Enterprise Software Architecture – Übungsprogramm

Jörn Kreutel– Beuth Hochschule für Technik Berlin – Sommersemester 2018
Stand: 10. April 2018

Inhaltsverzeichnis

Aufgaben

Lerneinheit	Übungskürzel	Übungsinhalt	Punkte	Abgabe
BAS	BAS2	Reflection für Attribute und Methoden	4	1
BAS	BAS3	Verwendung von Annotationen	3	1
SER	SER3	Delete Funktion für Web Service	4	1
SER	SER4	Create Funkion für Web Service	5	1
JRS	JRS2	REST Service Implementierung	6	1
JRS	JRS3	Polymorphie	2	1
JWS	JWS4	Erstellung eines neuen Web Service	6	1
JWS	JWS5	Erstellung eines Web Service Clients	4	1
WSV	WSV1	JAX-RS Client	14	3
ECH	ECH	E-Business Hackathon	10	
EJB	EJB2	Implementieren von StockSystem	5	2
JPA	JPA2	Datenmodellierung	6	2
JPA	JPA3	DAO für AbstractProduct	7	2
JPA	JPA4	DAO für StockItem	9	2
ADD	ADD2	EJB als JAX-WS WebService		
ADD	ADD3	StockSystem als JAX-WS Web Service	6	3
ADD	ADD4	Nutzung von Transaktionen	2	3
ADD	ADD5	Stateless ShoppingSession		
PAT	PAT1	ShoppingSession als Session Facade	6	3
PAT	PAT2	Erweiterte ShoppingSessionFacade	7	3
JSF	JSF1	Verwendung von Products EJB		
JSF	JSF2	Verwendung von StockSystem EJB		
JSF	JSF3	Verwendung von CustomerCRUD EJB		
JSF	JSF4	Einbinden von ShoppingSession EJB		
JSF	JSF5	GUI zur Verwaltung von StockItem	12	3
JSF	JSF6	Aktion zur Ausführung von doShopping()	2	
			120	

Grün hinterlege Aufgaben sind **Pflichtaufgaben**, von deren Umsetzung die nachfolgenden Aufgaben z.T. abhängen. Grau hinterlegte Aufgaben gehören im Sommersemester 2018 nicht zum Übungsprogramm. Eine aktive Vermittlung des Lernstoffs für die gelb gekennzeichneten Aufgaben JSF5 jund JSF6 wird voraussichtlich entfallen, das Thema JSF wird jedoch durch Skript, Präsentationsfolien und Implementierungsbeispiele abgedeckt.

Falls die Veranstaltung E-Business Hackathon (EHC), blau markiert, im Mai/Juni 2018 angeboten wird, wird eine erfolgreiche, durch ein Zertifikat bescheinigte, Teilnahme mit bis zu 10 Punkten auf das ESA Übungsprogramm angerechnet.

100% Bemessungsgrundlage für die Bewertung sind 92 Punkte. Zusätzlich zu den Pflichtaufgaben können Sie Aufgaben nach Belieben auswählen, um diese Punktzahl zu erreichen.

SETUP

0. Voraussetzungen

- Verwenden Sie für alle Installationen und Workspaces keine Verzeichnisse, in deren Pfad Leerzeichen oder Sonderzeichen enthalten sind.
- Stellen Sie sicher, dass Sie ein Java JDK in Version 1.8 installiert haben.
- Falls Sie keine entsprechende Version verfügbar haben, können Sie unter den folgenden Links ein JDK herunterladen und installieren:
  - `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`
- Stellen Sie sicher, dass die Versionsverwaltungssoftware *Git* auf Ihrem Entwicklungsrechner installiert ist. Installieren Sie andernfalls eine für Ihren Rechner geeignete Version: `https://git-scm.com/downloads`

1. IDEA installieren

- Laden Sie die Jetbrains IDEA Entwicklungsumgebung in der *Ultimate* Variante herunter und installieren Sie sie: `https://www.jetbrains.com/idea/download`
- Starten Sie IDEA und nutzen Sie entweder die Evaluierungslizenz für 30 Tage oder registrieren Sie sich *unter Ihrer Hochschuladresse* bei JetBrains unter `https://www.jetbrains.com/student/` – für Adressen unter `@beuth-hochschule.de` bekommen Sie freien Zugriff auf IDEA und andere JetBrains Produkte.
- Zur Dokumentation von IDEA siehe `https://www.jetbrains.com/idea/documentation/`
- Für deutschsprachige Tastatur-Shortcuts siehe `https://victorvolle.wordpress.com/2012/05/16/intellij-german-keyboard-shortcuts-reference/` - sollten auch damit Probleme auftreten, können Sie Shortcuts individuell einrichten. Siehe dafür: `https://www.jetbrains.com/help/idea/configuring-keyboard-shortcuts.html`
- Im Zuge der Lehrveranstaltung benötigen Sie u.a. die beiden 'Tool Windows' *Maven Projects* und *Application Servers*. Diese können Sie über die Menüoption *View* → *Tool Windows...* öffnen. Siehe zu Tool Windows auch `https://www.jetbrains.com/idea/help/manipulating-the-tool-windows.html`

2. Implementierungsbeispiele aus Git laden

- IDEA zeigt Ihnen nach dem Starten eine Ansicht zum Öffnen, Erstellen und Laden von Projekten an.
- Wählen Sie die Option *Check out from Version Control* → *Git* und geben Sie als *Repository URL* die folgende URL ein: `https://github.com/dieschnittstelle/org.dieschnittstelle.jee.esa.mvn.git`
  - Wenn Sie Ihren Quellcode selbst über GitHub verwalten möchten, dann richten Sie, falls noch nicht vorhanden, einen GitHub Account ein, erstellen Sie eine *Fork* des o.g. Projekts und geben Sie deren URL in IDEA ein. Siehe dafür `https://help.github.com/articles/fork-a-repo/`

- IDEA wird das ausgewählte Projekt laden und die Entwicklungsumgebung entsprechend den Projektanforderungen initialisieren. Dieser Vorgang kann mehrere Minuten dauern.
- Akzeptieren Sie insbesondere etwaige Aufforderungen bezüglich der Ausführung von Java und des Akzeptieren von eingehenden Netzwerkverbindungen.
- Die Implementierungsbeispiele enthalten alle Ressourcen, die Sie zur Umsetzung der Übungsaufgaben benötigen, sowie Konfigurationen zur Laufzeitausführung.
- Öffnen Sie *View* → *Tool Windows* → *Maven Projects*. Ihnen wird eine Liste aller Teilprojekte der Implementierungsbeispiele angezeigt.
- Klappen Sie in der *Maven Projects* Ansicht im Projekt `build-all-modules` die *Lifecycle* Aktionen für dieses Projekt auf. Diese Aktionen dienen dazu, Projekte mit Maven zu bauen, d.h. den Quellcode zu kompilieren, eine JAR-Datei zu erstellen, diese ins lokale Maven Repository zu übertragen, etc. sowie ggf. gebaute Artefakte 'aufzuräumen'.
- Führen Sie zunächst durch Doppelklick die Aktion `clean` und nach Abschluss die Aktion `install` aus. Damit sollten alle Projekte gebaut und die JAR Artefakte in Ihrem lokalen Maven Repository abgelegt werden.
  - Das lokale Maven Repository befindet sich normalerweise im Verzeichnis *.m2* im Home-Verzeichnis des auf Ihrem Entwicklungsrechner angemeldeten Users. Die JARs der Projekte werden in der dortigen Verzeichnisstruktur in den Verzeichnissen unter `repository/org/dieschnittstelle/jee/esa` abgelegt.
- Für die einzelnen Aufgaben des Übungsprogramms wurden für IDEA sogenannte 'Scopes' deklariert. Diese können Sie nutzen, um die Projektansicht übersichtlich für die jeweils aufgabenrelevanten Teilprojekte zu filtern. Scopes können durch Öffnen des Aufklappmenüs in der Projektansicht oben links geöffnet werden. Beachten Sie, dass Ihnen bei Auswahl eines Scopes die Teilprojekte unter den Namen der zu den Teilprojekten gehörenden Maven-Artefakte angezeigt werden. Diese entsprechen in der Regel den Projektnamen nach dem `.esa` Segment, z.B. entspricht das Artefakt `ser-client` dem Projekt `org.dieschnittstelle.jee.esa.ser.client`. Eine Ausnahme stellen die Projekte `.ser`, `.jrs`, `.jws` sowie `.ejb` dar, deren Artefaktnamen zusätzlich den Java EE Modultyp bezeichnen, d.h. `ser-webapp`, `jrs-webapp`, `jws-webapp` sowie `ejb-earapp`.
- Eine Dokumentation zur Integration von Maven in IDEA finden Sie hier: `https://www.jetbrains.com/help/idea/maven.html`
- Für weitere Hinweise zur Verwendung von Git in IDEA siehe: `https://www.jetbrains.com/help/idea/using-git-integration.html`

3. Java für Implementierungsbeispiele überprüfen/einrichten

- Öffnen Sie in IDEA die Projektkonfiguration unter *File* → *Project Structure* (siehe auch
- Überprüfen Sie unter *Projects*, ob als *Project SDK* Java 1.8 gesetzt ist.
- Überprüfen Sie andernfalls, ob Ihnen Java 1.8 als Option angeboten wird und weisen Sie diese zu. Wählen Sie ansonsten via *New...* Ihr Java 1.8 Installationsverzeichnis aus und weisen Sie Java 1.8 als *Project SDK* zu.

4. Apache Tomcat konfigurieren

- Laden Sie Apache Tomcat in Version 8.5.27 herunter: `https://archive.apache.org/dist/tomcat/tomcat-8/v8.5.27/bin/apache-tomcat-8.5.27.zip`
- Entpacken Sie die `.zip` Datei in einem Verzeichnis, in dem Sie weitestgehende Lese- und Schreibbefugnisse haben.
- Öffnen Sie in IDEA die Konfigurationsansicht unter *Run* → *Edit Configurations...* und wählen Sie in der Gruppe *Tomcat Server* die Konfiguration *JRS*.
- Öffnen Sie die Konfigurationseinstellungen via *Configure...* und weisen Sie dem Server mit Namen *Tomcat 8.5.11* das entpackte Tomcat-Verzeichnis zu.
- Öffnen Sie via *View* → *Tool Windows...* die Ansicht *Application Servers* und starten Sie Tomcat über die Konfiguration *JRS*. Nach erfolgreichem Starten sollte der Browser für die URL `http://localhost:8888/org.dieschnittstelle.jee.esa.jrs/` geöffnet werden.
- Falls Sie einen Mac verwenden und beim Starten von Tomcat einen *Permission denied* Fehler bekommen, dann öffnen Sie ein Terminal auf dem Installationsverzeichnis von Tomcat und führen Sie das folgende Kommando aus: `chmod a+x bin/catalina.sh`

5. JBoss Wildfly Application Server konfigurieren

- Laden Sie JBoss Wildfly in Version 11.0.0.Final von `http://download.jboss.org/wildfly/11.0.0.Final/wildfly-11.0.0.Final.zip` herunter.
- Entpacken Sie die `.zip` Datei in einem Verzeichnis, in dem Sie weitestgehende Lese- und Schreibbefugnisse haben.
- Öffnen Sie in IDEA die Konfigurationsansicht unter *Run* → *Edit Configurations...* und wählen Sie in der Gruppe *JBoss Server* die Konfiguration *EJB*.
- Öffnen Sie die Konfigurationseinstellungen via *Configure...* und weisen Sie das entpackte JBoss-Verzeichnis zu.
- Wenn nach Zuweisung des JBoss-Verzeichnisses zur *EJB* Konfiguration für die anderen für JBoss eingerichteten Konfigurationen (z.B. *EJB+JSF*, *JWS* etc.) Fehler angezeigt werden, dann starten Sie IDEA neu. Danach sollten auch diese Konfigurationen als funktionsfähig dargestellt werden und ausführbar sein.
- Führen Sie jetzt Schritt 6 – *H2 Datenbank starten* – aus, wie unten beschrieben.
- Öffnen Sie via *View* → *Tool Windows...* die Ansicht *Application Servers* und starten Sie JBoss über die Konfiguration *EJB+JSF*. Nach erfolgreichem Starten sollte der Browser für die URL `http://localhost:8080/org.dieschnittstelle.jee.esa.jsf/` geöffnet werden. Zuvor müssen Sie aber noch die nachfolgend beschriebe Einrichtung der H2 Datenbank durchführen...
- Führen Sie die Konfiguration *EJB+JPA: TotalUsecase* aus, um den client-seitigen Zugriff auf JBoss zu verifizieren.

