Workshop 2 - 27.01.2025

*Gruppemedlemmer:*

Emil Berglund

Sebastian Thomsen

Obaida Taha

Hedda Ørnelund Nilsen

Bibliotek 1: Emil og Obaida

**Feil/Brytning av retningslinjer**

* Get og Set er skrevet vertikalt, ikke horisontalt.
* Alle List deklarasjoner har mellomrom, noe den ikke skal ha.
  + List < ShopOrder > (feil)
  + List <ShopOrder> (riktig)
* “public static decimal CalculateShippingCost” returnerer 0, men bør returnere den totale kostnaden for frakt.
* “public static bool IsWithinBudget” returnerer 0, men bør sjekke om totalprisen er innenfor budsjettet, og returnerer en bool verdi deretter.
* “public OptimizationResult OptimizeOrders” returnerer ingenting, men må dette, ettersom den ikke er markert med void. Eksempelvis “return new OptimalizationResult”
* Generell feilhåndtering. Metoder med settere, mangler feilhåndtering. Eksempelvis “MaxAmount”. Her kan man egentlig skrive et negativt tall, som ikke er heldig i programmets sammenheng.

**Mangler i koden:**

Koden er ufullstendig, og mangler blant annet en funksjon som søker opp de komponenetene som er tilgjengelige. Denne kunne eksempelvis sett slik ut:

**Pseudo-code**

Scenario 1: Søk etter produkt, og finn billigst pris.

Søk opp komponent CPU

Hent ulike alternativer

Initialiser minimum pris og butikk

Gå gjennom butikkene og finn den billigste prisen

Returner den billigste butikken og prisen

Scenario 2: Bestilling av komponeneter innenfor et budsjett

Start med å søke etter komponenten "CPU"

Hent de alternative komponentene

Initialiser minimum pris og butikk

Gå gjennom butikkene og finn den billigste prisen

Returner den billigste butikken og prisen

Scenario 3: Optimalisering av bestillinger fra flere butikker:

Start med en liste over nødvendige komponenter

Samle alle butikker

Optimaliser bestillinger

Regn ut den totale prisen for bestillingen inkludert frakt

Sjekk om den totale kostnaden er innenfor budsjettet

Bibliotek 2: Hedda og Sebastian

**Hva programmet gjør:**

Library 2 er et rammeverk for å optimalisere innkjøp og sammensetning av PC-komponenter. Det er organisert i navnerommet *PCOrderOptimizer* og gir en strukturert måte å håndtere PC-komponenter, produkter og butikker på. Komponenter kategoriseres med enumen *ComponentCategory*, mens produkter og butikker modelleres gjennom henholdsvis strukturen Product og klassen Store.

Brukere kan definere PC-konfigurasjoner med klassen *PCBuildConfig*, som inkluderer budsjett og nødvendige komponenter. Grensesnittet *IOrderOptimizer* lar brukeren implementere optimaliseringsstrategier via metoden *OptimizeOrders*. Resultatet av en optimalisering modelleres med strukturen *OrderStrategy*, som viser ordrer og totale kostnader.

Enkelte metoder, som *OptimizeOrders* og *GroupByEquivalence*, mangler implementasjon, og biblioteket kunne dratt nytte av validering og feilhåndtering. Til tross for dette gir Library 2 en fleksibel og utvidbar grunnstruktur for å planlegge og optimalisere PC-bygging.

**Pseudokode**

Scenario 1:

Opprett en instans av klassen “Store” og sett egenskaper:

* Navn på butikken
* Gratis frakt-grense
* Fraktkostnad
* Opprett flere butikker

Opprett klassen: “Component category”:

* Navn på de ulike kategoriene

Scenario 2:

Opprette flere produkter ved bruk av “Product” strukturen:

* Sette navnet på produktet
* Sette prisen på produktet
* Kategorisere produktet
* Gruppere med tilsvarende produkter

Legg produktene i listen til “Products” i “Store”

Scenario 3:

Opprett en instans av *PCBuildConfig*

* Sett budsjett
* Legg til nødvendige komponentkategorier (f.eks. RAM, CPU)

Implementer *OptimizeOrders*` fra *IOrderOptimizer*:

* Beregn den totale kostnaden for produkter som oppfyller kravene i *PCBuildConfig*.
* Velg den billigste strategien for innkjøp.
* Returnerer optimalisert ordrestrategi.

**Retningslinjer biblioteket bryter**

* Mangler validering: Det ser ikke ut til å være innebygd validering for egenskaper som Budget eller Price, *EquivalentGroup*.
* Ingen implementasjon av metoder: Mange metoder, som *GroupByEquivalence* og *OptimizeOrders*, mangler implementasjon. Dette gjør biblioteket uferdig. Her må det implementeres manglende metoder for å gjøre biblioteket funksjonelt.
* Ingen feilhåndtering: Det finnes ingen håndtering av potensielle feil, som å legge til produkter med ugyldige verdier. Budsjetter i *PCBuildConfig* som er null eller negative. Fraktkostnader som er ugyldige eller null. Feilhåndtering bør inkluderes ved å validere input og kaste unntak der det er nødvendig.
* Det mangler return på *IOrderOptimzer*
* Store inneholder en liste av produkter (Products), men det er ingen mekanisme for å validere at produktene i butikken oppfyller en spesifikk konfigurasjon i *PCBuildConfig*.
* *PCBuildConfig* definerer krav til komponentkategorier, men det er ingen sjekk for at Products i en Store faktisk dekker alle nødvendige kategorier.

**Forslag til forbedringer:**

* Implementer alle manglende metoder.
* Legg til feilhåndtering for negative priser, ugyldige budsjetter og andre invalidverdier.
* Gjør avhengighetene mellom klasser tydeligere, f.eks. ved å sikre at alle nødvendige kategorier i *PCBuildConfig* kan dekkes av produkter i Store.
* Fjern nullable typer der det ikke er nødvendig, og erstatt dem med fornuftige standardverdier.

**Diskusjon og Konklusjon:**

I del 2 byttet vi på, hvor Emil og Obaida analyserte Bibliotek 2, mens Hedda og Sebastian analyserte Bibliotek 1. Begge gruppene identifiserte hovedsakelig de samme feilene og foreslo lignende løsninger. Hovedfokus i diskusjonen var på strukturelle forbedringer, implementering av manglende metoder, og hvordan man kan forbedre feilhåndtering og validering av input.

**Enighet om forbedringer:**

* Viktige forbedringer som ble enstemmig foreslått inkluderte implementering av manglende metoder og validering av input.