

## Aufgabenstellung für das *dritte* Übungsbeispiel der LVA “Objektorientiertes Programmieren“

**Abgabeschluss für den Source Code:      Sonntag 14.06.2020**

**WICHTIG:**

Wird das Beispiel nicht fristgerecht abgegeben, haben Sie in Folge nicht mehr die Möglichkeit, die LVA im laufenden Semester erfolgreich abzuschließen!

**Abgabegespräch:**

Zwischen **16.06.2020 und 18.06.2020** (Den Tag und die genaue Uhrzeit wählen Sie selbst in TUWEL.)

**Allgemeines**

Jeder Teilnehmer muss die Beispiele **eigenständig** ausarbeiten. Bei der Abgabe jedes Beispiels gibt es ein Abgabegespräch mit einem Tutor oder Assistenten. Diese Abgabegespräche bestehen aus den folgenden Punkten:

- Im Zuge der Abgaben müssen Sie eine Codeänderung vornehmen können.
- Beantworten von Fragen zu Ihrem Programm.
- Ihr Programm wird mit automatisierten Tests geprüft.
- Beantwortung theoretischer Fragen zu den jeweils in der Übung behandelten Konzepten.

**Umsetzung und Abgabe**

- Halten Sie sich exakt an die Beschreibungen der Klassen in der **JavaDoc**.
- Geben Sie Ihrem Eclipse-Projekt einen eindeutigen Namen. Halten Sie sich dabei an folgende Benennungsvorschrift:

**Eclipse-Projekt:**    *matrNr\_Beispiel3\_Nachname*

**Beispiel:**            *00123456\_Beispiel3\_Mustermann*

**ACHTUNG:** Wenn Sie sich nicht an die Benennungsvorschrift halten, kann Ihre Abgabe nicht gewertet werden!

- Name und Matrikelnummer muss zu Beginn jeder Klasse in einem Kommentarblock angeführt werden!
- Exportieren Sie das Projekt (inklusive aller für Eclipse notwendigen Dateien, z.B.: .classpath, .project) als ZIP-Datei. Eine Anleitung dazu finden Sie in TUWEL.

**Kontakt:**

Inhaltliche Fragen:      [Diskussionsforum](#) in TUWEL

Organisatorische Fragen: [oop@ict.tuwien.ac.at](mailto:oop@ict.tuwien.ac.at)

## Aufgabe: Registrierkassensystem

### Ziel:

Erwerb konkreter Erfahrung und Kompetenz in der Umsetzung folgender Konzepte: Klassendiagramme, Sequenzdiagramme, Casting, Polymorphismus, Fehlerbehandlung, Testen, Patterns und Verwendung von (externen) Libraries.

### Einleitung:

Für das Erreichen der Standardanforderung im Praxis-Teil ist Voraussetzung, dass alle Anforderungen der JavaDoc und des Sequenzdiagramms korrekt umgesetzt wurden. Für die zusätzlichen Extrapunkte müssen Sie auch noch die weiter unten angeführte Bonusaufgabe lösen. Die Funktion Ihrer Programmteile müssen Sie im eigenen Ermessen testen.

Ihre Aufgabe besteht darin einen Teil eines Registrierkassensystems zu implementieren. Eine Übersicht der von Ihnen zu implementierenden Packages und Klassen finden Sie in Abbildung 4. Teile der **rot** markierten Packages müssen von Ihnen implementiert werden. Konkret sind folgende Klassen von Ihnen umzusetzen:

- ManagementServer
- CashRegister
- CashRegisterFactory
- CashRegisterConsoleUI
- NotRegisteredException
- ShoppingCart
- ShoppingCartFactory
- ShoppingCartNotFoundException.

Die im Klassendiagramm grau markierten Packages sowie die Interfaces der **rot** markierten Packages werden als Bibliothek (Library) in einer JAR-Datei bereitgestellt. Entsprechend können Sie sich die Methodensignaturen der zu implementierenden Methoden automatisch generieren lassen.

In der ZIP-Datei befinden sich auch die Klassendiagramme (inklusive Methoden) für jedes Package. Die Klassendiagramme wurden mit ObjectAid erzeugt. Diese sind als Basis für die Umsetzung heranzuziehen.

Eine genaue Beschreibung der einzelnen Methoden der Klassen finden Sie in der beigelegten **JavaDoc**. Halten Sie sich **exakt** an die **Spezifikationen** in den **Klassendiagrammen**, **Sequenzdiagrammen** und der **JavaDoc**.

