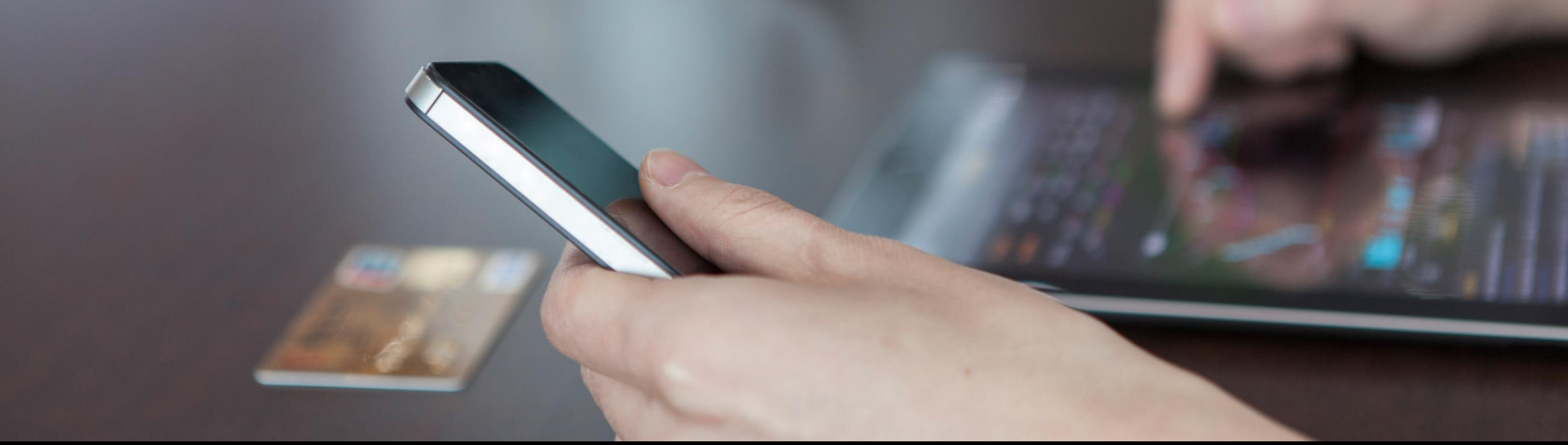


App-Entwicklung mit Android

Autoren: Philipp Stehle, Jacob Federer, Daniel Appenmaier
Version 4.2

PUBLIC

Agenda

- Einführung
- LoveCalculatorApp
- LoginApp
- ToDoApp
- ToDoAppExtended

Einführung

Android

- Android stellt eine freie Software-Plattform zur Entwicklung von mobilen Applikationen (Apps) dar
- Der Marktanteil von Android lag 2019 bei ca. 75%
- Geschichte:
  - 2003 gründet Andy Rubin die Firma Android mit dem Ziel, ein Betriebssystem für Digitalkameras zu entwickeln
  - 2005 wird Android von Google aufgekauft
  - 2007 gründet Google mit anderen Unternehmen die *Open Handset Alliance* mit dem Ziel, ein offenes Betriebssystem für alle zu entwickeln
  - 2008 erscheint das erste Smartphone (HTC Dream) mit Android als Betriebssystem

Wichtige Merkmale von Android

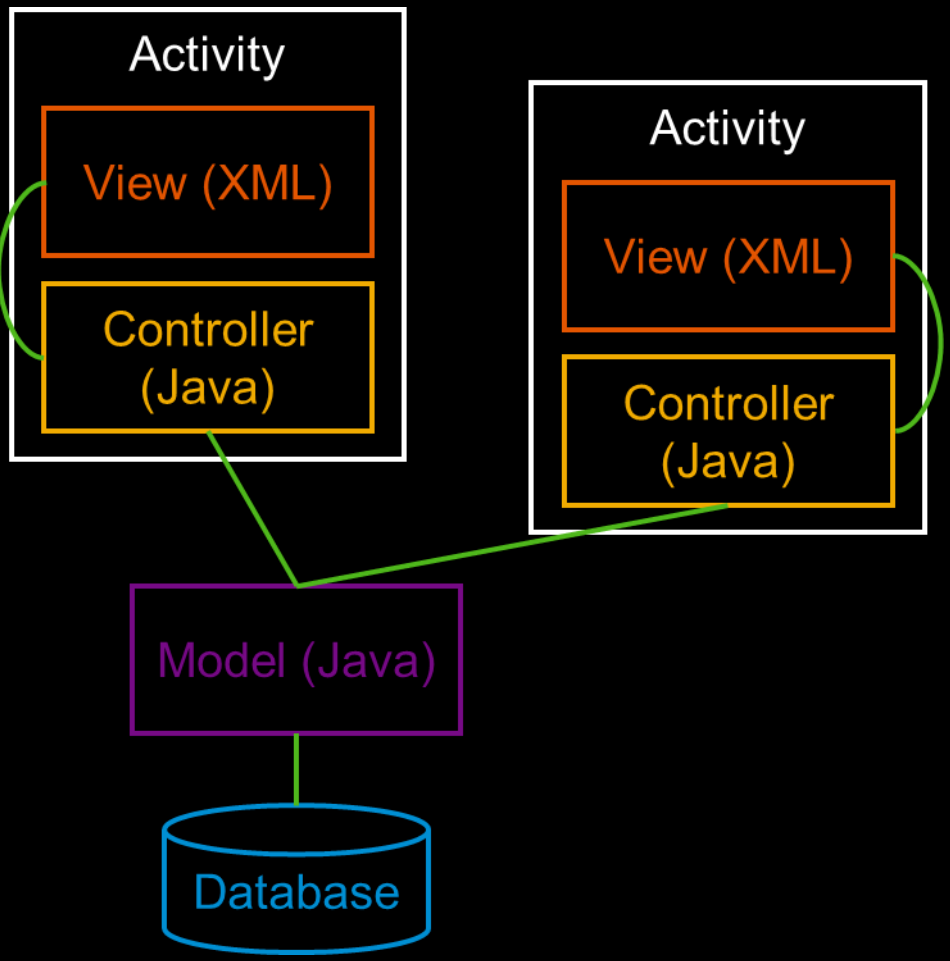
- Apps in Android können in Java und Kotlin entwickelt werden
- Zur App-Entwicklung stehen fast alle Java-Klassen zur Verfügung
- Android-Apps besitzen keine main()-Methode, sondern lose gekoppelte Komponenten
- Die wichtigste Komponente stellt die sogenannte „Activity“ dar (entspricht einem sichtbaren Fenster auf dem Bildschirm)