- Falls z.B. aufgrund von Firewall-Einstellungen auf Ihrem Entwicklungsrechner, kein Zugriff des Clients auf den Server möglich ist, dann versuchen Sie die folgende Fehlerbehebung:
  - Stoppen Sie die *EJB+JSF* Konfiguration
  - Ändern Sie in der Konfigurationsdatei *standalone.xml* im Verzeichnis *standalone/configuration* Ihrer JBoss Installation das `socket-binding` für `http` auf Port `4447`.
  - Ändern Sie in der Konfigurationdatei *jboss-ejb-client.properties* die `host` Einstellung auf `127.0.0.1` und die `port` Einstellung auf `4447`.
  - Starten Sie *EJB+JSF* neu und führen Sie `TotalUsecase` noch einmal aus.

6. H2 Datenbank starten

- Um für die Aufgaben zu JPA*ff* die von JBoss verwendete H2 Datenbank einzusehen, muss die Datenbank als Server gestartet werden und ein Viewer für H2 verwendet werden, der in der JBoss Installation enthalten ist.
- Öffnen Sie in IDEA die Konfigurationsansicht unter *Run* → *Edit Configurations...* und wählen Sie die Konfiguration *H2 DB*.
- Weisen Sie als *Path to JAR* die Datei `h2-1.4.193.jar` im Unterverzeichnis *modules/system/layers/base/com/h2database/h2/main/* Ihrer JBoss Wildfly Installation zu.
- Prüfen Sie vor Starten von JBoss und vor dem Starten der Datenbank die Inhalte der Datei `jpa-ds.xml` im Verzeichnis *src/main/resources/META-INF* des Projekts `.esa.ejb.ejbmodule.erp`. Als Connection-URL für H2 sollte ein Element mit dem folgenden Wert verwendet werden: `<connection-url>jdbc:h2:tcp://localhost/~/crm_erp_db</connection-url>`
- Falls für H2 ein anderes `<connection-url/>` Element gesetzt ist, kommentieren Sie dieses aus und ersetzen es durch das vorgenannte Element.
- Starten Sie den H2 Datenbankserver in IDEA durch Ausführung der *H2 DB* Konfiguration. Sie sollten ein User Interface im Browser erhalten, über das Sie sich anmelden können.
- Geben Sie als Credentials den Nutzernamen `sa` ein und verwenden Sie ein leeres Passwort. Geben Sie `jdbc:h2:tcp://localhost/~/crm_erp_db` als Wert des Formularfelds 'JDBC URL' ein. Achtung: Bei Übertragung der JDBC URL via Copy&Paste muss mindestens das Tilde-Symbol ('~') neu eingegeben oder ersetzt und müssen die Unterstriche ersetzt, d.h. ebenfalls neu eingegeben, werden!!!
- Das Starten der Datenbank auf dem genannten Wege ist erforderlich. Falls die Datenbank nicht gestartet wurde, schlägt der Start der Konfigurationen für JBoss fehl. Unter MacOS kann es evtl. erforderlich sein, einen unter dem Namen *Console* laufenden H2 Datenbankprozess manuell zu stoppen.

7. Implementierungsvarianten

- Die Implementierungsbeispiele für die Übungsaufgaben SER, JRS, JWS sowie EJB beinhalten neben dem synchronen client-seitigen Zugriff auf server-seitige Anwendungsfunktionen jeweils Beispiele für die durch die aktuellen Java EE APIs unterstützten asynchronen Zugriffs-

bzw. Bereitstellungsvarianten. Die Verwendung dieser Varianten können Sie aktivieren, indem Sie in den Client-Klassen `ShowTouchpointService`, `ShowTouchpointRESTService`, `ShowTouchpointSOAPService` bzw. `TotalUsecase` das dort jeweils deklarierte boolesche Instanzattribut bzw. die lokale boolesche Variable `async` auf `true` setzen. Beachten Sie jedoch, dass sich das Übungsprogramm im aktuellen Stand jeweils auf die synchronen Zugriffs- und Bereitstellungsvarianten bezieht.

- Die Beispiele für EJB demonstrieren als Alternative zur Verwendung von `@Stateful` EJBs als zustandsbehafteten server-seitigen Komponenten den Zugriff auf server-seitige Funktionen mittels zustandsloser REST Services. Wo eine Umsetzung dieser Variante im Rahmen des Übungsprogramms sinnvoll ist, wird explizit darauf hingewiesen.

BAS

Projekte:

- `org.dieschnittstelle.jee.esa.bas`

Server Runtime:

- (keine)

Ü BAS1 Methodenaufrufe via Reflection

Aufgabe

Erweitern Sie die Methode `buildStockItemFromElement` der Klasse `ReflectedStockItemBuilder` so, dass die in der XML Datei angebenen Werte für <`price`> auf den erzeugten Instanzen von `IStockItem` gesetzt werden können.
Diese Aufgabe ist bereits umgesetzt. Machen Sie sich mit dem Programmcode vertraut.

Anforderungen

1. Verwenden Sie die Ausdrucksmittel der Reflection API, d.h. nehmen Sie an, dass Sie weder die beiden Klassen `Milk` und `Chocolate`, noch die Existenz eines `price` Attributs auf diesen Klassen kennen.

Bearbeitungshinweise

- Alle Attributnamen und Werte, die in der XML Datei verwendet werden sind in der lokalen `HashMap` namens `nodes` enthalten.
- Die Stelle, an der Sie die Ergänzungen vornehmen können, ist im Quellcode markiert.

Ü BAS2 Reflection für Attribute und Methoden (4 Punkte)

Aufgabe

Implementieren Sie die Methode `showAttributes()` der Klasse `ShowAnnotations`. Sie können hier und im folgenden die Ausdrucksmittel der Reflection API anhand eines einfachen Anwendungsfalls praktisch anwenden. Dieser Anwendungsfall findet sich in realen Anwendungen z.B. überall dort wieder, wo Java Objekte zwecks Übertragung über Netzwerkverbindungen in textueller Form, z.B. in JSON oder XML, repräsentiert werden müssen.

Anforderungen

1. Geben Sie für jedes Objekt, das der Methode übergeben wird, den Namen der Objektklasse sowie die durch die Objektklasse selbst deklarierten Attributnamen und deren Werte in der folgenden Form aus:

   `{<einfacher Klassenname> <attr1>:<Wert von attr1>, ...}`, z.B.:
   `{Milch menge:20, markenname:Mark Brandenburg}`

2. Verwenden Sie die Ausdrucksmittel der Reflection API, d.h. nehmen Sie an, dass Sie die beiden Klassen `Milch` und `Schokolade` nicht kennen.

Ü BAS3 Verwendung von Annotationen (3 Punkte)

Aufgabe

Deklarieren Sie einen Annotationstyp `DisplayAs` mit Attribut `value`, der für Attribute von Klassen gesetzt und zur Laufzeit via Reflection ausgelesen werden kann.

Anforderungen

1. Ändern Sie die Implementierung der `showAttributes()` Methode aus Aufgabe 2 wie folgt: Wenn die `DisplayAs` Annotation für ein Attribut gesetzt ist, stellen Sie deren `value` Parameterwert anstelle des Attributsnamens dar, andernfalls verwenden Sie den Attributsnamen.
2. Setzen Sie die `DisplayAs` Annotation auf ausgewählten Attributen der Klassen `Milch` und `Schokolade`.

SER

Projekte:

- `org.dieschnittstelle.jee.esa.ser`
- `org.dieschnittstelle.jee.esa.lib.entities.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.ser.client` (Client)

Server Runtime:

- `SER` (Tomcat)

Ü SER0 Überblick

Aufgabe

Hier finden Sie Erläuterungen zu den Implementierungsbeispielen.

Bearbeitungshinweise

- Die Implementierungsbeispiele bestehen aus einer Webanwendung (`.ser` bzw. `ser-webapp`) und einer Client-Anwendung (`.ser.client`) und illustrieren einige der in der Präsentation angesprochenen Funktionen von Java EE Web Applikationen anhand zweier Szenarien:
  - Das `TouchpointGUIServlet` steuert eine einfache graphische Nutzerschnittstelle zur Darstellung und Manipulation einer Menge existierender Verkaufsstellen (Touchpoints).- Hintergrund ist als E-Business-Szenario eine *Abverkaufskampagne für verderbliche Güter*.
    - Der Aufbau des HTML Markups erfolgt mittels der JSP `gui.jsp`
    - Der Zugriff auf das Servlet wird mittels des Filters `TouchpointGUIServletFilter` auf Browser eingeschränkt.
  - Das `TouchpointServiceServlet` stellt eine programmatische Schnittstelle zur Verfügung.
    - Als Client fungiert die Klasse `ShowTouchpointService` im Projekt `.ser.client`.
    - Daten werden als serialisierte Java-Objekte ausgetauscht (in der Realität wäre das aufgrund der damit verbundenen Abhängigkeiten zwischen Client(s) und Webanwendung problematisch).
    - Bisher können nur Daten ausgelesen werden – die Erweiterung ist Bestandteil des Übungsprogramms und kann mittels der JUnit Testklasse `TestTouchpointService` verifiziert werden.
- In beiden Szenarien wird für das Persistieren von Daten die Klasse `TouchpointCRUDExecutor` verwendet. Diese wird durch den `TouchpointServletContextListener` verwaltet und über den `ServletContext` der Anwendung für alle Komponenten der Anwendung verfügbar gemacht.
- Eingehende Http Requests werden durch die Klasse `HttpTrafficLoggingFilter` als Logmeldung ausgegeben.

Ü SER1 Programmatischer Zugriff auf GUI Servlet

Aufgabe

Versuchen Sie, aus der Client Anwendung auf das Servlet zuzugreifen, das die graphische Nutzeroberfläche steuert.

Bearbeitungshinweise

- Ersetzen Sie temporär in der Methode `readAllTouchpoints` der Client-Anwendung `ShowTouchpointService` die zugegriffene URI durch `http://localhost:8888/org.dieschnittstelle.jee.esa.ser/gui/touchpoints` und führen Sie die Anwendung von neuem aus.
- Was für einen Fehler bekommen Sie, und welche Komponente auf Seiten der Webanwendung ist dafür verantwortlich?
- Wie könnten Sie diese Komponente 'überlisten'? – einen Hinweis dafür finden Sie in den Kommentaren der modifizierten Methode. Versuchen Sie, die Implementierung entsprechend zu ändern – sie werden aufgrund des nun nicht mehr verarbeitbaren Rückgabeformats aber einen client-seitigen Fehler bekommen.
- Machen Sie die Änderungen wieder rückgängig.

Ü SER2 `Filter` für Service

Aufgabe

Verhindern Sie, dass aus dem Browser auf das Servlet, das die Programmierschnittstelle zur Verfügung stellt, zugegriffen werden kann.

Bearbeitungshinweise

- Versuchen Sie von einem Browser aus auf die URI `http://localhost:8888/org.dieschnittstelle.jee.esa.ser/api/touchpoints` zuzugreifen. Ihnen sollte ein Download eines unbekannten Datenformats angeboten werden.
- Unterbinden Sie die Browser-Zugreifbarkeit auf die besagte URI durch Implementierung und Konfiguration einer geeigneten Java EE Komponente. Sie können sich dafür an der bestehenden Umsetzung der Zugriffsverhinderung für programmatische Clients aus Ü1 orientieren.

