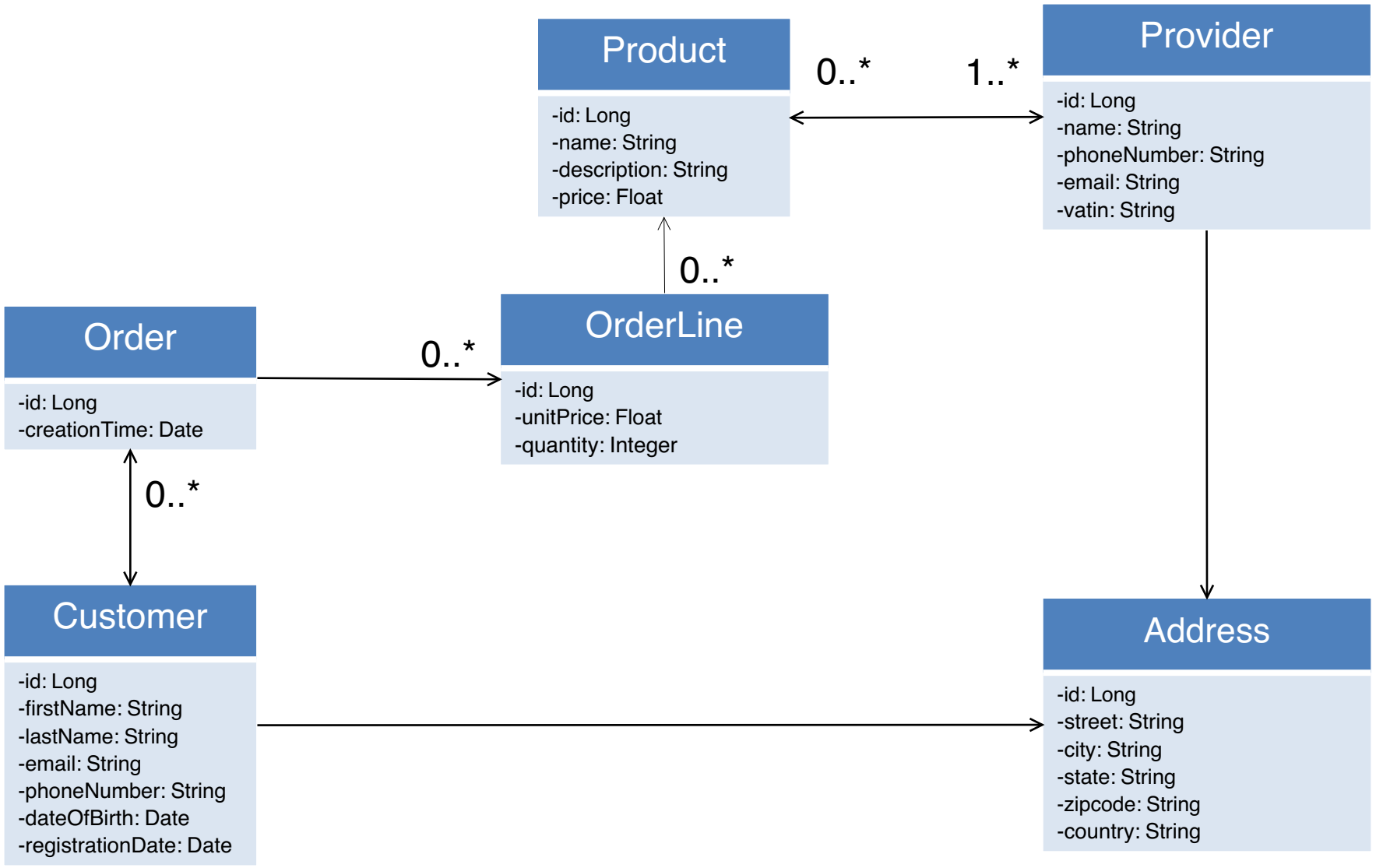
Sistemi informativi su Web

Esercizione Persistenza

aa 2024-2024

Paolo Merialdo
Università degli Studi Roma Tre

Esempio


Product
-id: Long
-name: String
-description: String
-price: Float
0..*
1..*
Provider
-id: Long
-name: String
-phoneNumber: String
-email: String
-vatin: String
0..*
Order
-id: Long
-creationTime: Date
0..*
OrderLine
-id: Long
-unitPrice: Float
-quantity: Integer
0..*
Customer
-id: Long
-firstName: String
-lastName: String
-email: String
-phoneNumber: String
-dateOfBirth: Date
-registrationDate: Date
Address
-id: Long
-street: String
-city: String
-state: String
-zipcode: String
-country: String

Operazioni Base

- Creiamo un progetto siw-jpa-02 con le stesse caratteristiche di siw-jpa-01
- Modifichiamo la classe SiwJpa02Application specificando che deve implementare l'interfcaccia CommandLineRunner

