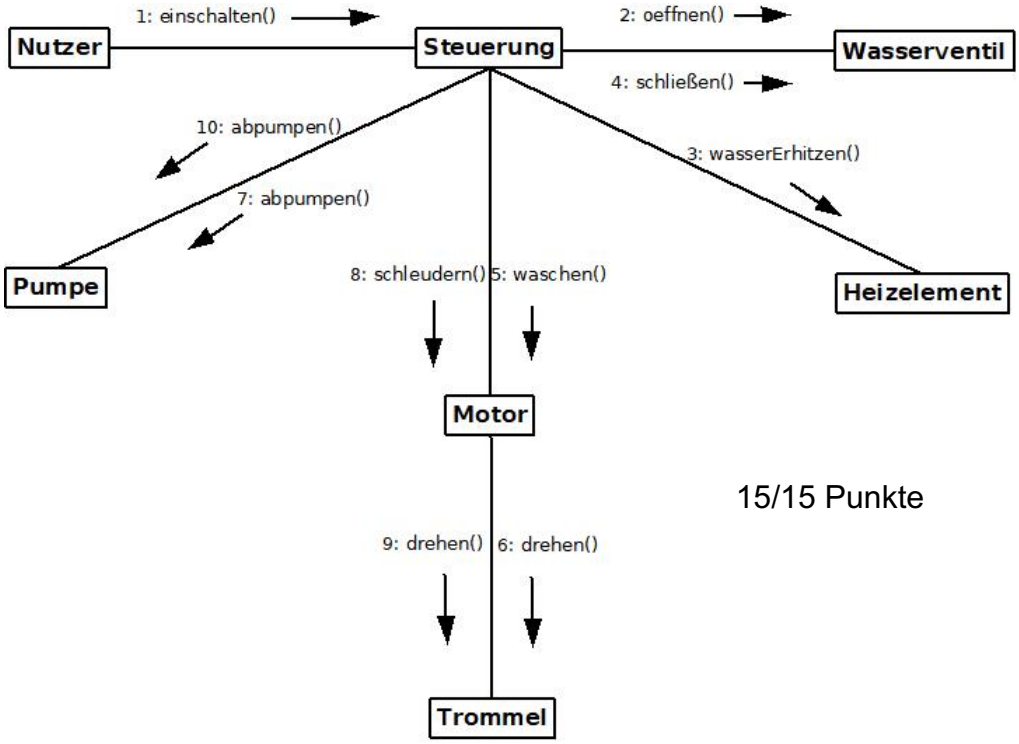
Übungsblatt:	5	**1. Abgabepartner:**	Matthias Wolff	(458 766)
		2. Abgabepartner:	Anton Mende	(461 328)
		2. Abgabepartner:	Anika Herbermann	(461 655)

Aufgabe 15



15/15 Punkte

Aufgabe 16

a)

- **parallele Verarbeitung**: Die Fön-Software muss sowohl Lüfterdrehzahl als auch Heizelementtemperatur gleichzeitig steuern

- **Einbenutzerfähigkeit**: Ein Fön wird immer nur von einer einzigen Person gleichzeitig benutzt

- **Schichten**: Der Fön muss sich nicht zu einem Server verbinden. Somit muss nur die Client-Schicht entwickelt werden, die die komplette Steuerung des Föns übernimmt

- **Notwendigkeit verschiedener Plattformen**: Das System soll nur als Embedded-System fest im Fön verbaut laufen. Somit muss nur für diese eine Plattform entwickelt werden

- **Wiederverwendung vorhandener Bausteine**: Die Fön-Software benötigt keine Wiederverwendung vorhandener Bausteine, da lediglich Temperatur und Lüfterdrehzahl geregelt werden müssen. Dafür kann man keine Bausteine wiederverwenden.

- **Hilfesystem**: Die Fön-Bedienung ist intuitiv genug, sodass kein Hilfesystem benötigt wird.

- **DBMS**: Der Fön muss keine Daten persistent speichern. Somit entfällt die Benutzung eines DBMS

- **Benutzerschnittstelle**: Die Benutzerschnittstelle wird durch Hardware-Schalter realisiert, durch die der Benutzer Eingaben zur gewünschten Lufttemperatur und Lüfterdrehzahl tätigen kann.

- **Rückgriffe auf sonstige Dienstleistungen**: entfallen, da die Fön-Software nicht sehr komplex ist, sodass Rückgriffe auf sonstige Dienstleistungen nicht nötig sind.

b)

- **sequentielle Verarbeitung**: Der Kunde soll seine Pizza Schritt für Schritt nacheinander konfigurieren und die Konfiguration soll danach an den Unternehmens-Server geschickt werden. Es läuft alles einzeln nacheinander ab, somit wird keine Parallelität / Nebenläufigkeit benötigt.

- **Einbenutzerfähigkeit**: Die Software auf dem Smartphone wird jeweils nur von einem einzigen Kunden benutzt. Das Softwaresystem auf dem Unternehmens-Server muss mehrbenutzerfähig sein, die Handy-App jedoch nicht.