Ü SER3 `Create` Funktion für Service (5 Punkte)

Aufgabe

Implementieren Sie die `createNewTouchpoint()` Methode der Klasse `ShowTouchpointService` durch Zugriff auf die Web Applikation und ergänzen Sie die Web Applikation entsprechend. Diese und die folgende Übung ermöglichen Ihnen das Kennenlernen client- und server-seitiger APIs zur Übermittlung und Bearbeitung von HTTP Requests in Java.

Anforderungen

1. Nutzen Sie für den HTTP Request, der durch den Client übermittelt wird die HTTP `POST` Methode.
2. Übermitteln Sie das server-seitig zu erstellende `AbstractTouchpoint` Objekt als serialisiertes Java-Objekt.
3. Nutzen Sie für die server-seitige Erstellung des Objekts die Methode `createTouchpoint()` auf `TouchpointCRUDExecutor`.
4. Testen Sie Ihre Implementierung sowie die Umsetzung von SER3 durch Ausführen der Testklasse `TestTouchpointService`.

Ü SER4 `Delete` Funktion für Service (4 Punkte)

Aufgabe

Implementieren Sie die `deleteTouchpoint()` Methode der Client-Klasse `ShowTouchpointService` durch Zugriff auf die Web Applikation und ergänzen Sie die Web Applikation entsprechend.

Anforderungen

1. Nutzen Sie für den HTTP Request, der durch den Client übermittelt wird, die HTTP `DELETE` Methode.
2. Wählen Sie für eine in Verbindung mit der genutzten HTTP Methode möglichst wenig redundante URI.
3. Nutzen Sie zur server-seitigen Ausführung des Löschens die Methode `deleteTouchpoint()` auf `TouchpointCRUDExecutor`.
4. Zeigen Sie dem Client die erfolgreiche Ausführung des Löschens bzw. etwaige Fehler durch geeignete Status Codes im HTTP Response an.

Bearbeitungshinweise

- ***Anforderung 1***: Sowohl der Apache HTTP Client, als auch die `HttpServlet` API unterstützen alle spezifizierten Methoden des HTTP Protokolls.
- ***Anforderung 4***: Auf Client-Seite brauchen Sie dann keinen Rückgabewert auszulesen, sondern nur den Status aus dem Response zu berücksichtigen.

Sonstiges

Wenn die Löschaktion erfolgreich ausgeführt wurde, müssen Sie über das GUI eine neue Verkaufsstelle erzeugen – oder Sie implementieren Ü SER4.

JRS

Projekte:

- `org.dieschnittstelle.jee.esa.jrs.api`
- `org.dieschnittstelle.jee.esa.jrs`
- `org.dieschnittstelle.jee.esa.lib.entities.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.jrs.client` (Client)

Server Runtime:

- `JRS` (Tomcat)

Ü JRS1 Erweiterung der Beispielanwendung

Aufgabe

Machen Sie die `updateObject()` Methode aus `GenericCRUDExecutor` über den in den Implementierungsbeispielen bereit gestellten REST Service für die Aktualisierung von `AbstractTouchpoint` Instanzen verfügbar.

Anforderungen

1. Erweitern Sie das Interface `ITouchpointCRUDService` und dessen Implementierung `TouchpointCRUDServiceImpl` um eine Methode, die den Zugriff auf die `updateObject()` Methode von `GenericCRUDExecutor` ermöglicht.

2. Verwenden Sie geeignete JAX-RS Annotationen, um die neue Methode über den REST Service aufrufbar zu machen.

3. Rufen Sie die neue Methode in der Client-Implementierung `ShowTouchpointRESTService` auf dem dort verwendeten `serviceClient` Objekt auf.

Sonstiges

Die Umsetzung dieser Aufgabe gehört nicht zum Pflichtprogramm, dient aber dem Kennenlernen der Ausdrucksmittel von JAX-RS als Vorbereitung für die nachfolgenden Pflichtaufgaben.

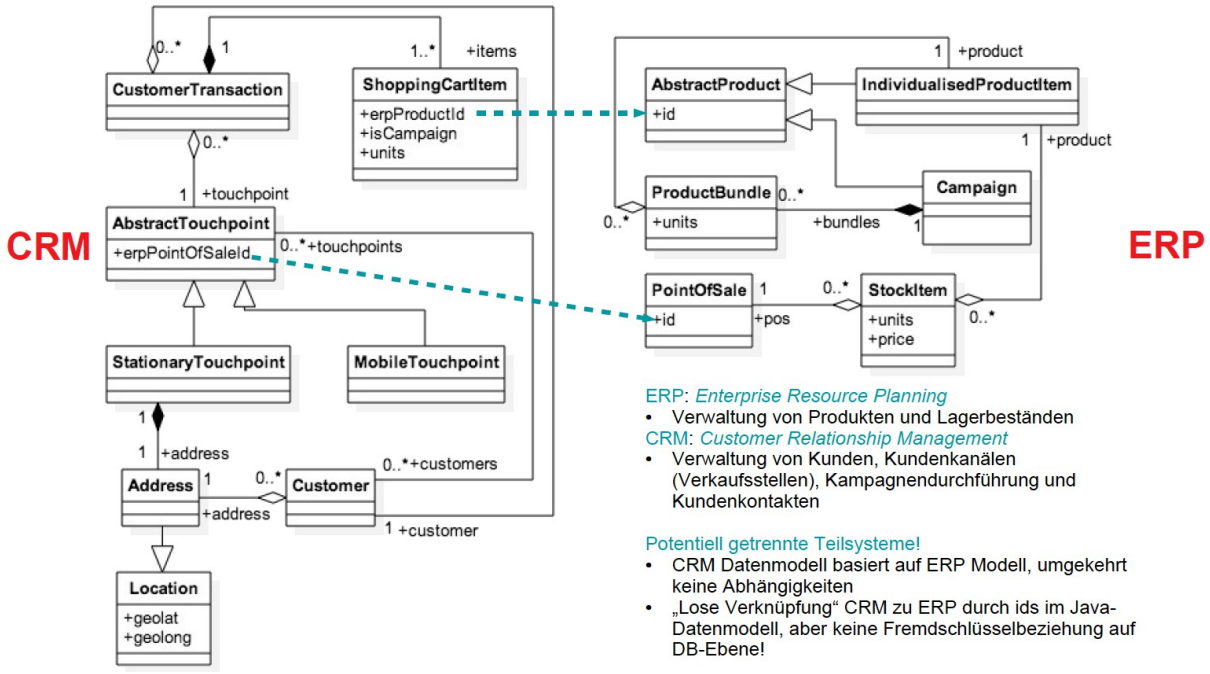
Ü JRS2 REST Service Implementierung (6 Punkte)

Aufgabe

Implementieren Sie unter Verwendung der Ausdrucksmittel von JAX-RS einen Service, der die Erstellung neuer Produkte und das Auslesen aller Produkte ermöglicht und greifen Sie auf diesen Service aus einer Client Implementierung zu. Die Übung ermöglicht Ihnen, mit JAX-RS eine standardisierte Beschreibungsform für Web APIs anzuwenden, die eine empfehlenswerte Alternative zur händischen Umsetzung proprietärer Web APIs darstellt, wie sie in SER realisiert wurde.

Ausgangssituation

- Im Package `org.dieschnittstelle.jee.esa.entities.erp` des Projekts `.lib.entities.erp` liegen mehrere Klassen, die ein Datenmodell für Produkte implementieren. Dieses finden Sie im nachfolgenden UML Diagramm rechts dargestellt.
  - Unterschieden werden einfache Produkte (`IndividualisedProductItem`) verschiedener Typen (`ProductType`), sowie Kampagnen (`Campaign`), die für einzelne Produkte in unterschiedlicher Stückzahl (`ProductBundle`) erstellt werden können. Als Abstraktion über `IndividualisedProductItem` und `Campaign` fungiert die Klasse `AbstractProduct`.
  - Mit diesem Datenmodell und seinen Erweiterungen werden sich auch die Übungsaufgaben in den kommenden Veranstaltungen beschäftigen.
- Im Package `org.dieschnittstelle.jee.esa.jrs` im Projekt `.jrs.api` finden Sie das Interface `IProductCRUDService`, dessen Implementierung `ProductCRUDServiceImpl` das Projekt `.jrs` bzw. `jrs-webapp` enthält.
- Die Klasse `ProductCRUDRESTClient` im Client-Projekt `.jrs.client` enthält ein Grundgerüst für einen Resteasy Client bezüglich `IProductCRUDService` und wird von der JUnit Testklasse `TestProductRESTService` verwendet. Um auf den Service zugreifen zu können, müssen Sie noch analog zum Beispiel in `ShowTouchpointRESTService` die `serviceProxy` Variable instantiieren.
- Die Klasse `GenericCRUDExecutor` im Package `org.dieschnittstelle.jee.esa.entities` des Projekts `.lib.entities.erp` generalisiert die Funktionalität von `TouchpointCRUDExecutor` aus der Veranstaltung SER und wird in der Beispielanwendung `.jrs` auf `AbstractTouchpoint` Instanzen angewendet. Die verwendete Instanz der Klasse wird in in `TouchpointServletContextListener` erzeugt.
- Für Instanzen von `AbstractProduct` wird in `ProductServletContextListener` eine entsprechend Instanz von `GenericCRUDExecutor` erstellt und im`ServletContext` als Wert des Attributs `productCRUD` abgelegt.



Anforderungen

1. Implementieren Sie die Methoden in `ProductCRUDServiceImpl` unter Verwendung der Instanz von `GenericCRUDExecutor`, die Sie als Wert des Attributs `productCRUD` aus dem `ServletContext` auslesen können.
2. Machen Sie die Implementierung durch Verwendung geeigneter JAX-RS Annotationen auf `IProductCRUDService` verfügbar. Nutzen Sie dafür die Wurzel-URI `/api/products`.
3. Instantiieren Sie im Client-Projekt die `serviceProxy` Variable in `ProductCRUDRESTClient`.
4. Verifizieren Sie die Funktionalität und Zugreifbarkeit des Services durch Ausführung der Testklasse `TestProductRESTService`.

Bearbeitungshinweise

- ***Anforderung 1***: Dafür können Sie `ServletContext` unter Verwendung der Annotation `javax.ws.rs.core.Context` im Konstruktor von `ProductCRUDServiceImpl` entgegen nehmen.
- ***Anforderung 1***: Für die Rückgabe aller Produkte können Sie in `ProductCRUDServiceImpl` den Rückgabetyp von `readAllObjects()` auf `GenericCRUDExecutor` auf den unspezifischen Typ `List` casten.

Sonstiges

Greifen Sie auf die Auslese-Methode außerdem zu, indem Sie die hierfür zu verwendende URI im Browser eingeben. Ihnen sollte eine Liste von JSON Objekten angezeigt werden. Speichern Sie diese Liste in einer Textdatei ab.

Ü JRS3 Polymorphie (2 Punkte)

Aufgabe

Verwenden Sie anstelle von `IndividualisedProductItem` die abstrakte Klasse `AbstractProduct` im Interface `IProductCRUDService` und in dessen Implementierung. Die Übung demonstriert, dass Web APIs bei Verwendung von Operationen, die abstrakte Klassen verwenden, Vorkehrungen treffen müssen, damit Instanzen konkreter Unterklassen überhaupt verarbeitet werden können.

Anforderungen

1. Ändern Sie die Methoden in `IProductCRUDService` und `ProductCRUDServiceImpl` so, dass Sie anstelle von `IndividualisedProductItem` die Klasse `AbstractProduct` verwenden.
2. Stellen Sie dann die Funktionsfähigkeit des Services wieder her.