SiwJpa01Application **implements** CommandLineRunner

- Annotare il metodo run() della classe SiwJpa02Application con l'annotazione @Transactional

Operazioni Base

- Aggiungiamo nel modello tutte le classi del nostro caso di studio (Customer, Address, Order, OrderLine, Provider)
- Creare il package repository
- Per ogni classe del modello creare la corrispondente interfaccia repository
- Nella classe SiwJpa02Application definiamo le variabili CustomerRepository e AddressRepository, e annotiamole come @Autowired

Esercizio 1

- Obiettivo: capire il funzionamento del metodo save() dei repository e la creazione e l'assegnazione dell'id

Esercizio 1

- Nel metodo run() della classe SiwJpa02Application scrivere il codice per:
  - Creare un oggetto Customer (con l'operatore java new)
  - Stampare i dati dell'oggetto (con il metodo toString() opportunamente definito affinchè stampi i valori di tutte le variabili di istanza)
  - Salvare tramite il repository l'oggetto Customer nel database
  - Stampare nuovamente i dati dell'oggetto Customer
- Che differenza c'è tra la prima e la seconda stampa?

Esercizio 2

- Obiettivo: capire il funzionamento di save() con i riferimenti che implementano le associazioni e la gestione dei cascade

Esercizio 2

- Modificare il metodo run() come segue:
  - Dopo aver creato un oggetto Customer, creare anche un oggetto Address ed assegnarlo all'oggetto Customer
  - Provare ad eseguire il codice
  - C'è un problema: capirlo e risolverlo (si può usare un LLM)

Esercizio 2 (cont.)

- Modificare la classe Customer abilitando il cascade dell'operazione di persist sulla associazione OneToOne con Address
- Modificare il metodo run() come segue:
  - Dopo aver creato l'oggetto Customer, creare anche un oggetto Address ed assegnarlo all'oggetto Customer
  - Attraverso il repository, salvare l'oggetto Customer
  - Provare ad eseguire il codice
- Analizzare (nella console) tutte le istruzioni SQL che sono state inviate al database

Esercizio 3

- Obiettivo: capire le strategie di Fetch

Esercizio 3: preparazione

- Modifichiamo il file application.properties affinchè non vengano cancellati i dati che sono già nel database ogni volta che eseguiamo il programma. Impostiamo:

  spring.jpa.hibernate.ddl-auto=none

Esercizio 3 (parte 1)

- Scriviamo il corpo del metodo run() come segue:
  - Invochiamo il findAll() del repository CustomerRepository per recuperare dal database tutti gli oggetti Customer
  - Iteriamo sulla collezione recuperata dal database stampando i dettagli (senza indirizzo) di ogni customer
  - Analizzare (nella console) le istruzioni SQL che sono state inviate al database. In particolare, quante istruzioni SELECT?

Esercizio 3 (parte 2)

- Modificare la classe Customer impostando a LAZY la strategia di fetch sulla associazione OneToOne con Address
- Eseguire il programma precedente
- Analizzare (nella console) le istruzioni SQL che sono state inviate al database. In particolare, quante istruzioni SELECT? Ci sono differenze rispetto al caso precedente?

Esercizio 3 (parte 3)

- Modificare la classe Customer impostando a LAZY la strategia di fetch sulla associazione OneToOne con Address
- Nella iterazione, far stampare anche i dettagli sull'indirizzo di ogni oggetto
- Analizzare (nella console) le istruzioni SQL che sono state inviate al database. In particolare, quante istruzioni SELECT? Ci sono differenze rispetto al caso precedente?

Esercizio 4

- Obiettivo: definire metodi del repository con la con convenzione sui nomi

Esercizio 4

Scrivere nella interface CustomerRepository i metodi che implementano queste operazioni

- // Trova tutti i clienti con un determinato nome
- // Trova tutti i clienti con un certo cognome
- // Trova un cliente con nome e cognome specifici
- // Trova tutti i clienti nati in una certa data
- // Trova tutti i clienti nati prima di una certa data
- // Trova tutti i clienti nati dopo una certa data
- // Trova tutti i clienti il cui nome inizia con una certa stringa
- // Trova tutti i clienti il cui cognome termina con una certa stringa
- // Trova tutti i clienti il cui nome contiene una certa stringa
- // Trova tutti i clienti con nome diverso da un certo valore
- // Trova tutti i clienti con nome o cognome uguale a un valore
- // Trova tutti i clienti ordinati per cognome
- // Conta quanti clienti hanno un certo nome
- // Conta quanti clienti hanno un certo cognome
- // Conta quanti clienti hanno un certo nome e cognome
- // Conta quanti clienti sono nati prima di una certa data
- // Conta quanti clienti hanno un nome che inizia con una certa stringa
- // Conta quanti clienti hanno un cognome che termina con una certa stringa
- // Conta quanti clienti hanno un nome diverso da un certo valore

Esercizio 4

- Popolare il database e verificare il funzionamento dei metodi del repository