Eine grobe Übersicht der im Gesamtsystem enthaltenen Packages finden Sie im Folgenden:

### Packages

- **cashregister:** *CashRegisters* können sich beim *ManagementServer* als *IObserver* registrieren. Sind sie registriert, werden sie über Änderungen im Produktsortiment vom *ManagementServer* benachrichtigt. Dieses Produktsortiment wird in jedem *CashRegister* zwischengespeichert. Zusätzlich kann ein *CashRegister* mehrere *ShoppingCarts* verwalten. Zu diesen *ShoppingCarts* (Warenkörben) können *IShoppingCartElements* hinzugefügt werden. Der *CashRegister* ermöglicht es Zahlungen für die von ihm verwalteten *ShoppingCarts* abzuwickeln. Dazu wird das Package *domain.record* verwendet.
- **cashregister.ui:** Dieses Package definiert eine Schnittstelle *ICashRegisterUI* für User Interfaces einer Registrierkasse. Ihre Aufgabe ist es eine Realisierung in Form eines User Interfaces, das alle Informationen auf die Konsole ausgibt. Mit diesem User Interface können die verwalteten Informationen des *CashRegister* dargestellt werden.
- **container:** Das Package *container* bietet eine Realisierung für das *java.util.Collection* Interface. Verwenden Sie in Ihrer Implementierung für das Registrierkassensystem für jedes Objekt des Typs *Collection* ein *Container*-Objekt. Dieses Package wurde bereits in Aufgabe 2 umgesetzt. Aus diesem Grund scheint es nicht im Klassendiagramm in Abbildung 4 **Fehler! Verweisquelle konnte nicht gefunden werden.** auf.
- **rbvs.product:** Dieses Package erlaubt Produkte zu erstellen und bietet Schnittstellen um auf diese zuzugreifen.
- **rbvs.record:** In diesem Package werden *Invoices* (Rechnungen) und *PaymentTransactions* verwaltet.

- **managementserver:** Der Managementserver verwaltet die bei ihm eingetragenen Registrierkassen und das Produktsortiment. Das Produktsortiment wird als Baumstruktur verwaltet (siehe Abbildung 2). Die Baumstruktur enthält dabei auf der ersten Ebene unterhalb des Wurzelknotens immer die zu dem Enum *ProductCategory* passenden *ProductCategoryTreeNodes*. Um das Produktsortiment auf dem aktuellen Stand zu halten, ist der Managementserver als *IWarehouseListener* beim Warehouse registriert. Dadurch wird er über Änderungen am Produktsortiment informiert und aktualisiert sein Sortiment entsprechend. Alle Registrierkassen die als *IObserver* registriert sind, werden über diese Änderungen informiert. Der Managementserver ist als *Singleton* zu implementieren.
- **paymentprovider:** Mit diesem Package können Bezahlvorgänge über einen *PaymentProvider* abgewickelt werden. Bei einer erfolgreichen Abwicklung wird eine *domain.record.PaymentTransaction* erzeugt.
- **tree:** Dieses Package bietet die Möglichkeit Elemente in einer Baumstruktur (*ITree<T>*) zu organisieren. Das Package wurde aus Beispiel 2 übernommen und erweitert. Aus diesem Grund scheint es nicht im Klassendiagramm in Abbildung 4 auf.
- **util:** Definiert diverse Hilfsklassen (z.B. eine Formatierungsklasse für ein Datum, eine generische Klasse Tupel um zusammengehörige Daten zu speichern).
- **warehouse:** Das Warehouse verwaltet verfügbare Produkte. Über das Interface *IWarehouseListener* kann man sich bei einem Warehouse registrieren und wird über Änderungen in dem Produktsortiment informiert. Änderungen, für die eine Benachrichtigung erfolgt, sind das Hinzufügen, Löschen und Ändern von Produkten. Das Warehouse ist als *Singleton* implementiert.

## Allgemeiner Programmablauf

Wie oben beschrieben sind im Registrierkassensystem mehrere Teilsysteme beteiligt. Die Teilsysteme sind das Warenhaus, ein Managementserver und die eigentlichen Registrierkassen.

Im Warenhaus werden Produkte verwaltet und es ist der Ausgangspunkt für alle weiteren Operationen. Im gesamten System kann nur mit Produkten gearbeitet werden, die im Warenhaus verwaltet werden. Das Warenhaus erlaubt es Produkte oder auch deren Kategorie zu ändern, neue Produkte anzulegen und zu löschen. Bei jeder dieser Änderung werden über ein *Observer*-Pattern alle registrierten *Listener* benachrichtigt.