Bearbeitungshinweise

- Führen Sie nun `TestProductRESTService` von neuem aus. Was für einen Fehler bekommen Sie?
- Entfernen Sie dann in der Klasse `AbstractProduct` die Auskommentierung der `@JsonTypeInfo` Annotation. Nach Neustart der Webanwendung sollte bei erneutem Zugriff mittels `TestProductRESTService` der Fehler nicht mehr auftreten.

Sonstiges

Greifen Sie nun mit dem Browser auf die URL für das Auslesen aller Produkte zu und vergleichen Sie die Ausgabe mit der in Ü JRS2 gespeicherten. Was hat sich strukturell an der Repräsentation der Daten geändert?

JWS

Projekte:

- `org.dieschnittstelle.jee.esa.jws`
- `org.dieschnittstelle.jee.esa.lib.entities.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.jws.client` (Client)
- `org.dieschnittstelle.jee.esa.ue.jws4` (Codegerüst)
- `org.dieschnittstelle.jee.esa.ue.jws5` (Client-Codegerüst)

Server Runtime:

- `JWS` (JBoss)
- `UE-JWS4` (JBoss)

Ü JWS0 Inbetriebnahme der Implementierungsbeispiele

Aufgabe

Die Kompilierung der Client-Anwendung `.jws.client` erfordert Klassen, die nicht im Quellcode-Repository enthalten sind, sondern auf Basis der durch die server-seitige Anwendung bereit gestellten WSDL Service-Beschreibung generiert werden. Um Fehlermeldungen betreffs nicht verfügbarer Klassen nach Auschecken des Projektverzeichnisses zu unterbinden, wurden die entsprechenden Codestellen auskommentiert. Bevor Sie den Client ausführen können, müssen Sie diese Stellen wieder einkommentieren.

Bearbeitungshinweise

- Starten Sie die Web Applikation `.jws` bzw. `jws-webapp`, die den Service implementiert, durch Ausführung der `JWS` Run-Konfiguration
- Öffnen Sie Klasse `ShowTouchpointSOAPService` im Client-Projekt `.jws.client`.
- Entfernen Sie die Kommentare von den `import`-Deklarationen sowie von der Implementierung der `main()` Methode der Klasse.
- Führen Sie dann die Run-Konfiguration für `JWS: ShowTouchpointSOAPService` aus. Die Client-Anwendung sollte nun gebaut und ausgeführt werden. Das Bauen beinhaltet den Zugriff auf die WSDL unter der in `pom.xml` angegebenen URL und die Generierung und Kompilierung der für den Zugriff auf den Service erforderlichen Klassen.

Ü JWS1 Inspizieren der WSDL

Aufgabe

Vergegenwärtigen Sie sich den Zusammenhang zwischen den Signaturen der in den Implementierungsbeispielen verwendeten Java Methoden und den Operationsdeklarationen der WSDL.

Bearbeitungshinweise

- Starten Sie die Web Applikation `.jws`.
  - Beim Starten sehen Sie in der JBoss Konsole Meldungen der Klasse `org.jboss.wsf.stack.cxf.metadata.MetadataBuilder`, die auf Basis der JAX-WS Annotationen den Web Service `TouchpointCRUDWebService` aufbaut.
- Öffnen Sie die URL `http://localhost:8080/org.dieschnittstelle.jee.esa.jws/TouchpointCRUDWebService?wsdl`. Hier wird Ihnen das generierte WSDL Dokument dargestellt.
- Suchen Sie ausgehend von der <`wsdl:operation`> `readAllTouchpoints`, welchem Datentyp die Rückgabedaten der Operation angehören und in welchem Verhältnis dieser Datentyp zum Rückgabetyp der `readAllTouchpoints()` Methode in `TouchpointCRUDService` steht.

Ü JWS2 WSDL und Client Generierung

Aufgabe

Modifizieren Sie die Annotationen, die für die Deklaration des Web Services verwendet werden, und generieren Sie die client-seitig für den Zugriff auf den Web Service verwendeten Klassen auf Grundlage der WSDL von neuem.

Bearbeitungshinweise

- Führen Sie `mvn install` für `.jws.client` aus.
- Schauen Sie sich im `target/generated-sources/jaxws` Verzeichnis des Client Projekts die Inhalte des Packages `org.dieschnittstelle.jee.esa.jws` an und halten Sie diese ggf. mit einem Screenshot fest.
- Kommentieren Sie in der Web Applikation die Annotation `@XmlTransient` auf dem Attribut `customers` der Klasse `AbstractTouchpoint` aus und starten Sie die Anwendung neu.
- Führen Sie `mvn install` von neuem aus.
- Weshalb wurden die neuen Klassen generiert und weshalb wurden sie im betreffenden Package und nicht in `org.dieschnittstelle.jee.esa.entities.crm` erzeugt?
- Ändern Sie die zugrunde liegenden Klassen in der Web Applikation so, dass sie bei erneuter Client-Generierung in `org.dieschnittstelle.jee.esa.entities.crm` erstellt werden.

Ü JWS3 Erweiterung des bestehenden Web Services

Aufgabe

Fügen Sie dem existierenden Web Service aus den Implementierungsbeispielen eine neue Operation hinzu.

Anforderungen

1. Erweitern Sie die Web Service Implementierung `TouchpointCRUDService` um die Implementierung einer `updateTouchpoint()` Methode.
2. Setzen Sie die `updateTouchpoint()` Methode durch Zugriff auf die gleichnamige Methode aus `TouchpointCRUDExecutor` um.
3. Starten Sie nach Abschluss der Implementierung die Webanwendung neu und generieren Sie die client-seitigen Klassen wie in Ü JWS2.
4. Rufen Sie die nun verfügbare Methode client-seitig in `ShowTouchpointSOAPService` auf.
5. Verifizieren Sie die Funktionsfähigkeit der Implementierung, indem Sie nach Ausführung der Client-Anwendung das GUI unter `http://localhost:8080/org.dieschnittstelle.jee.esa.jws/` aufrufen und überprüfen, ob die Aktualisierung erfolgreich war.

Ü JWS4 Erstellung eines neuen Web Service (6 Punkte)

Aufgabe

Implementieren Sie einen neuen Web Service, der Ihnen CRUD Zugriffsoperationen auf Produkte bereit stellt und dafür die Ausdrucksmittel von JAX-WS verwendet. Hier geht es um die alternative Umsetzung und Bereitstellung eines Services, dessen Funktionalität dem Service aus den Aufgaben JRS2 und JRS3 entspricht. Durch Verwendung von JAX-WS wird für den Service eine implementierungsunabhängige WSDL Beschreibung generiert, auf deren Grundlage Clients in jeglichen Programmiersprachen den Zugriff auf den Service implementieren können.

Anforderungen

1. Verwenden Sie das Codegerüst `.ue.jws4` und implementieren Sie die dort in `ProductCRUDService` vorgesehenen Methoden.
2. Für die Implementierung der Methoden sollen die entsprechenden Methoden der Instanz von `GenericCRUDExecutor` verwendet werden, die Sie aus dem `ServletContext` auslesen können.
3. Der Web Service soll unter dem folgenden Namen bereit gestellt werden: `ProductCRUDWebService`
4. Das Service Interface soll unter dem folgenden Namen bereit gestellt werden: `IProductCRUDService`
5. Der Web Service soll unter einem Namespace bereit gestellt werden, der dem folgenden Java Package entspricht:`org.dieschnittstelle.jee.esa.jws`
6. Die vom Web Service genutzten Klassen `AbstractProduct`, `IndividualisedProductItem` und `ProductType` sollen unter einem Namespace bereit gestellt werden, der ihrem Java Package entspricht, d.h.:`org.dieschnittstelle.jee.esa.entities.erp`.

Bearbeitungshinweise

- ***Anforderung 2***: Dafür müssen Sie, wie in `TouchpointCRUDService` gezeigt, mittels der `javax.annotation.Resource` Deklaration ein Attribut mit Typ `javax.xml.ws.WebServiceContext` deklarieren und daraus in der Implementierung der Operationen den `ServletContext` auslesen.
- ***Anforderung 3***, ***Anforderung 5***: Dafür können Sie die Annotationsattribute `serviceName` und `targetNamespace` verwenden.
- ***Anforderung 6*** Dafür können Sie die Annotation `XmlType` und deren `namespace` Attribut nutzen. Einem Package der Form `a.b.c.d` entspricht ein Namespace der Form `http://b.a/c/d`, d.h. die ersten beiden Segmente des Packages werden umgekehrt als Domain und Top-Level-Domain notiert und alle weiteren Segmente werden als URL-Segmente mit Trennzeichen `/` notiert.

Sonstiges

Punkte werden für diese Aufgabe nur vergeben, wenn auch Aufgabe Ü JWS5 umgesetzt wird.

Ü JWS5 Erstellung eines neuen Web Service Clients (4 Punkte)

Aufgabe

Ergänzen und nutzen Sie eine Testklasse, die unter Verwendung eines generierten Web Service Clients auf den in Ü JWS4 erstellten Web Service zugreift und dessen Operationen aufruft. Dafür können Sie die als Resultat Ihrer Umsetzung von JWS4 durch den Server bereit gestellte WSDL Beschreibung des Services verwenden. Diese ermöglicht die automatische Generierung client-seitiger Zugriffsklassen wie auch die Generierung der Klassen des Datenmodells für die zwischen Client und Server zu übermittelnden Inhalte.

Anforderungen

1. Verwenden Sie das Codegerüst `.ue.jws5` und kommentieren Sie den auskommentierten Quellcode in `TestProductSOAPService` und `ProductCRUDSOAPClient` ein. Wenn die Anforderungen von JWS4 vollständig umgesetzt worden sind, sollten nach Generieren der client-seitigen Klassen alle erforderlichen Imports verfügbar sein.

2. Ergänzen Sie den Code des Konstruktors in `ProductCRUDSOAPClient` so, dass `serviceProxy` mit einem geeigneten Wert für den Zugriff auf den Web Service instantiiert wird.

3. Führen Sie dann die JUnit Testklasse `TestProductSOAPService` aus.

4. Dem verwendeten Projekt `.ue.jws5` dürfen keine Abhängigkeiten zu anderen `.esa.*` Projekten oder zu dem Projekt hinzugefügt werden, in dem Sie JWS4 umgesetzt haben.

Bearbeitungshinweise

- ***Anforderung 1*** Um die Client-seitgen Klassen generieren zu können, müssen Sie in der `pom.xml` Datei des Projekts die URL der WSDL Beschreibung des in JWS4 entwickelten Services angeben. Wenn Sie die Änderung vorgenommen haben, können Sie in der *Maven Projects* Ansicht der Entwicklungsumgebung *ue-jws5→Lifecycle* auswählen und durch Doppelklick auf *install* den Generierungsvorgang starten. Danach stehen Ihnen die erforderlichen Klassen zur Verfügung.

- Testen Sie Ihre Implementierung inklusive der Umsetzung von JWS4 durch Ausführung von `TestProductSOAPService`.

WSV

Projekte:

- `org.dieschnittstelle.jee.esa.wsv.client` (Client-Codegerüst)
- `org.dieschnittstelle.jee.esa.lib.entities.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.crm` (SHARED)

Server Runtime:

- JRS (Tomcat)

Ü WSV1 JAX-RS Client (14 Punkte)

Aufgabe

Implementieren Sie einen `InvocationHandler` der Ihnen für ein JAX-RS annotiertes Interface den Zugriff auf einen REST Service ermöglicht, welcher das Interface bereit stellt. Diese Übung verleiht Ihnen Einblick in die Mechanismen, auf denen die Implementierungen generischer Client-Frameworks für JAX-RS, aber auch für JAX-WS oder EJB, basieren.