- **Schichten**: Die Software läuft lokal auf einem Smartphone als Client. Erst beim Abschluss der Konfiguration sendet der Client die fertig konfigurierte Pizza an einen Unternehmens-Server. Die zu entwickelnde Software ist also Client-Software.
- **Notwendigkeit verschiedener Plattformen**: Die Software muss auf allen gängigen Smartphones laufen (Android, Apple, WindowsPhone,...)
- **Wiederverwendung vorhandener Bausteine**: Die Auswahl-Menüs für Zutaten, Pizzaboden, Pizzasoße, etc. können mit anderen Werten wiederverwendet werden, da der strukturelle Aufbau der graphischen Auswahl-Menüs jeweils gleich ist.
- **Hilfesystem**: Ein Hilfesystem ist nicht zwingend erforderlich, da die App den Kunden intuitiv durch die Konfiguration seiner Pizza führt und Schritt für Schritt nach Angaben fragt.

- **DBMS**: Die Software, die auf dem Smartphone läuft sollte zur besseren Bedienbarkeit die persönlichen Daten des Kunden wie Name, Adresse, Telefonnummer, etc. lokal speichern, sodass der Kunde diese nicht jedes Mal von Hand eingeben muss. Da dies nur ein einziger Eintrag ist, ist hierfür ein DBMS nicht nötig. Zusätzlich müssen die möglichen Wahloptionen für Zutaten, Pizzaböden und Soßen in einer relationalen Datenbank gespeichert werden.

- **Benutzerschnittstelle**: Die Benutzerschnittstelle soll in Form einer GUI realisiert werden
- **Rückgriff auf sonstige Dienstleistungen**: GUI-Framework zur Erstellung der Benutzerschnittstelle, Netzwerk-Bibliothek zum Senden der Pizza-Konfiguration über 5G an den Unternehmens-Server. Für das Hilfesystem könnte ebenfalls ein Halbfabrikat verwendet werden.

c)

- **sequentielle Verarbeitung**: Jeder Registrierungs bzw. Bezahlvorgang wird nacheinander abgearbeitet.
- **Mehrbenutzerfähigkeit**: Mehrere Kunden werden das System gleichzeitig nutzen
- **Schichten**: Die Registrierung erfolgt in einer Web-Applikation. Als Client dient hierbei ein Smartphone bzw. Rechner des jeweiligen Kunden, der sich registrieren möchte. Die Bezahlung erfolgt am Automaten als Client, der den Bezahlvorgang mit dem Bezahlsystem als Server abwickelt.
- **Notwendigkeit verschiedener Plattformen**: Die Software zur Registrierung läuft in einem Browser (PHP, HTML). Die Software zum Bezahlen läuft lokal auf dem Pizzaautomaten.
- **Wiederverwendung vorhandener Bausteine**: entfällt, da keine Teile des Systems sich dafür ähnlich genug sind.
- **Hilfesystem**: Ein Hilfesystem für den Kunden ist wünschenswert, um Fragen bei der Registrierung, beim Bezahlvorgang oder beim Verlust der Karte zu beantworten.

- **DBMS**: Die personenbezogenen Daten sollen vertraulich in einem relationalen DBMS gespeichert werden. Zusätzlich sollten getätigte Bestellungen gespeichert werden.
- **Benutzerschnittstelle**: Die Benutzerschnittstelle wird als GUI realisiert, die den Kunden im Browser durch den Registrierungsvorgang leitet. Am Automaten wird auf dem vorhandenen Bildschirm der Status der kontaktlosen Bezahlung angezeigt.
- **Rückgriff auf sonstige Dienstleistungen**: Es kann auf ein Halbfabrikat zur datenschutzkonformen Speicherung der personenbezogenen Daten zurückgegriffen werden, das die Daten (eventuell verschlüsselt) in einer Datenbank ablegt. Zusätzlich kann eine Bibliothek zum kontaktlosen Auslesen der Karten verwendet werden. Für das Hilfesystem kann ebenfalls ein Halbfabrikat verwendet werden.