Im Falle unseres Systems ist der Managementserver ein *Listener*. Dieser ist dafür zuständig, dass das Produktsortiment des Warenhauses anhand bestimmter Kriterien in einer Baumstruktur organisiert wird. Jede Änderung im Sortiment des Warenhauses löst eine Benachrichtigung aus, die der Managementserver als *Listener* empfängt. Anhand der dabei weitergegebenen Informationen können diese Änderungen direkt in das Produktsortiment des Managementserver übernommen werden. Der Managementserver hat dadurch eine Kopie des Produktsortiments des Warenhauses in einer Baumstruktur.

Weiters ist es über den Managementserver möglich Registrierkassen zu verwalten. Über den Managementserver können dynamisch Registrierkassen angelegt werden.

Der Managementserver implementiert ein *Observer*-Pattern und benachrichtigt alle registrierten Observer über Änderungen an seinem Produktsortiment. Registrierkassen können als Observer in dem Managementserver registriert werden. Sind diese registriert, so werden sie im Falle von Änderungen am Produktsortiment benachrichtigt. Die Benachrichtigung über Änderungen am Produktsortiment wird manuell ausgelöst und nicht automatisch durch Änderungen, die vom Warenhaus gemeldet werden. Dies erfolgt mit der Methode `ManagementServer#propagateProducts()`.

Registrierkassen haben ein lokales Produktsortiment, das sie vom Managementserver beziehen. Dabei wird eine Kopie des Produktsortiments des Managementserver lokal in der Registrierkasse verwaltet.

Eine Registrierkasse kann mehrere Warenkörbe haben. Aus dem Produktsortiment können Produkte selektiert und in diesen abgelegt werden. Produkte, die in einem Warenkorb abgelegt sind, können bezahlt werden. Im Zuge dessen wird eine Rechnung erstellt und der Warenkorb geleert. Rechnungen werden in der Registrierkasse gesammelt.

Registrierkassen, die beim Managementserver als Observer registriert sind, werden über Änderungen benachrichtigt. Im Falle einer Benachrichtigung muss das Produktsortiment vom Subject (Managementserver) abgeholt werden.

## Graphische Benutzeroberfläche für das Registrierkassensystem

Um die Arbeit mit dem Registrierkassensystem zu erleichtern, ist eine graphische Benutzeroberfläche (GUI) verfügbar. Dieses GUI wird in der Library zur Verfügung gestellt.

Für die Packages *warehouse*, *managmentserver* und *cashregister* sind eigenständige GUIs (jeweils als ein eigenes Fenster) implementiert.

### Warenhaus GUI:

Die GUI des Warehouses erlaubt es ein Produktsortiment zu verwalten. Sie bietet eine tabellarische Ansicht der verfügbaren Produkte. Es können Produkte angelegt, gelöscht und aktualisiert werden. Weiters ist es möglich Produkten Kategorien zuzuordnen. Die Kategorie wird verwendet um diese automatisch in eine Baumstruktur einzusortieren (vgl. Managementserver). Einem *CompositeProduct* können bestehende Produkte hinzugefügt werden. Bitte beachten Sie, dass hier keine Überprüfung auf etwaige Zyklen in Ihren CompositeProducts durchgeführt wird.

**Warehouse Manager**

Name:   
 Price/Discount Percentage:   
 Category:   
 Add Product:

**Available Products**

Name	Price	Category	Type	Children
Taco	3	FOOD	SIMPLE	
Burrito	6.5	FOOD	SIMPLE	
Sushi	4	FOOD	SIMPLE	
Beer	3	BEVERAGE	SIMPLE	
Wine	6.5	BEVERAGE	SIMPLE	
Water	1	BEVERAGE	SIMPLE	
Taco Menu	3.92	FOOD	JOINT	[Product [name=Taco, price=3.0], Product [name=Wate...
<b>Taco Deluxe Menu</b>	<b>9.31</b>	<b>FOOD</b>	<b>JOINT</b>	<b>[Product [name=Taco, price=3.0], Product [name=Wine...</b>
Orphal Menu	23.02	FOOD	JOINT	[JointProduct [name=Meal plate, price=13.095, product...
Gedeck	1	DEFAULT	SIMPLE	

