Design Patterns

Repository Pattern & Unit of Work

Introduktion till Designmönster

Vad är Designmönster?
Designmönster är återanvändbara lösningar på vanliga problem inom mjukvaruutveckling.
De hjälper oss att skapa kod som är:

- **Lättare att underhålla**
- **Testbar**
- **Utökningsbar**

Varför använda dem?

- Standardiserar utvecklingen.
- Förbättrar koden’s struktur.
- Underlättar samarbeten i team.

Repository Pattern

Vad är Repository Pattern?

- Ett abstraktionslager mellan domänlogik och datåtkomst.
- Centraliserar datåtkomstlogik i en separat klass.

Fördelar:

- **Lättare att testa:** Simulerar databasåtkomst via mock-objekt.
  Till exempel kan du skapa enhetstester som verifierar logiken utan att någonsin koppla upp sig mot en riktig databas.
  Detta sparar tid och minskar komplexiteten.
- **Läsbarhet:** Samlar all datåtkomst på ett ställe, vilket gör koden renare och enklare att förstå.
  Exempelvis vet alla utvecklare att de ska gå till repository-klassen för att hitta eller uppdatera databaslogik.
- **Flexibilitet:** Underlättar byte av databas eller datakälla.
  Du kan till exempel byta från SQL Server till en NoSQL-lösning genom att bara justera repository-implementationen utan att påverka affärslogiken.

Repository Pattern

Hur fungerar det?

Komponenter:

1. **Interface:** Definierar de operationer som stöds, t.ex. CRUD.
2. **Repository-klassen:** Implementerar interfacet och innehåller logik för datåtkomst.
3. **Domänlogik:** Konsumerar repository för att hämta eller manipulera data.

Exempel:

https://github.com/z3ph1/CleanCode-NET23/assets/Samples/Design%20Patterns/repository.cs

Unit of Work

Vad är Unit of Work?

Unit of Work är ett designmönster som används för att hantera transaktioner och koordinera ändringar mellan flera repositories.

Det fungerar som en mellanhand som samlar alla ändringar i ett enda objekt och ser till att de antingen sparas tillsammans eller rullas tillbaka om något går fel.

Detta gör att databasen hålls konsekvent och integriteten bevaras.

Huvudfunktioner:

- **Transaktionshantering:** Samlar flera databasoperationer i en enda transaktion.
- **Centralisering:** Håller reda på alla ändringar (inserts, updates, deletes) i en session.
- **Effektivitet:** Reducerar antalet databasoperationer genom att optimera skrivningar.

Unit of Work

En säker transaktion

När en transaktion misslyckas, utförs en rollback som återställer databasen till sitt tidigare tillstånd.
Detta säkerställer att applikationen förhåller sig till affärslogikens regler och undviker halvfärdiga uppdateringar.

Om en transaktion misslyckas i Unit of Work, rullas alla ändringar tillbaka (rollback).
Detta sker genom att databasen återställer sitt tidigare tillstånd, så att inga halvfärdiga eller inkonsekventa data sparas.

Till exempel:

Om en order skapas och produktsaldot uppdateras, men uppdateringen av kundens orderhistorik misslyckas, räknas hela transaktionen som ogiltig.

Systemet gör rollback på alla operationer och databasen återgår till sitt ursprungliga tillstånd.

Detta garanterar att inga inkonsekvenser uppstår och att applikationen förhåller sig till affärslogikens regler.

Unit of Work

Rollback

Triggas vanligtvis av ett undantag (exception) som kastas under transaktionen.

Exempelvis:

- Om en databasoperation misslyckas (t.ex. en constraint violation; datan du skickar in inte passar databasens modell/tabell/column eller logik).
- Om en applikationsspecifik regel bryts och kod explicit kastar ett undantag.

Databasen eller transaktionshanteraren identifierar att ett fel har inträffat och utför rollback för att återställa till föregående tillstånd.

- UnitOfWork är ett mönster som koordinerar flera repository-uppdateringar i en enda transaktion.
- Säkerställer att alla operationer lyckas eller rullas tillbaka.

Fördelar:

- **Dataintegritet:** Minimerar risken för inkonsistens i databasen.
- **Effektivitet:** Samlar flera operationer i en enda transaktion.
- **Testbarhet:** Centraliserar databasåtkomst.

Unit of Work

Hur fungerar det?

Komponenter:

1. **Interface:** Definierar metoder som är relaterade till transaktionshantering.
2. **Implementation:** Hanterar livscykeln för databasen och repositories.

Exempel:

https://github.com/z3ph1/CleanCode-NET23/assets/Samples/Design%20Patterns/UnitOfWork.cs

Varför kombinera Repository och Unit of Work?

Syfte:

- Separera ansvaret mellan databasoperationer och affärslogik.
- Hantera flera operationer i en enda transaktion.

Användningsfall:

1. **Komplexa affärsregler:** Hantering av flera databasuppdateringar.
2. **Skalbara applikationer:** Förbättrar struktur och testbarhet.
3. **Databasåtkomst:** Underlättar byte av databas.

Exempel på Användning

Orderhantering i en e-handelsplattform.

1. En kund lägger en order.
2. Systemet uppdaterar:
    1. Ordertabellen.
    2. Produktsaldot.
    3. Kundens orderhistorik.
3. Allt hanteras inom en transaktion via Unit of Work.

```
using (var unitOfWork = new UnitOfWork(context, productRepository))
{
    var product = unitOfWork.Products.GetById(order.ProductId);
    product.Stock -= order.Quantity;

    unitOfWork.Orders.Add(order);

    await unitOfWork.CommitAsync();
}
```

Sammanfattning

Repository Pattern:

- Abstraherar datåtkomst.
- Gör koden mer testbar och flexibel.

Unit of Work:

- Koordinerar transaktioner mellan flera repositories.
- Säkerställer dataintegritet.

Tillsammans:

- Förbättrar struktur, testbarhet och underhåll av applikationer.