Aufgabenblatt 5

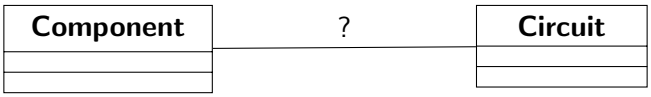
Themen

- C++23 Standardbibliothek
- Dynamische Speicherallokation (Smart pointer)
- Exception Handling

Aufgabe 1 - Verwendung Dynamischer Speicher

In Ihrem System ist es möglich, dieselben Bauteile in mehreren Schaltkreisen gleichzeitig einzubauen. Unterbinden Sie dies indem Sie im `Component`-Container der Klasse `Circuit` statt `Component*` einen passenden *Smart Pointer* der C++ Standardbibliothek verwenden.

Welche Assoziation wird mit diesem Schritt zwischen `Component` und `Circuit` umgesetzt?



Aufgabe 2 - Schaltkreise editieren

Erweitern Sie Ihre Anwendung, sodass eine Komponente aus einem Schaltkreis entfernt werden kann.

Empfohlene Vorgehensweise:

- Ersetzen Sie in der Klasse `Circuit` den Container für die Komponenten durch `std::map`. Als *Key* empfiehlt sich der Name der Komponente.
- Passen Sie den Code entsprechend an und testen Sie Ihr Programm. Es sollte nun genauso funktionieren wie vorher!
- Implementieren Sie die Funktion `bool delete_component(std::string)`.

Hinweis: Bedenken Sie, dass es beliebig viele Sub-Schaltkreise geben kann. Überlegen Sie sich eine Möglichkeit, alle Komponenten zu durchsuchen.

Aufgabe 3 - Ausgabe mit `std::print`

In den vorherigen Aufgaben haben Sie die einzelnen Bauteile (`Component`) eines Schaltkreises (`Circuit`) per `print()`-Methode auf der Konsole ausgegeben. Erweitern Sie Ihren Quelltext nun so, dass die Bauteile auch über `std::print()` ausgegeben werden können, wie in unten dargestellt[1].

Hinweise:

- Definieren Sie für z. B. Ihre `Resistor`-Klasse eine `std::formatter`-Spezialisierung, damit diese in `std::print()` verwendet werden kann.
- Die Ausgabe soll den Namen und gegebenenfalls weitere Eigenschaften der Komponente enthalten.
- Nutzen Sie dabei die C++23 Standardbibliothek.

```
Resistor c{"Resistor", 1000};
std::print("Bauteil: {}\n", c);
```

Aufgabe 4 - Exception-Handling

Erweitern Sie Ihr Programm um eine sinnvolle Ausnahmebehandlung.

Überlegen Sie zunächst, welche problematischen Zustände in Ihrem Programm auftreten könnten - zum Beispiel durch ungültige Benutzereingaben, falsche Daten oder unerwartetes Verhalten bei der Programmausführung. Identifizieren Sie diese möglichen Fehlerquellen und implementieren Sie für jeden dieser Fälle eine geeignete Fehlerbehandlung.

Sie können verschiedene Strategien anwenden - vom einer einfachen Ausgabe einer Warnung bishin zum Abbruch des Programms. Wichtig ist, dass Sie bei jedem Fehler abwägen, welche Folgen er für den weiteren Programmablauf hat.

Beantworten Sie insbesondere die folgenden Fragen:

- Welche Auswirkungen hat der jeweilige Fehlerzustand auf den weiteren Verlauf des Programms?
- Kann das Programm trotz des Fehlers weiterarbeiten oder muss es an dieser Stelle abgebrochen werden?
- Kann dieser Zustand vom Programm erkannt und entsprechend behandelt werden?

(Optional): Definieren Sie eigene Exception-Klassen, um bestimmte Fehlertypen besser zu unterscheiden und gezielt zu behandeln.

Abtestat

- Passen Sie Ihr UML-Klassendiagramm an Ihre Änderungen in diesem Aufgabenblatt an.
- Fassen Sie (mündlich) zusammen, welche Änderungen Sie an Ihrem Code vorgenommen haben. Welche Anforderungen haben Probleme bereitet und wie haben Sie diese gelöst?
- Demonstrieren Sie das Löschen einer Komponente anhand einiger Beispiele.
- Benennen Sie (ggfs. Notizen erstellen) welche Fehler Sie abfangen und wie Sie damit umgehen. Zeigen Sie dies an einem konkreten Beispiel.

[1] `https://www.cppstories.com/2022/custom-stdformat-cpp20/#custom-formatters`