Anforderungen

1. Der Funktionsumfang des `InvocationHandler` soll es Ihnen ermöglichen, die 5 CRUD Operationen aufzurufen, die in `ITouchpointCRUDService` mittels JAX-RS Annotationen deklariert werden. **(11 Punkte)**

   - Ihre Implementierung des `InvocationHandler` darf keine Abhängigkeiten zu Klassen der Implementierungsbeispiele für `esa.wsv` mit Ausnahme der Klassen für die Umwandlung von JSON Objekten in Java-Objekte enthalten, d.h. Abhängigkeiten zu `ITouchpointCRUDService`, `StationaryTouchpoint` etc. sind dort nicht erlaubt.
   - Verifizieren Sie die Funktionsfähigkeit Ihrer Implementierung durch Zugriff auf den JAX-RS Web Service für Touchpoints aus dem `.jrs` bzw. `jrs-webapp` Projekt. Verwenden Sie dafür die Klasse `AccessRESTServiceWithInterpreter`.

2. Ersetzen Sie in allen Methodensignaturen der lokalen Deklaration von `ITouchpointCRUDService` im Projekt `.wsv.client` den Typ `StationaryTouchpoint` durch `AbstractTouchpoint`. Stellen Sie dann die volle Funktionsfähigkeit von `AccessRESTServiceWithInterpreter` wieder her. **(3 Punkte)**

Bearbeitungshinweise

- Ein 'Gerüst' für die Implementierung finden Sie im Projekt `.wsv.client`. Dort wird das Service Interface `ITouchpointCRUDService` lokal deklariert. Um die `update` Methode ausführen zu können, müssen Sie diese noch entsprechend der – nicht selbständig bepunkteten – Aufgabe JRS1 server-seitig umsetzen.
- Darin finden Sie im Package `.esa.wsv.interpreter` u.a. ein Codegerüst für einen `InvocationHandler` in `JAXRSClientInterpreter` sowie die Klassen `JSONObjectSerialiser` und `JSONObjectMapper`, mit der Sie die Abbildung von Instanzen anwendungsspezifischer Klassen auf JSON Objekte vornehmen können und umgekehrt.
- Für die Ausführung von HTTP Requests können Sie in der Implementierung von `JAXRSClientInterpreter` die Apache HTTP Client API nutzen, die Sie auch in SER verwendet haben. Die Imports hierfür sind bereits im `pom.xml` vorhanden.
- Um zu überprüfen, ob der Rückgabetyp einer Methode ein generischer Typ ist, können Sie die Methode `getGenericReturnType()` auf `java.lang.reflect.Method` verwenden. Dies benötigen Sie für die Behandlung von `List`-wertigen Rückgabetypen wie in `readAllTouchpoints()`.

- ***Anforderung 2*** Dafür müssen Sie eine Ergänzung an der mit *TODO* markierten Stelle in `JSONObjectMapper` vornehmen sowie ggf. die Annotation `JsonTypeInfo` auf `AbstractTouchpoint` einkommentieren.

EJB

Projekte:

- `org.dieschnittstelle.jee.esa.ejb`
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.crm`
- `org.dieschnittstelle.jee.esa.lib.entities.erp`
- `org.dieschnittstelle.jee.esa.ejb.client` (Client)

Server Runtime:

- EJB (JBoss)

Ü EJB1 Implementierungsbeispiele

Aufgabe

Hier wird in Kürze der Aufbau der Implementierungsbeispiele beschrieben.

Ausgangssituation

- Die Beispielanwendung umfasst ein EJB Modul `.ejb.ejbmodule.crm`, das die Verwendung von `Stateful`, `Stateless` und `Singleton` Sessions Beans illustriert – im zweiten EJB Modul – `.ejb.ejbmodule.erp` – werden Sie die heutige Übungsaufgabe umsetzen.
- Die beiden Module sind zusammen mit den Paketen des Datenmodells – `.lib.entities.crm` und `.lib.entities.erp` – zu dem EAR Archiv `org.dieschnittstelle.jee.esa.ejb` paketiert, das Sie in JBoss zur Ausführung bringen können. Dieses Archiv wird im Projekt `.ejb` bzw. `ejb-earapp` deklariert.
- Der Remote Zugriff auf die EJBs erfolgt aus mehreren einfachen Java SE Anwendungen im Projekt `.ejb.client` und verwendet den JBoss-spezifischen Zugriff auf Remote EJBs via JNDI.
- Umgesetzt sind u.a. die folgenden client-seitigen Use Cases:
  - Erstellung von Touchpoints (`TotalUsecase.createTouchpoints()`)
  - Erstellung von Kunden (`TotalUsecase.createCustomers()`)
  - Durchführung von Kampagnen (`TotalUsecase.prepareCampaigns()`)
  - Befüllung eines Warenkorbs und (partielle) Durchführung eines Kaufs (`TotalUsecase.doShopping()`)
  - Nachverfolgen der Einkäufe eines Kunden (`TotalUsecase.doShopping()`, `TotalUsecase.showTransactions()`)
- Für `TotalUsecase.doShopping()` wird die client-seitige Implementierung einer `ShoppingSession` verwendet, deren Funktionsumfang partiell dem Beispiel im Skript entspricht.

Inspizieren der Anwendung

- Kommentieren Sie die Annotation `@Startup` auf `CampaignTrackingSingleton` aus und aktualisieren Sie die Anwendung. Führen Sie dann `PrepareCampaigns` aus. Suchen Sie im JBoss Log nach der Logmeldung, die bei Initialisierung der Bean geschrieben wird. Wann erfolgt die Initialisierung?
- Kommentieren Sie `@Startup` wieder ein und aktualisieren Sie die Anwendung. Enthält das JBoss Log nun bereits die Initialisierungsmeldung?
- Ändern Sie in `ejb-jar.xml` im Projekt `.ejb.ejbmodule.crm` den Timeout für `ShoppingCartStateful` auf 100 Millisekundenund führen Sie nach Aktualisierung der Anwendung `TotalUsecase` aus. Warten Sie an der Step-Stelle nach Erhalt der Logmeldung '`got shopping cart bean: ...`' einen kurzen Moment und setzen Sie die Ausführung fort. Was passiert? Versuchen Sie, das Verhalten anhand des JBoss Logs nachzuvollziehen.

Ü EJB2 Implementieren von `StockSystemRemote` (5 Punkte)

Aufgabe

Implementieren Sie auf einfach(st)e Weise das Interfaces `org.dieschnittstelle.jee.esa.ejb.ejbmodule.erp.StockSystemRemote` als Session Bean eines geeigneten Typs. Hier können Sie – nach intensiver Beschäftigung mit Web APIs und Web Services in den vorangegangenen Übungen – die Ausdrucksmittel von EJBs zur Umsetzung von clientseitigen Zugriffen auf server-seitige Anwendungsfunktionen kennen lernen, die zur Verteilung von Funktionen der Geschäftslogik von Anwendungsystemen auf mehrere Laufzeitumgebungen genutzt werden können.

Diese Übung ist vorläufig, da die vorzunehmende EJB Implementierung in JPA4 modifiziert und ersetzt werden wird. Als Einstieg in die Beschäftigung mit EJBs sei eine zumindest partielle Bearbeitung, bestehend aus ***Anforderung 5*** und ***Anforderung 6*** jedoch empfohlen. Für ein von der Abfolge der Übungen abweichendes Vorgehen bei der Umsetzung der Aufgaben zu EJB und JPA beachten Sie bitte den Punkt 'Sonstiges' unten.

Anforderungen

1. Implementiert werden soll die Methode `addToStock()`.
2. Implementiert werden soll außerdem die Methode `getUnitsOnStock()`.
3. Die anderen Methoden aus `StockSystemRemote` benötigen Sie für die heutige Übungen noch nicht.
4. Verwenden Sie für die Implementierung als einfachen 'Datenspeicher' z.B. ein Instanzattribut auf der EJB Implementierung, das mittels einer `Map` Assoziationen zwischen Produktnamen, Ids von `PointOfSale` sowie Instanzen von `StockItem` repräsentiert. In Ü JPA4 werden Sie diese Implementierung um den Zugriff auf eine Datenhaltungsschicht erweitern.
5. Versehen Sie das `StockSystemRemote` Interface selbst mit einer geeigneten Annotation, die Ihnen den Zugriff auf die EJB von außen **als Remote EJB** ermöglicht. Wenn Sie anstelle einer Remote EJB einen REST Service verwenden möchten, dann implementieren Sie das Interface `StockSystemRESTService` und berücksichtigen Sie die dort unter *TODO* vermerkten Bearbeitungshinweise.
6. Greifen Sie auf die implementierte Session Bean aus der Klasse `StockSystemClient` im Client Projekt zu und führen Sie die Client-Klasse `ShowStockSystem` aus.

Bearbeitungshinweise

- ***Anforderung 4*** `StockItem` repräsentiert die Menge der verfügbaren Exemplare eines Produkts an einer Verkaufsstelle und befindet sich im Projekt `.lib.entities.erp`.
- ***Anforderung 6*** Dafür müssen Sie im Konstruktor der Klasse das Instanzattribut `ejbProxy` durch Verwendung der `getProxy()` Methode von `EJBProxyFactory` instantiieren.

Sonstiges

Die Übungen zu EJB und JPA können Sie auch entsprechend dem folgenden Vorgehen bearbeiten, das sich an der voraussichtlichen Abfolge der Demos im SU der Veranstaltung orientiert:

1. Ü EJB2: Codegerüst und Client für `StockSystem`, ohne weitere Implementierung von `StockSystem`

   Test: `ShowStockSystem`

2. Ü JPA2: Datenmodellierung nur für `IndividualsedProductItem`, ohne `Campaign`

   Test: Starten von JBoss; Anpassungen, falls Fehlschlag

3. Ü JPA3: Umsetzung der CRUD EJB für `AbstractProduct`

   Test: `ShowStockSystem`

4. Ü JPA4: Umsetzung der CRUD EJB für `StockItem`, Verwendung in `StockSystem`

   Test: `ShowStockSystem`; falls erfolgreich ausführbar: `TestStockSystem`

5. Fortsetzung Ü JPA2, JPA3: Behandlung von `Campaign`

   Test: `ShowStockSystem` mit Erstellung von Campaigns in `createProducts()` (derzeit aaskommentiert); falls erfolgreich ausführbar: `TestProductCRUD`

JPA

Projekte:

- `org.dieschnittstelle.jee.esa.ejb`
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.crm`
- `org.dieschnittstelle.jee.esa.lib.entities.erp`
- `org.dieschnittstelle.jee.esa.ejb.client` (Client)

Server Runtime:

- EJB (JBoss)

Ü JPA1 Inspizieren der Beispiele

Aufgabe

Vergegenwärtigen Sie sich einige wichtige Merkmale von JPA anhand einer Manipulation der Implementierungsbeispiele.

-] Kaskadierung
  - – Kommentieren Sie in `CustomerTransaction` die `@OneToMany` Annotation mit Attributen aus und entfernen Sie den Kommentar von der einfachen Annotation ohne Attribute. Aktualisieren Sie die Anwendung und führen Sie dann `TotalUsecase` aus.
  - ? Was für einen Fehler bekommen Sie im Schritt nach der Befüllung des `ShoppingCart` bei der Ausführung von `ShoppingSession.purchase()` und weshalb?
  - – Entfernen Sie dann in `CustomerTransactionCRUDStateless` die Kommentare von der `for`-Schleife in der `createTransaction()` Methode. Aktualisieren Sie die Anwendung und führen Sie dann `TotalUsecase` noch einmal aus.
  - ? Weshalb kann `createTransaction()` jetzt problemlos ausgeführt werden?
- *Lazy vs. Eager* Laden assoziierter Instanzen
  - – Bei der Ausführung des letzten Schrittes von `TotalUsecase` tritt bisher immer ein Fehler auf.
  - ? Versuchen Sie, den Grund dafür herauszufinden, indem Sie einen Blick auf das `transactions` Attribut der `Customer` Klasse werfen.
  - – Tauschen Sie dann die aktive und die auskommentierte `@OneToMany` Assoziation auf `transactions` in `Customer` aus, aktualisieren Sie die Anwendung und führen Sie `TotalUsecase` noch einmal aus.
  - ? Tritt der Fehler noch auf? Wenn nein, warum nicht?
  - ? Welche Probleme könnten bei hohen Zugriffszahlen auf `readCustomer()` mit der jetzigen Umsetzung auftreten?
  - – Machen Sie die Änderung bezüglich `transactions` wieder rückgängig.

