }
237    catch (const string& err) {
238       cerr << err << TestCaseFail;
239    }
240    catch (bad_alloc const& error) {
241       cerr << error.what() << TestCaseFail;
242    }
243    catch (const exception& err) {
244       cerr << err.what() << TestCaseFail;
245    }
246    catch (...) {
247       cerr << "Unhandelt␣Exception" << TestCaseFail;
248    }
249
250    if (output.is_open()) output.close();
251
252    return 0;
253
254
255 }
256
257 bool TestCoffeeIngridient(std::ostream & ost,ICoffee::Uptr cof, const std::string & description, const double price)
258 {
259    assert(cof != nullptr);
260    assert(ost.good());
261
262    std::string error_msg;
263    bool TestOK = true;
264
265    try {
266       TestOK = TestOK && check_dump(ost, "Test␣ICoffee␣Description", cof->GetDescription(), description);
267       TestOK = TestOK && check_dump(ost, "Test␣ICoffee␣Price", cof->GetCost(), price);
268    }
269    catch (const string& err) {
270       error_msg = err;
271    }
272    catch (bad_alloc const& error) {
273       error_msg = error.what();
274    }
275    catch (const exception& err) {
276       error_msg = err.what();
277    }
278    catch (...) {
279       error_msg = "Unhandelt␣Exception";
280    }
281
282    TestOK = TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Testcase", true, error_msg.empty());
283
284
285    return TestOK;
286 }
287
288 bool TestCoffeeIngridientException(std::ostream& ost)
289 {
290    assert(ost.good());
291
292    std::string error_msg;
293    bool TestOK = true;
294
295    try {
296       ICoffee::Uptr cof = make_unique<Milk>(nullptr);
297    }
```

```
298     catch (const string& err) {
299        error_msg = err;
300     }
301     catch (bad_alloc const& error) {
302        error_msg = error.what();
303     }
304     catch (const exception& err) {
305        error_msg = err.what();
306     }
307     catch (...) {
308        error_msg = "Unhandelt␣Exception";
309     }
310
311     TestOK = TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Ingedient␣CTOR", Ingredient::ERROR_NULLPTR, error_msg);
312
313     return TestOK;
314 }
315
316 bool TestCoffeePreparation(std::ostream& ost) {
317
318     assert(ost.good());
319
320
321     std::string error_msg;
322     bool TestOK = true;
323
324     try {
325        CoffeePreparation CoffeeMachine;
326
327        CoffeeMachine.Prepare(make_unique<Milk>(make_unique<Espresso>()));
328        CoffeeMachine.Prepare(make_unique<SojaMilk>(make_unique<ExtendedOne>()));
329        CoffeeMachine.Prepare(make_unique<Sugar>(make_unique<Mocha>()));
330
331        stringstream expected_output;
332        stringstream actual_output;
333
334
335        CoffeeMachine.Display(actual_output);
336
337        expected_output << CoffeeInfo::mEspressoInfo + ":␣" + CoffeeInfo::mMilkInfo + "␣" << CoffeeInfo::mEspressoPrice + CoffeeInfo::mMi
338
339        TestOK = TestOK && check_dump(ost, "Test␣CoffeePreparation␣Display␣1", actual_output.str(), expected_output.str());
340
341        ICoffee::Uptr cof = CoffeeMachine.Finished();
342
343        actual_output.str("");
344        expected_output.str("");
345
346        CoffeeMachine.Display(actual_output);
347
348        expected_output << CoffeeInfo::mExtendedInfo + ":␣" + CoffeeInfo::mSojaMilkInfo + "␣" << CoffeeInfo::mExtendedPrice + CoffeeInfo:
349
350        TestOK = TestOK && check_dump(ost, "Test␣CoffeePreparation␣Display␣2", actual_output.str(), expected_output.str());
351
352        cof = CoffeeMachine.Finished();
353
354        actual_output.str("");
355        expected_output.str("");
356
357        CoffeeMachine.Display(actual_output);
358
359        expected_output << CoffeeInfo::mMochaInfo + ":␣" + CoffeeInfo::mSugarInfo + "␣" << CoffeeInfo::mMochaPrice + CoffeeInfo::mSugarPr
360
361        TestOK = TestOK && check_dump(ost, "Test␣CoffeePreparation␣Display␣3", actual_output.str(), expected_output.str());
362
363        cof = CoffeeMachine.Finished();
364
365     }
366     catch (const string& err) {
367        error_msg = err;
368     }
369     catch (bad_alloc const& error) {
370        error_msg = error.what();
371     }
372     catch (const exception& err) {
```

```
373         error_msg = err.what();
374      }
375      catch (...) {
376         error_msg = "Unhandelt␣Exception";
377      }
378
379      TestOK = TestOK && check_dump(ost, "Test␣for␣Exception␣in␣Testcase", true, error_msg.empty());
380
381      try {
382
383         CoffeePreparation CoffeeMachine;
384
385         stringstream badstream;
386
387         badstream.setstate(ios::badbit);
388
389         CoffeeMachine.Display(badstream);
390      }
391      catch (const string& err) {
392         error_msg = err;
393      }
394      catch (bad_alloc const& error) {
395         error_msg = error.what();
396      }
397      catch (const exception& err) {
398         error_msg = err.what();
399      }
400      catch (...) {
401         error_msg = "Unhandelt␣Exception";
402      }
403
404      TestOK = TestOK && check_dump(ost, "Test␣Exception␣Bad␣Ostream␣in␣CoffeePreparation", CoffeePreparation::ERROR_BAD_OSTREAM, error_ms
405
406
407      return TestOK;
408   }
```

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
```

```
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
```