15/15 Punkte

Aufgabe 17

a)

```java
import javax.persistence.*;
import java.util.ArrayList;
import java.util.Date;

/**
 * Aufgabe 17a
 *
 * @author Antonius Mende – 461 328
 * @author Anika Herbermann – 461 655
 * @author Matthias Wolff – 458 766
 *
 * @author "Gruppe_A"
 */



/**
Kunden-Klasse
*/

@Entity
public class Kunde implements java.io.Serializable{
    //Primaerschluessel
    protected String kundenId;
    protected String name;
    //Durch Diagramm vorgegebener Name buchung fuer Kollektion von Bestellungen
    protected Collection<Bestellung> buchung = new ArrayList<Bestellung>();

    public Kunde(String name){
        this.name = name;
    }

    //eindeutige KundenIDs sollen automatisch generiert werden
    @Id @ GeneratedValue(stragtegy=GenerationType.AUTO)
    public String getKundenId(){return this.kundenId;}
    public void setKundenId(String kundenId){this.kundenId=kundenId;}

    public String getName(){return this.name;}
    public void setName(String name){this.name=name;}

    //Ein Kunde ist mit mehreren Bestellungen verbunden
    @OneToMany
    public Collection<Bestellung> getBuchung(){return this.buchung;}
    public void setBuchung(Collection<Bestellung> b){this.buchung = b;}
    public void addBuchung(Bestellung b){this.buchung.add(b);}
    public void removeBuchung(Bestellung b){this.buchung.remove(b);}

}
/**
Bestellungs-Klasse
*/

@Entity
public class Bestellung implements java.io.Serializable{
    //Primaerschluessel
    protected int bestellungsId;
    protected Date zeitpunkt;
    protected Kunde kunde;
    protected Pizza pizza;

    public Bestellung(Kunde k, Pizza p){
        this.kunde = k;
        this.pizza = p;
    }

    //eindeutige BestellungsIDs sollen automatisch generiert werden
    @Id @ GeneratedValue(stragtegy=GenerationType.AUTO)
    public int getBestellungsId(){return this.bestellungsId;}
    public void setBestellungsId(int bestellungsId){this.bestellungsId=bestellungsId;}

    public Date getZeitpunkt(){return this.zeitpunkt;}
    public void setZeitpunkt(Date zeitpunkt){this.zeitpunkt = zeitpunkt;}

    //Mehrere Bestellungen sind mit einer Pizza verbunden
    @ManyToOne
    public Pizza getPizza(){return this.pizza;}
    public void setPizza(Pizza pizza){this.pizza = pizza;}
    //Mehrere Bestellungen sind mit einem Kunden verbunden
    @ManyToOne
    public Kunde getKunde(){return this.kunde;}
    public void setKunde(Kunde kunde){this.kunde = kunde;}

```

```
83  }
84
85  /**
86  Pizza-Klasse
87  */
88
89  @Entity
90  public class Pizza implements java.io.Serializable{
91      //Primaerschluessel
92      protected int pizzaId;
93      protected double preis;
94      protected String name;
95      //Name buchung fuer Kollektion von Bestellungen aus Assoziation Kunde-Bestellung uebernommen
96      protected Collection<Bestellung> buchung = new ArrayList<Bestellung>();
97
98      public Pizza(String name, double preis){
99          this.name = name;
100         this.preis = preis;
101     }
102
103     //eindeutige PizzaIDs sollen automatisch generiert werden
104     @Id @ GeneratedValue(stragtegy=GenerationType.AUTO)
105     public int getPizzaId(){return this.pizzaId;}
106     public void setPizzaId(int pizzaId){this.pizzaId=pizzaId;}
107
108     public String getName(){return this.name;}
109     public void setName(String name){this.name=name;}
110
111     public double getPreis(){return this.preis;}
112     public void setPreis(double preis){this.preis = preis;}
113
114     //Eine Pizza ist mit mehreren Bestellungen verbunden
115     @OneToMany
116     public Collection<Bestellung> getBuchung(){return this.buchung;}
117     public void setBuchung(Collection<Bestellung> b){this.buchung = b;}
118     public void addBuchung(Bestellung b){this.buchung.add(b);}
119     public void removeBuchung(Bestellung b){this.buchung.remove(b);}
120
121 }
```

b)

```java
import java.util.Date;

/**
 * Aufgabe 17b
 *
 * @author Antonius Mende – 461 328
 * @author Anika Herbermann – 461 655
 * @author Matthias Wolff – 458 766
 *
 * @author "Gruppe_A"
 */

//Man moechte von aussen auf das Objekt Bestellservice zugreifen, um z.B. neue Bestellungen anzulegen
@Remote
public interface BestellFassade{
    public static void erstelleBestellung(Kunde k, Pizza p) throws Exception;
}


//Wir brauchen keine Session-uebergreifende Speicherung von Zustaenden
@Stateless
//Jeder darf eine Bestellung aufgeben, egal welche Rolle sie im System haben
@PermitAll
public class Bestellservice implements BestellFassade{
    //injiziere EntityManager
    @PersistenceContext
    private EntityManager em;


    //Transaktionen sollen vom System verwaltet werden
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public static void erstelleBestellung(Kunde k, Pizza p) throws Exception{
        //Erstelle jetzigen Zeitpunkt als Date
        Date zeitpunkt = new Date();
        /*
        Es muss nicht geprueft werden, ob eine solche Bestellung schon existiert, da man mehrere Bestellungen für einen Kunden mit gleicher Pizza erlauben muss.
        Der Zeitpunkt reicht nicht aus, um "doppelte" Bestellungen zu finden, da der zeitpunkt auf Millisekunden genau ist
        und somit alleine durch kurze Laufzeiten des Programms nichtmehr übereinstimmt.

        Man koennte ein Feature einbauen, sodass ein Kunde innerhalb von X Sekunden nur einmal die gleiche Pizza bestellen kann,
        um Eingabefehler durch den Kunden (z.B. doppeltes Klicken auf Bestellen) abzufangen
        */
        Bestellung bestellung = new Bestellung(k,p);
        bestellung.setZeitpunkt(zeitpunkt);
        em.persist(bestellung);


}
```

sehr ausführlich, 20/20 Punkte