Ü JPA2 Datenmodellierung (6 Punkte)

Aufgabe

Erweitern Sie den bestehenden Java Quellcode um JPA Annotationen, die Ihnen die Persistierung von Instanzen der Klassen des Datenmodells ermöglichen. Sie können sich anhand dieser Übung u.a. die Spezifika von Datenbankschemata vergegenwärtigen, die aus UML Modellen abgeleitet werden können, aus der Umsetzung eines Datenmodells in Java jedoch z.T. nicht mehr hervorgehen und daher durch Annotationen ausgedrückt werden müssen.

Anforderungen

1. Deklarieren Sie die Klassen des Packages `org.dieschnittstelle.jee.esa.entities.erp` als JPA Entities – ausgenommen davon sind die Primary Key Klasse `ProductAtPosPK` und die Enumeration `ProductType`.

2. Wählen Sie geeignete Beziehungen der in JPA definierten Typen – `OneToOne`, `ManyToOne`, etc. – für die Assoziationen zwischen den Klassen.

3. Sehen Sie vor, dass die Primärschlüssel für Instanzen von `AbstractProduct` unabhängig von den Schlüsseln der anderen Entities generiert werden.

Bearbeitungshinweise

- Verifizieren Sie Ihr Datenmodell vor Durchführung weiterer Schritte anhand Neustartens der Anwendung – falls die JPA Annotationen in schwerwiegender Weise nicht korrekt oder unvollständig sind, wird der Neustart der Anwendung fehlschlagen.

Ü JPA3 DAO für `AbstractProduct` (7 Punkte)

Aufgabe

Erstellen Sie eine neue EJB Implementierung für das Interface `ProductCRUDRemote`. Hier können Sie die client- und server-seitig erforderlichen Maßnahmen zur Bereitstellung und Nutzung von EJBs rekapitulieren und die Grundfunktionen des `EntityManagers` kennen lernen, der Ihnen auf Grundlage eines JPA-annotierten Datenmodells den lesenden und schreibenden Zugriff auf Datenbankinhalte ermöglicht.

Anforderungen

1. Deklarieren Sie das Interface **als `@Remote` EJB Interface oder als REST Service**.
2. Implementieren Sie das Interface als `Stateless` EJB.
3. Instantiieren Sie das Attribut `ejbProxy` in der Clientklasse `ProductCRUDClient` durch Verwendung von `EJBProxyFactory` und verwenden Sie `ejbProxy` wie in den bisher auskommentierten Anweisungen vorgesehen.

   ! Kommentieren Sie in `ProductCRUDClient` die manuelle ID Zuweisung in `createProduct` aus und entfernen Sie die Kommentare auch von den darauf folgenden Methoden.

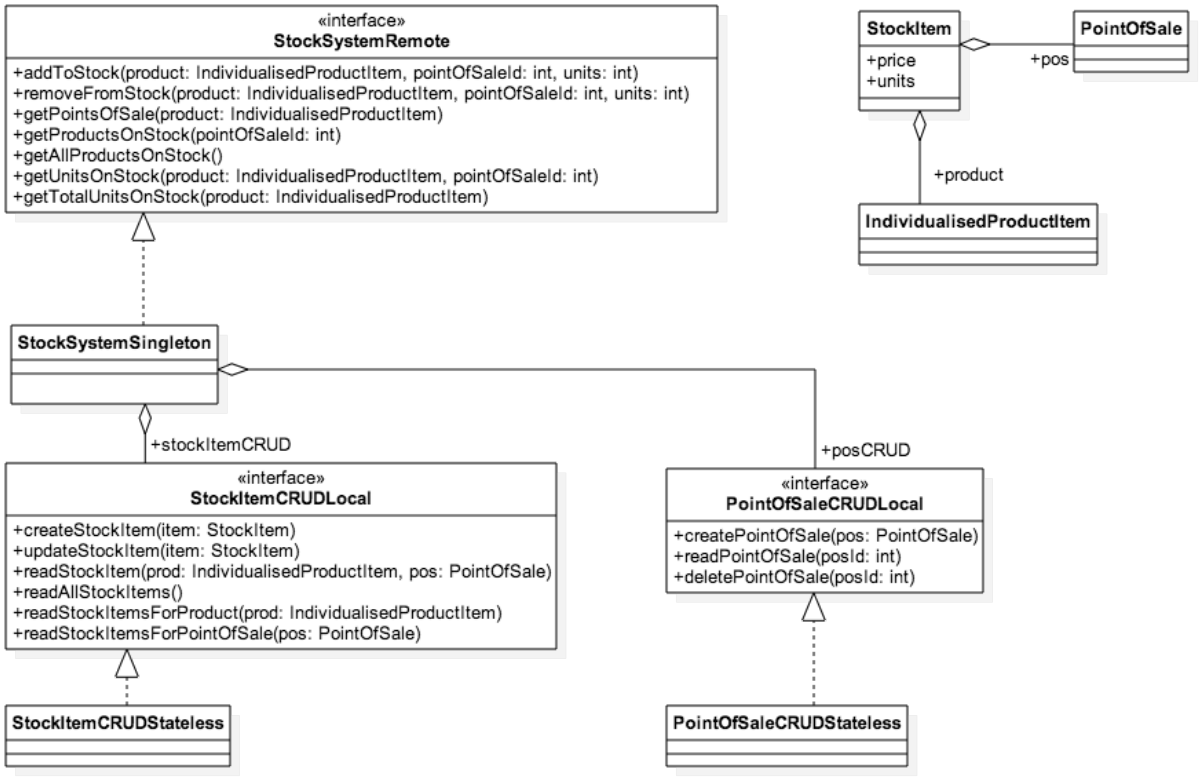
4. Verifizieren Sie Ihre Implementierung anhand der Testklasse `TestProductCRUD`.

Ü JPA4 DAO für `StockItem` (9 Punkte)

Aufgabe

Erstellen Sie eine neue EJB, die Ihnen die CRUD Operationen für Instanzen von `StockItem` bereitstellt und verwenden Sie diese EJB in Ihrer Implementierung von `StockSystemRemote` aus Ü EJB2. Alle Methoden des Interfaces sollen in der EJB implementiert werden. In der Umsetzung dieser Aufgabe bringen Sie noch einmal umfassend Ihre Kenntnisse in der Verwendung von EJB und JPA zur Anwendung.

Das folgende UML Diagramm stellt Funktionalität und Abhängigkeiten der für die Umsetzung zu verwendenden Komponenten dar:



Anforderungen

1. Verwenden Sie für die Umsetzung das Interface `StockItemCRUDLocal` im Projekt `.ejb.ejbmodule.erp`
2. Implementieren Sie dieses Interface in einer geeigneten EJB.
3. Greifen Sie auf die EJB in Ihrer Implementierung von `StockSystemRemote` zu. Der Zugriff auf `readAllStockItems()` ist dabei nicht zulässig.
4. Testen Sie die Funktionsfähigkeit Ihrer Implementierung mit der Testklasse `TestStockSystem`.

Bearbeitungshinweise

- Die Interfaces `StockSystemRemote` und `StockItemCRUDLocal` enthalten ausführliche Hinweise zur Implementierung der EJBs entsprechend den Anforderungen.
- ***Anforderung 2***: Um die Methode `getUnitsOnStock()` aus `StockSystemRemote` auf der Ebene von `StockItemCRUDStateless` umzusetzen, können Sie in `StockItemCRUDImpl` die Ausdrucksmittel der JPA QL nutzen, vergleichbar der Implementierung der `readAll...()` Methoden aus`CustomerTransactionCRUDStateless`.
- ***Anforderung 2***: Um für eine gegebene Kombination von `AbstractProduct` und `PointOfSale` auf die persistierten `StockItem` Instanzen zuzugreifen, können Sie der `find()` Methode von `EntityManager` eine Instanz von `ProductAtPosPK` als Identifikator übergeben.
- ***Anforderung 3***: Zur Ermittlung eines `PointOfSale` für eine gegebene `pointOfSaleId` kann Ihre `StockSystemRemote` Implementierung das Interface `PointOfSaleCRUDLocal` verwenden und das daraus ausgelesene `PointOfSale` Objekt an `StockItemCRUDLocal` übergeben. Auch auf `PointOfSaleCRUDLocal` können Sie via Dependency Injection zugreifen.

ADD

Projekte:

- `org.dieschnittstelle.jee.esa.ejb`
- `org.dieschnittstelle.jee.esa.ejb.webapp`
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.crm`
- `org.dieschnittstelle.jee.esa.lib.entities.erp`
- `org.dieschnittstelle.jee.esa.ejb.client` (Client)
- `org.dieschnittstelle.jee.esa.ue.add2` (ADD, Client-Codegerüst)
- `org.dieschnittstelle.jee.esa.ue.add3` (ADD, Client-Codegerüst)

Server Runtime:

- EJB (JBoss)

Ü ADD1 Inspizieren der Implementierungsbeispiele

Aufgabe

Vergegenwärtigen Sie sich den vermittelten Lernstoff anhand einer Betrachtung der Implementierungsbeispiele.

1. Web Services und Lazy Loading
   - Kommentieren Sie die im Projekt `.lib.entities.crm` auf `AbstractTouchpoint` die `@JsonIgnore` Annotation auf dem `customers` Attribut aus und aktualisieren Sie die EJB Anwendung in JBoss.
   - Führen Sie in `.ejb.client` im `demos` Package die Klasse `CreateTouchpoints` aus.
   - Rufen Sie im Browser die URL `http://localhost:8080/org.dieschnittstelle.jee.esa.ejb.webapp/api/touchpoints` auf.
     ? Was für ein Fehler ist in den JBoss Logmeldungen zu sehen und weshalb?
   - Machen Sie die Änderung wieder rückgängig.
2. Transaktionen
   - Die von dem o.g. Service-Client aufgerufene Methode `createTouchpoint()` erzeugt zunächst eine `PointOfSale` Instanz und dann eine Instanz von `AbstractTouchpoint`, die die `id` der erzeugten Instanz als Wert des `erpPointOfSaleId` Attributs gesetzt bekommt.
   - Führen Sie den Service-Client aus und versuchen Sie, anhand der Logs zu ermitteln, wann eine Persistierung der `PointOfSale` Instanz erfolgt.
   - Kommentieren Sie dann das `@TransactionAttribute` auf `PointOfSaleCRUDStateless` ein und greifen Sie nach Aktualisierung des EAR erneut mit einem der Clients auf die Anwendung zu.
     ? Hat sich eine Änderung gegenüber dem Ausgangszustand ergeben? Wenn ja, warum?
   - Ändern Sie nun in `TouchpointCRUDStateless` in der `create()` Methode den Ausdruck `false` im `if`-Statement auf `true`, sodass bei jedem Aufruf der Methode eine Exception geworfen wird, und greifen Sie nach Aktualisierung des EAR wieder auf die Anwendung zu.
     ? Findet nun ebenfalls eine Persistierung des `PointOfSale` statt? Begründen Sie das beobachtete Verhalten.
   - Kommentieren Sie dann das `@TransactionAttribute` auf `PointOfSaleCRUDStateless` wieder aus und greifen Sie (nach Aktualisierung) erneut auf die Anwendung zu.
     ? Wird `PointOfSale` nun persistiert?
     ? Welche Vor- und Nachteile bringen die beiden Varianten mit bzw. ohne `@TransactionAttribute` mit sich, wenn man annimmt, dass die Mengen der von der Anwendung verwalteten `AbstractTouchpoint` und `PointOfSale` Instanzen deckungsgleich sein sollen?
   - Stellen Sie die Funktionsfähigkeit der Anwendung durch `false`-n der Exception in `TouchpointCRUDStateless` wieder her.

