

Rețeaua socială

Seminar 4 – Repository în baza de date, Simple Factory, configurare externă

Enunț

Până acum, `Main` alege singur implementarea de repository: `new InMemoryRepository<>(...)` sau `new UtilizatorFileRepository(...)`. Urmează un repository în baza de date. Vrem ca schimbarea stocării să nu mai modifice codul care folosește repository-ul.

- `UtilizatorDbRepository` implementează `Repository<Long, Utilizator>`; metodele vor fi scrise cu JDBC într-un seminar viitor.
- `RepositoryFactory` este singurul loc care cunoaște clasele concrete de repository (Simple Factory).
- Tipul de stocare și parametrii lui se citesc din `config.properties`, nu din cod.
- Nu folosim încă interfețe funcționale și expresii lambda.

Cerința 4. Repository în baza de date (schelet)

Creați clasa `UtilizatorDbRepository` în pachetul `curs.repository.db`. Clasa implementează `Repository<Long, Utilizator>` și primește prin constructor un `Validator<Utilizator>`, url-ul bazei de date, user-ul și parola.

Metodele **nu** se implementează încă: fiecare aruncă `UnsupportedOperationException` cu un mesaj clar (de exemplu `"findAll nu este implementat inca"`).

(axa de schimbare: locuri noi de stocare)

Întrebări

1. În `cerinta3`, `Main` scrie `new UtilizatorFileRepository(validator, "data/utilizatori.txt")`. Câte locuri din program trebuie modificate dacă datele trec în baza de date? De ce clase concrete depinde `Main`?

Răspuns: Fiecare `new UtilizatorFileRepository(...)` trebuie înlocuit, iar `Main` trebuie recompilat. `Main` depinde de clasa concretă, nu doar de interfața `Repository`, și cunoaște detalii de stocare (numele fișierului) care nu îl privesc.

2. De ce `UtilizatorDbRepository` nu extinde `InMemoryRepository`, așa cum face `AbstractFileRepository`?

Răspuns: Fișierul text se încarcă în memorie la pornire și se rescrie la fiecare modificare, deci memoria este sursa datelor. Baza de date este ea însăși sursa: fiecare operație devine un query. O copie în memorie ar duplica datele și s-ar putea desincroniza de tabel.

3. De ce aruncăm `UnsupportedOperationException` și nu întoarcem `null` din metodele neimplementate?

Răspuns: `null` are deja un sens în contractul `Repository` (de exemplu, `save` întoarce `null` dacă entitatea a fost salvată). O metodă neimplementată care întoarce `null` ar părea că a reușit. Excepția oprește programul și spune exact ce lipsește.

4. `UnsupportedOperationException` este checked sau unchecked? Ce s-ar întâmpla dacă ar fi checked?

Răspuns: Unchecked (extinde `RuntimeException`). Dacă ar fi checked, ar trebui declarată cu `throws` în antet, dar o metodă suprascrisă nu poate declara excepții checked pe care metoda din interfață nu le declară: codul nu ar compila.

Cerința 5. Factory pentru repository

Definiți enum-ul `RepositoryType` cu valorile `MEMORY`, `FILE` și `DATABASE`.

Creați clasa `RepositoryFactory` în pachetul `curs.repository.factory`, cu metoda `Repository<Long, Utilizator> createUtilizatorRepository(Validator<Utilizator> validator)`, care întoarce implementarea potrivită pentru tipul de stocare. Folosiți un `switch` clasic (`case MEMORY: return ...;`).

Pentru un tip necunoscut, metoda aruncă `IllegalArgumentException`.

Parametrii diferă de la o stocare la alta: fișierul are nevoie de un nume, baza de date de url, user și parolă.

Factory-ul primește un obiect `Properties` prin constructor și citește tipul din cheia `repository.type`.

Tratați cazul în care proprietatea lipsește.

(axa de schimbare: tipuri noi de stocare)

Întrebări

- De ce un enum și nu un `String` de tipul `"file"` sau `"db"` trimis prin program?

Răspuns: Enum-ul fixează mulțimea valorilor posibile: o greșeală de scriere în cod nu compilează. Textul din fișierul de configurare se transformă o singură dată, cu `RepositoryType.valueOf(...)`, care aruncă `IllegalArgumentException` pentru o valoare necunoscută.

- Metoda `createUtilizatorRepository` întoarce `Repository<Long, Utilizator>`, nu `UtilizatorFileRepository`. De ce?