Currently registered at warehouse: warehouse.Warehouse@53bd815b

Abbildung 1 - Warehouse GUI

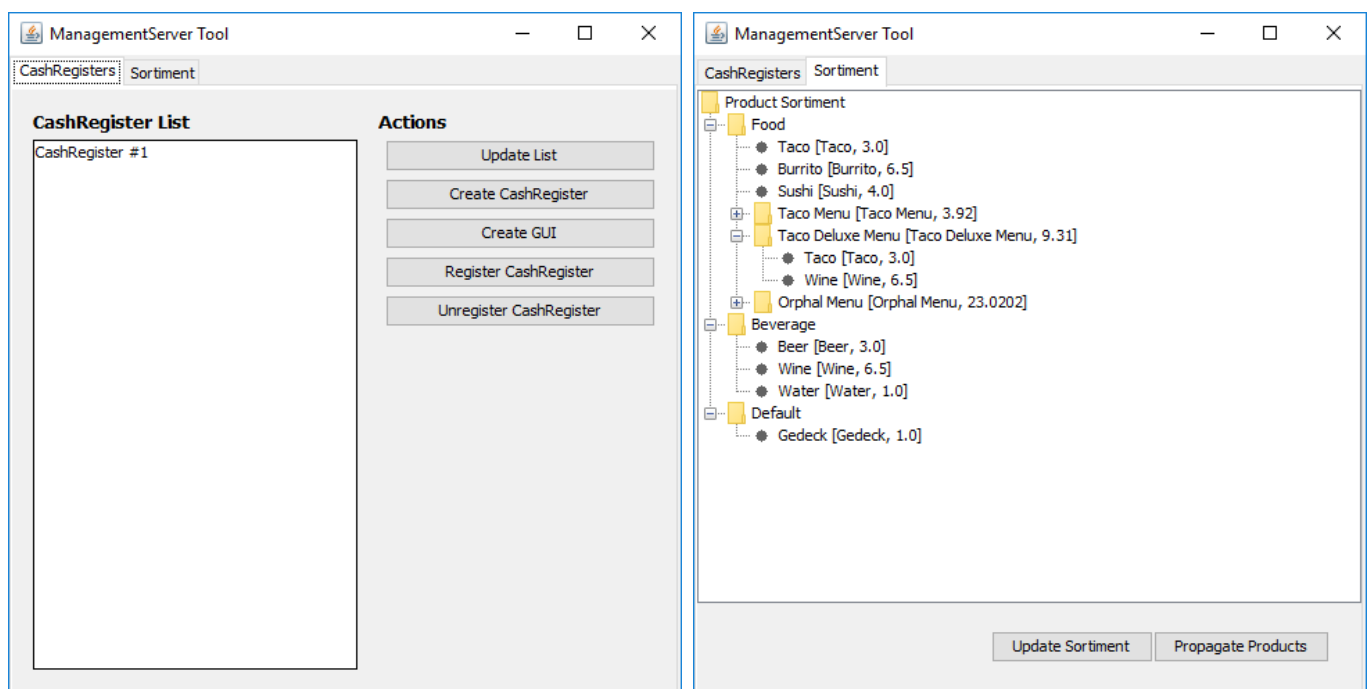
**ManagementServer GUI:**

Die GUI des ManagementServers erlaubt es neue Registrierkassen anzulegen und für eine gewählte Registrierkasse ein zusätzliches GUI zu erstellen. Diese GUI wird in der Library zur Verfügung gestellt.

Mittels der Menüpunkte “Register CashRegister“ und “Unregister CashRegister“ kann für eine gewählte Registrierkasse die Registrierung beim Managementserver durchgeführt oder aufgehoben werden (*Observer*-Pattern).

Am ManagementServer kann manuell eine Benachrichtigung an alle registrierten Observer ausgelöst werden. Dies geschieht über den Button “Propagate Products“. In diesem Fall werden alle registrierten Observer darüber benachrichtigt, dass sich das Produktsortiment geändert hat. Die Observer können dann vom Managementserver (Subject) das aktuelle Produktsortiment abholen.

Die GUI selbst bietet eine graphische Repräsentation des aktuellen Produktsortiments als Baumstruktur. Diese Darstellung wird allerdings nicht automatisch bei jeder Änderung aktualisiert. D.h., ändert sich das Produktsortiment im Managementserver, so sind diese Änderungen nicht sofort in der GUI ersichtlich. Über den Button “Update Sortiment“ kann die graphische Darstellung manuell aktualisiert werden.