Ü ADD2 EJB als JAX-WS Web Service

Aufgabe

Machen Sie das DAO für `AbstractProduct`, das Sie in Ü JPA3 der Veranstaltung als EJB für das Interface `ProductCRUDRemote` implementiert haben, mittels JAX-WS als Web Service verfügbar.

Anforderungen

1. Verwenden Sie das Codegerüst `.ue.add2`.
2. Als Endpoint Interface des Web Service soll `ProductCRUDRemote` verwendet werden.
3. Die Bereitstellung als Service soll allein mittels Annotationen auf der bestehenden EJB und dem `@Remote` Interface und ohne Erstellung zusätzlicher Klassen erfolgen.
4. Generieren Sie die client-seitig erforderlichen Klassen für den Zugriff durch Anpassung von `pom.xml` und Ausführung von `mvn install`.
5. Modifizieren/Ergänzen Sie die Klassen im Package `.ue.add2.junit` an den mit `TODO:` markierten Stellen und modifizieren Sie ggf. Ihren Service, bis die Testklasse `TestProductEJBSOAPService` erfolgreich ausgeführt werden kann.
6. Alle vom Client-Projekt benötigten Klassen (mit Ausnahme von JUnit) müssen durch die Ausführung von `mvn install` unter Verwendung der server-seitigen WSDL bereit gestellt werden. Das Client-Projekt darf keine direkten Abhängigkeiten zu den server-seitig verwendeten Komponenten enthalten.

Bearbeitungshinweise

- ***Anforderung 3***: Hinweise darauf finden Sie im vorliegenden Veranstaltungsskript und in den Implementierungsbeispielen in der Klasse `TouchpointAccessStateless`.

Ü ADD3 `StockSystem` als JAX-WS Web Service (6 Punkte)

Aufgabe

Machen Sie die EJB für `StockSystemRemote`, die Sie in Ü EJB2 erstellt und in Ü JPA4 erweitert haben, mittels JAX-WS als Web Service verfügbar. Die Übung zeigt, wie ohne zusätzliche serverseitige Implementierungskomponenten Funktionalität, die bereits in EJBs implementiert wurde, als Web Service bereit gestellt werden kann.

Anforderungen

1. Die Bereitstellung als Service soll allein mittels Annotationen auf der bestehenden EJB und deren `@Remote` Interface ohne Erstellung zusätzlicher Klassen erfolgen.
2. Verwenden Sie das Codegerüst `.ue.add3`.
3. Als Endpoint Interface des Web Service soll `StockSystemRemote` verwendet werden.
4. Generieren Sie die client-seitig erforderlichen Klassen für den Zugriff durch Anpassung von `pom.xml` und Ausführung von `mvn install`.
5. Ergänzen Sie die Testklasse `TestStockSystemSOAPService` im Projektgerüst, bis sie ausgeführt werden kann.
6. Führen Sie eine der Anwendungen aus dem Projekt `.ejb.client` aus, welche auf `StockSystem` zugreifen und Lagerbestände erstellen, z.B. `TotalUsecase` oder `ShowStockSystem`.
7. Führen Sie dann die Testklasse `TestStockSystemSOAPService` unter Verwendung der Run-Konfiguration `ADD: TestStockSystemSOAPService` aus. Wenn die server-seitige Implementierung von `StockSystem` und die Bereitstellung als Web Service korrekt sind, sollten keine Fehler auftreten.
8. Es gelten außerdem die in ***Anforderung 6*** aus ADD2 genannten Anforderungen.

Bearbeitungshinweise

- ***Anforderung 1***, ***Anforderung 3***: Ein Beipiel für die Deklaration einer EJB als Web Services und die Verwendung eines Endpoint Interfaces finden Sie im Veranstaltungsskript und in der Klasse `TouchpointAccessStateless`.

Ü ADD4 Nutzung von Transaktionen (2 Punkte)

Aufgabe

Angenommen wird, dass Sie `ShoppingSession` entsprechend Ü PAT1 und PAT2 server-seitig als Stateful EJBs entwickelt haben. Unterbinden Sie nun die Nutzung verwendeter EJB Methoden außerhalb von `purchase()` durch Deklaration ihres transaktionalen Verhaltens. Damit können Sie sich wesentliche Unterschiede zwischen lokalen und remote Zugriffen auf EJBs im Hinblick auf Transaktionen vergegenwärtigen.

Anforderungen

1. Die Methode `removeFromStock()` Ihrer `StockSystemRemote` Implementierung soll nur innerhalb einer bereits bestehenden Transaktion ausgeführt werden können.

2. Wird im Zuge der Durchführung von `purchase()` ein fachlicher oder technischer Sachverhalt detektiert, der der Durchführung der Operation entgegen steht – insbesondere die Nicht(mehr)verfügbarkeit von Produkten oder Kampagnen oder das Auftreten einer Exception – soll die Methode durch eine `ShoppingException` beendet werden. Diese Klasse finden Sie im Projekt `.ejb.ejbmodule.crm`. Sie deklariert u.a. in einer Enumeration eine Menge möglicher Abbruchgründe.

3. Verifizieren Sie Ihre Implementierung durch Anpassung und Ausführung von `TotalUsecase` bzw. Ausführung von `TestStockSystem`.

Bearbeitungshinweise

- ***Anforderung 1***, ***Anforderung 3***: Das geforderte Transaktionsverhalten von `removeFromStock()` können Sie anhand eines Durchlaufs von `TestStockSystem` oder durch Ausführung des Tests aus ADD3 verifizieren. Da hier der remote EJB Client verwendet wird, sollte bei Erfüllung von ***Anforderung 1*** die erfolgreiche Ausführung des Testcases nicht möglich sein. Kommentieren Sie die vorgenommene Änderung für weitere Tests mit `TestStockSystem` ggf. wieder aus.

- ***Anforderung 2***, ***Anforderung 3***: Das Auftreten der Exception bei Ausführung von `purchase()` können Sie in `TotalUsecase` durch Setzen des Attributs `provokeErrorOnPurchase` auslösen. `TotalUsecase` selbst wird dann immer beim Aufruf von `purchase()` fehlschlagen; bei Ausführung des Testcases `TestShoppingSession` wird die betreffende Einstellung überschrieben, d.h. das Verhalten der Testcases wird durch die Änderung nicht beeinträchtigt.

PAT

Projekte:

- `org.dieschnittstelle.jee.esa.ejb`
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.eb.ejbmodule.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.crm`
- `org.dieschnittstelle.jee.esa.lib.entities.erp`
- `org.dieschnittstelle.jee.esa.ejb.client` (Client)

Server Runtime:

- EJB (JBoss)

Ü PAT1 `ShoppingSession` als SessionFacade (6 Punkte)

Aufgabe

Übertragen Sie die Funktionalität der `ShoppingSession` Klasse im Clientprojekt in eine server-seitige EJB, die das `SessionFacade` Pattern umsetzt. Diese und die folgende Übung dienen der Wiederholung der Anwendung von Ausdrucksmitteln für EJBs im Rahmen der Umsetzung eines relevanten Entwurfsmusters für Anwendungen mit komplexer server-seitiger Geschäftslogik.

Anforderungen

1. Verwenden Sie für die Umsetzung die Interfaces und Klassen / Codegerüste aus den Packages `org.dieschnittstelle.jee.esa.ejb.client.shopping` und `org.dieschnittstelle.jee.esa.ejb.ejbmodule.crm.shopping`, die Sie in den Projekten `.ejb.client` bzw. `.ejb.ejbmodule.crm` finden.
2. Übertragen Sie die Funktionalität, die die Klasse `ShoppingSession` aus der Client-Anwendung implementiert, in eine EJB, die das Interface `ShoppingSessionFacadeRemote` implementiert.
3. Nutzen Sie anstelle des Auslesens der von EJBs via JNDI, das in `ShoppingSession` verwendet wird, in Ihrer EJB die Möglichkeit der Dependency Injection mittels der `@EJB` Annotation.
4. Erstellen Sie lokale Interfaces für die von Ihrer `ShoppingSessionFacade` EJB genutzten EJBs und lassen Sie sie auf diese EJBs zugreifen.
5. Greifen Sie aus der Client-seitigen Klasse `ShoppingSessionFacadeClient` auf die neue EJB zu.
6. Verwenden Sie in `TotalUsecase.doShopping()` die Klasse `ShoppingSessionFacadeClient` anstelle von `ShoppingSession`.
7. **Auch für die Umsetzung diese Aufgabe können Sie anstelle eines remote EJB Interfaces für `ShoppingSessionFacadeRemote` und deren Implementierung als `@Stateful` EJB einen REST Service verwenden.** Dafür benötigen Sie u.a. eine neue Entity-Klasse, die den `Customer`, den `Touchpoint` sowie den Warenkorb umfasst. Die `ids` der Instanzen dieser Klasse können Sie client-seitig in `ShoppingSessionFacadeClient` als Instanzattribut festhalten. Anhaltspunkte für die Umsetzung können Sie den Klassen `ShoppingCartClient` sowie `ShoppingCartRESTServiceImpl` entnehmen.

Bearbeitungshinweise

- ***Anforderung 4*** diese Anforderung wird durch den JUnit Testcase für PAT2 nicht überprüft.
- ***Anforderung 6***: die betreffende Stelle ist im Quellcode mit Hinweis auf PAT1 markiert – sie müssen lediglich das Attribut `useShoppingSessionFacade` auf `true` setzen. Diese Änderung können Sie für das weitere Übungsprogramm beibehalten.

Ü2 PAT2 Erweiterte `ShoppingSessionFacade` (7 Punkte)

Aufgabe

Erweitern Sie die Implementierung der in PAT1 umgesetzten `ShoppingSessionFacade` EJB.

Anforderungen

1. Implementieren Sie die in der `purchase()` Methode von `ShoppingSession` vorgesehene Methode `checkAndRemoveProductsFromStock()` in Ihrer Implementierung von `ShoppingSessionFacadeRemote` entsprechend den im Code vorgegebenen Instruktionen.

2. Erstellen Sie in der Implementierung von `ShoppingSessionFacadeRemote` eine `@PreDestroy` Methode und überprüfen Sie hier, ob die Session abgeschlossen ist. Andernfalls erstellen Sie eine `CustomerTransaction` und übertragen Sie diese an die `CustomerTracking` Bean. **Wenn Sie einen REST Service für Shopping Session verwenden**, dann deklarieren Sie anstelle dessen eine mit `@Schedule` gekennzeichnete Methode, die die Überprüfung für alle Shopping Sessions durchführt und 'abgelaufene' Sessions aus dem persistenten Datenspeicher entfernt.

3. Setzen Sie in `ejb-jar.xml` einen geeigneten Timeout für die `ShoppingSessionFacade` EJB, mit dem Sie die Ausführung Ihrer `@PreDestroy` Methode überprüfen können. Wenn Sie REST Services nutzen, dann deklarieren Sie den Timeout als `env-entry` in `ejb-jar.xml`.

4. Testen Sie Ihre Implementierung anhand der Testklasse `TestShoppingSession`.

Bearbeitungshinweise

- ***Anforderung 1*** Dafür muss Ihre `ShoppingSessionFacade` EJB zusätzlich zu den in PAT1 verwendeten EJBs noch Abhängigkeiten zur `ProductCRUD` EJB sowie zur `StockSystem` EJB deklarieren.

