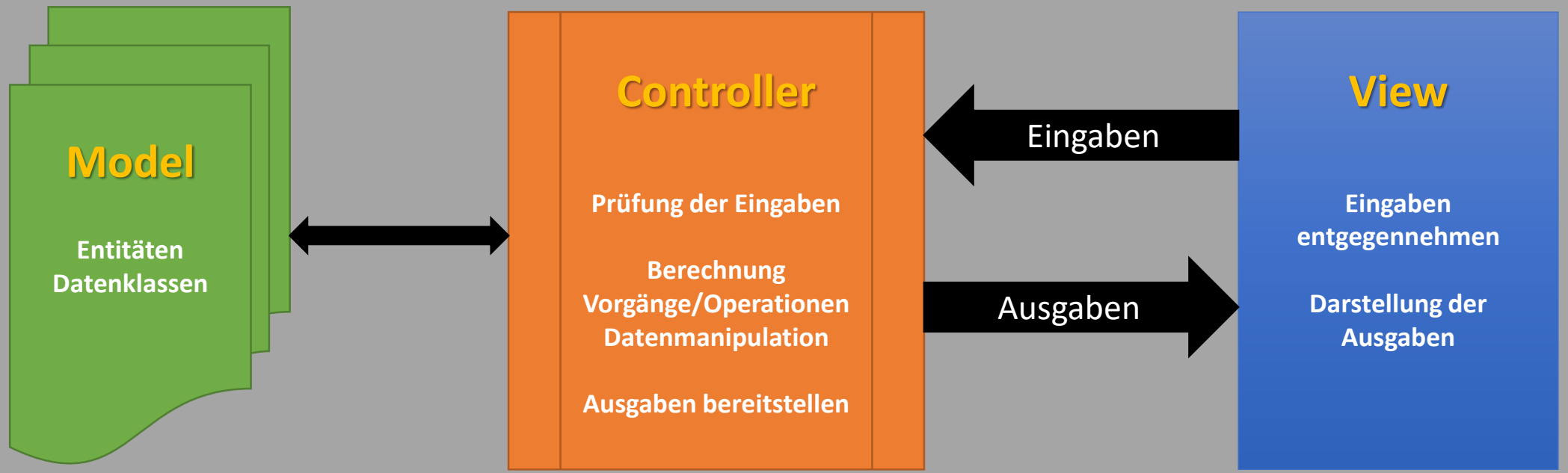
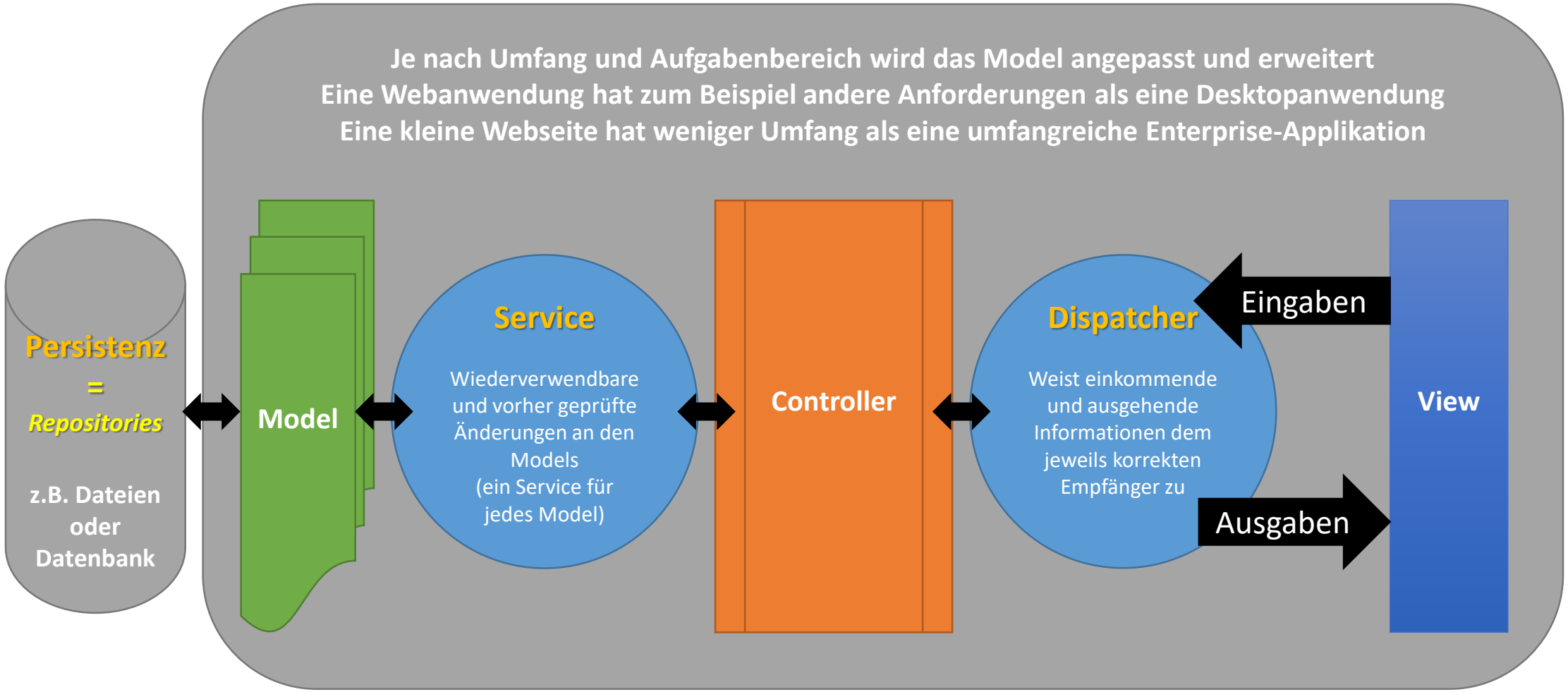
JAVA 06

Model View Control

Aufteilung der Aufgaben eines JAVA-Programms in verschiedene Klassen & Packages/Ordner
Die einzelnen Teile sind in sich abgeschlossen, austauschbar und wiederverwendbar!