*Abbildung 2 - Produktsortiment im Managementserver*

### Registrierkassen GUI:

Die GUI einer Registrierkasse hat eine graphische Darstellung des aktuellen lokalen Produktsortiments. Dieses Produktsortiment wird aktualisiert, sofern die Registrierkasse als Observer beim Managementserver eingetragen ist und dieser eine Benachrichtigung ausschickt. Diese GUI wird in der Library zur Verfügung gestellt.

Warenkörbe können erstellt und Produkte hinzugefügt werden. Produkte in Warenkörben können bezahlt werden. Wird ein Warenkorb bezahlt, wird eine Rechnung erstellt und alle Produkte aus dem Warenkorb werden entfernt.

Die auf der Registrierkasse vorhandenen Rechnungen können in der GUI angezeigt werden.

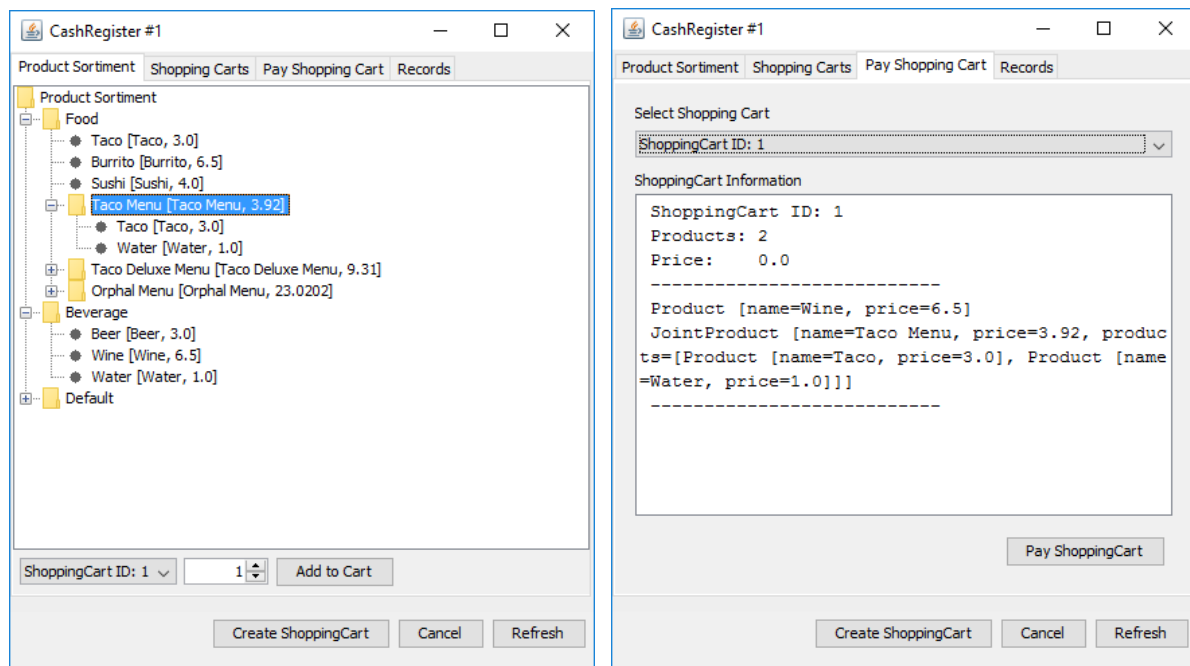


Abbildung 3 - CashRegister GUI

### Weiterführende Informationen

Folgende Patterns werden im Zuge der Übung behandelt. Weitere Detailinformationen finden Sie in den Vorlesungsfolien bzw. unter den folgenden Links:

- Listener Pattern [http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/event\\_listener.html](http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/event_listener.html)
- Observer Pattern <http://www.vogella.com/tutorials/DesignPatternObserver/article.html>
- Singleton Pattern <http://www.vogella.com/tutorials/DesignPatternSingleton/article.html>
- Factory Pattern [http://www.tutorialspoint.com/design\\_pattern/factory\\_pattern.htm](http://www.tutorialspoint.com/design_pattern/factory_pattern.htm)

## BONUSAUFGABE – Spring REST

Erweitern Sie Ihre Implementierung um eine Möglichkeit von außen mit dem System zu interagieren. Bieten Sie zu diesem Zweck eine RESTful-API an, über die mit dem System kommuniziert werden kann.