- ***Anforderung 2*** Um zu überprüfen, ob eine Shopping Session abgeschlossen ist oder nicht, können Sie ein privates Attribut in Ihrer Implementierung von `ShoppingSessionFacadeRemote` verwenden, das Sie nach Durchlaufen von `purchase` auf einen geeigneten Wert setzen.

- ***Anforderung 2***, ***Anforderung 3*** Ein Beispiel für die Nutzung des Java EE Timer Service mittels Verwendung von `@Schedule` und für das Setzen anwendungsspezifischer Konfigurationsparameter mittels `env-entry` finden Sie in `ShoppingCartRESTServiceImpl` bzw. den diesbezüglichen Einstellungen in `ejb-jar.xml`.

JSF

Projekte:

- `org.dieschnittstelle.jee.esa.jsf`/`org.dieschnittstelle.jee.esa.jsf.cdi`
- `org.dieschnittstelle.jee.esa.ejb`
- `org.dieschnittstelle.jee.esa.ejb.webapp`
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.crm` (SHARED)
- `org.dieschnittstelle.jee.esa.ejb.ejbmodule.erp` (SHARED)
- `org.dieschnittstelle.jee.esa.lib.entities.crm`
- `org.dieschnittstelle.jee.esa.iib.entities.erp`
- `org.dieschnittstelle.jee.esa.ue.jsf5` (Codegerüst)

Server Runtime:

- `EJB+JSF` / `EJB+JSF-CDI` (JBoss, Implementierungsbeispiele)
- `EJB+JSF` / `UE-JSF5` (JBoss, Codegerüst)

Ü JSF1 Verwendung von Products EJB (3 Punkte)

Aufgabe

Verwenden Sie in `ProductsViewController` das lokale Interface der `ProductCRUD` EJB aus PAT2 und lesen Sie daraus die darzustellenden Produkte aus.

Anforderungen

1. Angezeigt werden sollen alle Produkte vom Typ `IndividualisedProductItem`, die in der Datenbank vorhanden sind.

Bearbeitungshinweise

- Entfernen Sie die manuelle Hinzufügung von Produkten in der `startup()` Methode der Bean.

Ü JSF2 Verwendung von `StockSystem` EJB (9 Punkte)

Aufgabe

Nutzen Sie in `ShoppingSessionViewController` die `StockSystem` EJB.

Anforderungen

1. Modifiziert werden sollen die folgenden Methoden entsprechend den Hinweisen, die Sie im Quellcode finden:
   - `getAvailableTouchpoints()`: es sollen nur Verkaufsstellen angezeigt werden, an denen der gesamte Warenkorb erworben werden kann.
   - `onTouchpointSelectionChanged()`: für die neu ausgewählte Verkaufsstelle sollen die Preise der einzelnen Produkte im Warenkorb neu ermittelt werden.
   - `validateUnitsUpdate()`: überprüfen Sie, ob der Warenkorb mit der geänderten Produktanzahl in der ausgewählten Verkaufsstelle – falls bereits eine Auswahl vorgenommen wurde – verfügbar ist. Setzen Sie die ausgewählte Verkaufsstelle zurück, falls der Warenkorb dort nicht verfügbar ist. Geben Sie eine Fehlermeldung aus, falls keine Verkaufsstelle die Produkte verfügbar hat.
2. Erweitern Sie `StockSystem`, falls erforderlich, um neue Methoden.

Bearbeitungshinweise

- ***Anforderung 2***: um den Preis eines Produkts an einer Verkaufsstelle zu ermitteln ist z.B. die Erweiterung von `StockSystem` um die folgende Methode sinnvoll:
  - `public int getPriceForProductAndPos(IndividualisedProductItem product, int pointOfSaleId);`

  Diese können Sie wie folgt implementieren:
  - lesen Sie das `StockItem` für `product` und `pointOfSale` aus.
  - wenn auf diesem `StockItem` ein Preis > 0 gesetzt ist, geben Sie diesen zurück.
  - andernfalls geben Sie den Preis zurück, der auf `product` gesetzt ist (das ist gewissermaßen der filialunabhängige Default-Preis)
- Hintergrund dafür ist, dass auf `StockItem` der 'Default-Preis' von `AbstractProduct` 'überschrieben' werden kann, z.B. mit dem GUI aus JSF5 für einen bestimmten `PointOfSale` (ich kenne das aus eigener Erfahrung z.B. als Kunde von Bäckereiketten, die teilweise standortabhängige Preise haben).
- ***Anforderung 2***: die ggf. ausgewählte Verkaufsstelle ist der Wert des Attributs `touchpoint` auf `ShoppingSessionViewController`.
- ***Anforderung 2***: für die Ausgabe der Fehlermeldung bei Nichtverfügbarkeit des Warenkorbs können Sie den bisher auf einer Dummy-Überprüfung basierenden Code in `validateUnitsUpdate()` verwenden.

Ü JSF3 Verwendung von `CustomerCRUD` EJB (8 Punkte)

Aufgabe

Nutzen Sie in `ShoppingSessionViewController` die `CustomerCRUDStateless` EJB.

Anforderungen

1. Erweitern Sie die `CustomerCRUDStateless` EJB um eine Methode `readCustomerForEmail()`, die Ihnen für eine gegebene Mailadresse ein `Customer` Objekt oder `null` zurückliefert.
2. Erweitern Sie ein für Ihre Anwendung geeignetes Interface der EJB durch die erstellte Methode.
3. Machen Sie das Interface der EJB in `ShoppingSessionViewController` verfügbar und nutzen Sie es, um in `registerCustomer()` die folgende Logik zu implementieren:
   - wenn der Nutzer ein bestehender Kunde ist, dann versuchen Sie, diesen durch Aufruf von `readCustomerForEmail()` zu ermitteln.
   - wenn der Nutzer ein neuer Kunde ist, dann erstellen Sie unter Verwendung der EJB einen neuen Kunden.
   - aktualisieren Sie das `customer` Attribut von `ShoppingSessionViewController` mit dem Rückgabewert der aufgerufenen Methoden, falls Sie hier nicht `null` bekommen.

Bearbeitungshinweise

- ***Anforderung 1***: Sie können dafür die JPA Query Language verwenden und damit – wie in `CustomerTransactionCRUDStateless` gezeigt Queries ausführen. Beachten Sie, dass im Query ein String, der '@' enthält mit einfachen Quotes umschlossen werden muss.
- ***Anforderung 3***: um zu überprüfen, ob `registerCustomer()` als *'Anmelden'* (eines bestehenden Kunden) oder als *'Registrieren'* (eines Neukunden) aufgerufen wird, können Sie das Attribut `newCustomer` auf `ShoppingSessionViewController` auswerten. Dieses ist für Neukunden auf `true` gesetzt.

Ü JSF4 Einbinden von `ShoppingSession` EJB (7 Punkte)

Aufgabe

Nutzen Sie in `ShoppingSessionViewController` die `ShoppingSession` EJB.7

Anforderungen

1. Nutzen Sie `ShoppingSession` in einer entsprechend der Funktionalität der EJB geänderten Implementierung von `purchaseProducts()`.

Bearbeitungshinweise

- Da `ShoppingCart` als eigene ManagedBean verwendet wird, besteht die einzige Lösung ohne JSF Komplikationen wohl darin, dass Sie `ShoppingSession` in `purchaseProduct()` mit den Inhalten von `shoppingCartModel` neu aufbauen und dann die `purchase` Operation durchführen.

Ü JSF5 GUI zur Verwaltung von StockItem (12 Punkte)

Aufgabe

Implementieren Sie mit JSF oder einer anderen Technologie Ihrer Wahl ein GUI, das Ihnen die Verwaltung von existierenden `StockItem` Elementen ermöglicht. Diese und die folgende Übung verschaffen Ihnen einen anschaulichen Zugriff auf die in den Aufgaben zu EJB, JPA und PAT von Ihnen umgesetzten Geschäftslogikfunktionen und runden die Vorstellung von Java EE als Full-Stack Framework für server-seitige Anwendungsentwicklung ab.

Anforderungen

1. Verwenden Sie das Codegerüst `.ue.jsf5`.
2. Dargestellt werden sollen alle existierenden `StockItem` Elemente in Form einer Tabelle.
3. Deklarieren Sie für das Auslesen aller `StockItem` Elemente die `getCompleteStock()` Methode von `StockSystemLocal`, d.h. greifen Sie nicht direkt aus der Präsentationsschicht auf `StockItemCRUD` zu.
4. Zur Darstellung der Verkaufsstelle soll der Name des `AbstractTouchpoint` verwendet werden, der dem `PointOfSale` des `StockItem` entspricht.
5. Für ein einzelnes `StockItem` Element sollen Quantität und Preis jeweils einzeln modifiziert werden können.
6. Falls die auf Ihren EJBs verfügbaren verfügbaren Methoden nicht ausreichen, dann fügen Sie ggf. weitere Methoden hinzu. Ziehen Sie aber auch in Erwägung, präsentationsbezogene Funktionalität in der Managed Bean Ihrer JSF Anwendung zu implementieren.

Bearbeitungshinweise

- Bei Verwendung von JSF können Sie folgendes beachten:
  - ***Anforderung 2***: Hinweise finden Sie in `products.xhtml` bzw. `shoppingCart.xhtml` anhand der Verwendung von `h:DataTable`.
  - ***Anforderung 4***: Dafür können Sie aus dem Facelet mittels einer `#{...}` Expression eine geeignete Methode von `StockSystemViewController` aufrufen, die die `id` des `PointOfSale` entgegennimmt und den entsprechenden Touchpoint-Namen zurückgibt.
  - ***Anforderung 5*** Sie können dafür innerhalb von `h:DataTable` `h:form` Elemente verwenden, die einen `h:inputText` für die Anzeige/Eingabe eines neuen Werts sowie einen `h:commandButton` für Ausführung der Wertänderung beinhalten, wie in `shoppingCart.xhtml`
  - ***Anforderung 5*** Um das zu modifizierende `StockItem` zu identifizieren, können Sie geeignete `ids` als `f:param` wie in `shoppingCart.xhtml` an die in der Managed Bean implementierten Update-Methoden übergeben.
  - Falls Sie Änderungen an den EJBs Ihrer Anwendung vornehmen, dann beachten Sie bitte, dass die Run-Konfiguration für `UE-JSF5` nur die JSF Anwendung baut. Um die EJBs zu aktualisieren, müssen Sie daher bei Änderungen jeweils das Projekt `build-ejb` durch Ausführung von *install* in der *Maven Projects* Ansicht bauen, bevor Sie `UE-JSF5` von neuem starten.

Ü JSF6 Aktion zur Ausführung von `doShopping()` (2 Punkte)

Aufgabe

Fügen Sie dem GUI aus JSF5 zu Testzwecken eine Aktion hinzu, die einen Einkauf auf den Warenbeständen auslöst.

Anforderungen

1. Verwenden Sie ein geeignetes Bedienelement, z.B. einen Button, bei dessen Betätigung die Methode `doShopping()` von `StockSystemViewController` aufgerufen wird. Diese greift auf die gleichnamige Methode der im Codegerüst enthaltenen Klasse `StockSystemHelper` zu.
2. Vervollständigen Sie die Implementierung von `doShopping()` in `StockSystemHelper` durch Einkommentieren und ggf. Anpassung der auskommentierten Codebestandteile. Diese befüllen einen Warenkorb und lösen dann einen Kauf aus.
3. Nach erfolgreicher Ausführung der Aktion soll die Anzeige der Warenbestände aktualisiert werden.

Bearbeitungshinweise

- ***Anforderung 2*** Bei Nutzung von JSF können Sie dafür z.B. die in PAT1 und PAT2 entwickelte `ShoppingSessionFacade` EJB in `StockSystemHelper` einbinden. Wenn Sie in PAT1 die Funktionalität von `ShoppingSessionFacade` als REST Service umgesetzt haben, können Sie hier die EJB verwenden, die den Service implementiert.