Dieses Prinzip gilt generell in der Softwareentwicklung - auch zur besseren Übersicht


Je nach Umfang und Aufgabenbereich wird das Model angepasst und erweitert
Eine Webanwendung hat zum Beispiel andere Anforderungen als eine Desktopanwendung
Eine kleine Webseite hat weniger Umfang als eine umfangreiche Enterprise-Applikation
Persistenz
=
Repositories
z.B. Dateien oder Datenbank
Model
Service
Wiederverwendbare und vorher geprüfte Änderungen an den Models (ein Service für jedes Model)
Controller
Dispatcher
Weist einkommende und ausgehende Informationen dem jeweils korrekten Empfänger zu
Eingaben
Ausgaben
View

JAVA 06 – Model View Control

1. **Was ist ein „Model“?**
2. **Was ist ein „Service“?**
3. **Was ist ein „View“?**
4. **Was ist ein „Controller“?**
5. **Was ist ein „Dispatcher“?**

1. Was ist ein „Model“?

- Der **Model**-Part des MVC dient für den Zugriff auf Daten, die bei Bedarf auch **persistent** hinterlegt sind/werden können.
- **Datenklassen (Models,Entitäten)** bestehen „nur“ aus den **Daten** (**private Member**) und die zugehörigen **Zugriffsmethoden (Accessoren)**.
- Dieses Konstrukt wird auch **POJO** genannt (*Plain Old Java Object*)

```
public class Person {
        private String name;
        private float cash;

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

        public void setCash(float cash)         {       this.cash = cash;}
        public float getCash()                  {       return this.cash;}
}
```

2. Was ist ein „Service“?

- **Services sind (unter anderem) die Schnittstellen zu den Models als Teil des Controllers**
- Der **Service**-Part ist z.B. sinnvoll, um nicht immer dasselbe in verschiedenen Controllern zu machen.
- Prüfung von Eingaben der Benutzer, bevor Daten wirklich erstellt/verändert
- Daten können nach dem jeweiligen Bedarf bereit gestellt werden (Beispiel: Liste alle Personen die Autos besitzen…)

```
public class PersonService {
        private ArrayList<Person> people = new ArrayList<>();

        public void createPerson(String name) {
                if (name.length < 3 || name.length > 64) return false;       // Prüfung
                Person p = new Person();
                p.setName(name);              // Datenmanipulation
                this.people.add(p);           // gebündelte Bereitstellung aller Personen als Array
                return true;
        }
}
```

3. Was ist ein „View“?

- Der **View**-Part des MVC dient zur (formatierten) **Darstellung** und für das Entgegennehmen von **Benutzereingaben**.
- Für unsere Beispiele genügt eine simple Form, die Eingaben (**Scanner** -> String, int, …) und Ausgaben (über **out** und **err**) bereitstellt
- Views werden in Teams oft von einem Designer erstellt, der „den Rest“ nicht im Detail kennt, sondern nur nutzt (Stichwort: API)

```
public class SimpleView {
        public String receiveString(String label) { // label-Bsp: „Bitte geben Sie einen Namen ein:“
                System.out.println(label);
                String name = scan().nextLine();
                return name;
        }

        private Scanner scan() {    return new Scanner(System.in);    }        // Hilfsmethode
}
```

-> Für Details zum „SimpleView“ bitte den beigefügten Quellcode mit betrachten!!!

4. Was ist ein „Controller“?

- Der **Control**-Part bringt Model und View zusammen und führt letztendlich einen Teil des Programmes wirklich aus:
  - Input: Entgegennahme von Eingaben (vom View)
  - Verarbeitung Individuellen Programmablauf durchführen (unter Verwendung der Services / Models)
  - Output Feedback/Ausgabe als Reaktion (zum View)

```
public class PersonController {
        private PersonService personService = new PersonService<>();
        private SimpleView simpleView = new SimpleView();

        public void run() {
                String name = this.view.receiveString(„Bitte geben Sie einen Namen ein“);
                if (!personService.createPerson(name)) { // Beachten Sie das ! => „NICHT“
                        view.showError(„Der Name muss zwischen 3 und 64 Buchstaben haben!“)
                }
        }
}
```

5. Was ist ein „Dispatcher“?

- Dispatcher sorgen je nach Anwendung (Desktop/Window, Web, Mobile, …) dafür, dass Anfragen vom View zum richtigen Controller gelangen. Für unsere einfachen Beispiele **benötigen wir diesen (eigentlich) nicht.**

- Wenn man so will, könnte man die **main-Methode** als so eine Art Dispatcher betrachten, weil Sie dafür sorgt, das wir einen Einstiegspunkt für das jeweilige Programm haben:

```
public class ExampleApplication {
        public static void main(String args[]) {
                PersonController controller = new personController();
                controller.run();
        }
}
```