Eine RESTful-API bietet mehrere REST (Representational State Transfer) Endpoints an. REST ist ein Architekturstil für verteilte Systeme und wird, unter anderem, für Webservices verwendet. Jeder REST-Endpoint wird dabei über einen eindeutigen URI (Uniform Resource Identifier) identifiziert. Auf diesen Endpoints können per HTTP unterschiedliche Methoden aufgerufen werden. Beispielsweise können Daten abgefragt werden (GET) oder Daten übertragen (POST) werden. Dabei kann eine Payload (Daten im Body) mit übergeben werden. REST-Endpoints können in einem Browser per URL aufgerufen werden, z.B. <http://localhost:8080/cashregisters>. Als Resultat soll das Datenformat JSON verwendet werden.

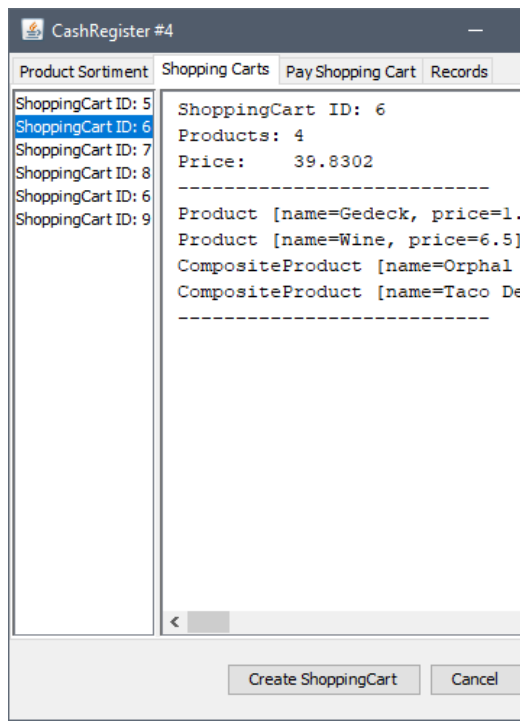
Ihre RESTful-API soll folgende Funktionalität anbieten:

- Abfrage des Produktsortiments
- Abfrage aller beim Server registrierten Registrierkassen
- Abfrage einer speziellen Registrierkasse. Hier soll für eine Registrierkasse zumindest alle darin enthaltenen ShoppingCarts zurückgeliefert werden.
- Abfrage aller Elemente in einem speziellen ShoppingCart in einer Registrierkasse.
- Bestellen von Produkten (für ein ShoppingCart)
- Bezahlen eines ShoppingCarts
- Abfrage von bezahlten ShoppingCarts

Für die Implementierung empfehlen wir die Verwendung des Spring Boot-Frameworks. Eine Basisimplementierung wurde bereits in dem Package *bonus* vorbereitet. Die Klasse *CashRegisterSystemRestController* im Package *bonus.spring.controller* beinhaltet bereits einige Beispiele für REST-Endpoints. Beispielsweise bietet die Methode *shoppingcarts* einen REST-Endpoint unter [/cashregister/{cid}/shoppingcart/{sid}](#) an. Dabei sind {cid} und {sid} variabel und werden durch konkrete IDs ersetzt. Alle Endpoints sind dabei unter <http://localhost:8080> (Port 8080) abrufbar. Der obige Endpoint kann beispielsweise unter:

<http://localhost:8080/cashregister/4/shoppingcart/6>

erreicht werden. Dabei werden die Elemente von ShoppingCart 6 in CashRegister 4 abgerufen. Das Ergebnis des Aufrufs soll im JSON-Format übertragen werden. Spring übernimmt dabei für Sie die Konvertierung Ihrer Datentypen in das passende Format. Eine mögliche Ausgabe sehen Sie hier:



```

{
  "shoppingCartElements": [
    {
      "name": "Gedeck",
      "price": 1,
      "category": "DEFAULT"
    },
    {
      "name": "Orphal Menu",
      "price": 23.0202,
      "category": "FOOD",
      "products": [
        {
          "name": "Meal plate",
          "price": 13.095,
          "category": "FOOD",
          "products": [
            {
              "name": "Taco",
              "price": 3,
              "category": "FOOD"
            }
          ]
        }
      ]
    }
  ]
}

```

Weiterführende Informationen zu REST und Spring finden Sie hier:

REST: [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)

SPRING: <https://spring.io/guides/tutorials/rest/>