Răspuns: Apelantul trebuie să depindă doar de interfață (DIP). Dacă tipul de retur ar fi clasa concretă, `Main` ar fi din nou legat de o implementare, iar factory-ul nu ar mai putea întoarce altă clasă.

- În `switch`, fiecare `case` se termină cu `return`. Ce se întâmplă dacă scriem `repo = ...;` fără `break`?

Răspuns: Execuția continuă în `case`-ul următor (fall-through): pentru `MEMORY` s-ar crea, pe rând, toate cele trei repository-uri și ar rămâne ultimul. Cu `return` ieșim imediat din metodă, deci `break` nu mai este necesar.

```
case MEMORY:
    repo = new InMemoryRepository<Long, Utilizator>(validator);
case FILE:
    // se executa si pentru MEMORY!
    repo = new UtilizatorFileRepository(validator, fisier);
```

- Ramura `default` nu se execută pentru niciuna dintre valorile actuale ale enum-ului. Compilează metoda fără ea? De ce o păstrăm?

Răspuns: Nu compilează: pentru un `switch` clasic compilatorul nu garantează că toate valorile sunt acoperite, deci există un drum fără `return` („missing return statement”). În plus, dacă se adaugă o constantă în enum și se uită `case`-ul, `default` semnalează eroarea.

- Ce trebuie modificat ca să adăugăm un al patrulea tip de stocare, de exemplu fișier XML? Ce rămâne neschimbat?

Răspuns: O clasă nouă de repository, o constantă în `RepositoryType` și un `case` în factory. `Main`, `Repository` și celelalte implementări rămân neschimbate. Factory-ul se modifică la fiecare tip nou, dar modificarea este într-un singur loc.

Cerința 6. Configurare externă

Creați fișierul `src/main/resources/config.properties` cu cheile `repository.type`, `file.utilizatori`, `db.url`, `db.user` și `db.password`.

În `Main`, scrieți metoda `Properties incarcaConfiguratie(String resursa)`, care citește fișierul din classpath cu `getResourceAsStream` și `try-with-resources`. Dacă resursa nu există sau nu poate fi citită, metoda aruncă `RuntimeException` și păstrează cauza.

În `cerinta4`, obțineți repository-ul din factory și afișați numele clasei create și rezultatul lui `findAll()`. Rulați programul cu `repository.type=FILE`, apoi cu `MEMORY`, apoi cu `DATABASE`.

(axa de schimbare: mediul în care rulează aplicația)

Întrebări

1. Ce s-a schimbat în cod între cele trei rulări? Ce excepție apare la `DATABASE` și de ce este un rezultat așteptat?

Răspuns: Codul nu s-a schimbat, doar valoarea din `config.properties`. La `DATABASE` factory-ul creează corect `UtilizatorDbRepository`, iar `findAll()` aruncă `UnsupportedOperationException`, pentru că metodele sunt încă schelet.

2. `getResourceAsStream` întoarce `null` dacă resursa nu există. Ce s-ar întâmpla dacă nu am verifica acest caz?

Răspuns: `props.load(null)` aruncă `NullPointerException`, al cărei mesaj nu spune ce fișier lipsește. Verificarea explicită transformă problema într-o excepție cu mesaj clar.

3. `config.properties` este în `src/main/resources`, iar `utilizatori.txt` în `data/`. De ce unul se citește cu `getResourceAsStream`, iar celălalt cu `FileReader`?

Răspuns: Resursele din `src/main/resources` ajung în classpath (în `build/resources/main` sau în jar) și se găsesc după nume, oriunde rulează programul. `FileReader` caută o cale în sistemul de fișiere, relativă la directorul curent. Configurația se livrează cu aplicația; datele se modifică la rulare și nu pot fi scrise într-un jar.

4. Parola bazei de date stă în `config.properties`. Ce problemă apare dacă proiectul este pus pe GitHub?

Răspuns: Parola devine publică. În practică se citește din variabile de mediu sau dintr-un fișier local exclus din git (`.gitignore`), iar în repository rămâne doar un exemplu (`config.example.properties`).

5. Când apare repository-ul de `Prietenie`, cum ne asigurăm că utilizatorii și prietenii folosesc același tip de stocare?

Răspuns: O metodă `createPrietenieRepository` în același factory, care citește același `repository.type`. Pasul următor este Abstract Factory: o interfață cu câte o metodă pentru fiecare repository și câte o implementare pentru fiecare tip de stocare (`FileRepositoryFactory`, `DbRepositoryFactory`), astfel încât toate repository-urile vin din aceeași sursă.

Soluție (`config.properties`, `Main.incarcaConfiguratie`, `Main.cerinta4`)