Android Studio

- Das Android Studio ist eine Entwicklungsumgebung für Java und Android
- Android Studio basiert auf der Entwicklungsumgebung IntelliJ
- Erste Anlaufstelle für angehende Android-Entwickler ist die offizielle Android-Entwickler-Seite: developer.android.com

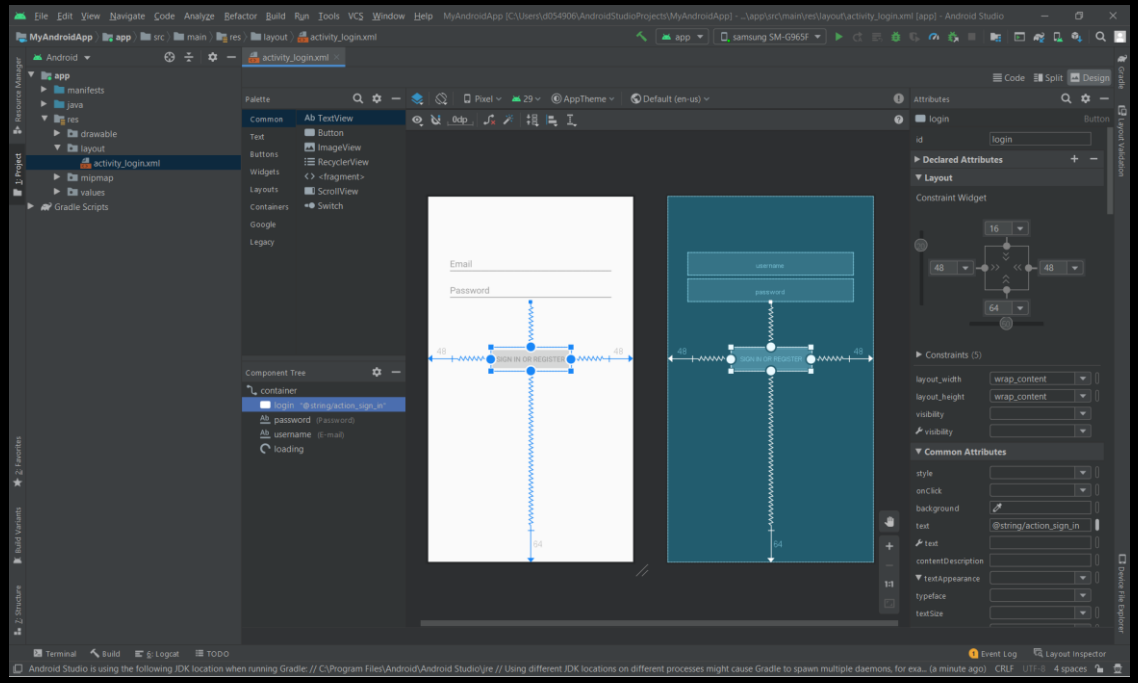
Architektur einer Andoid-App

- Android unterstützt die Entwicklung von Software nach dem MVC-Pattern
- Das MVC-Pattern (*Model-View-Controller*) stellt ein Entwurfsmuster zur Implementierung von grafischen Benutzeroberflächen dar
- Das MVC-Pattern sieht vor, grafische Benutzeroberflächen in 3 Komponenten zu unterteilen
  - Model: Datenhaltung und Datenverarbeitung
  - View: Anzeige der Daten
  - Controller: Benutzerinteraktion



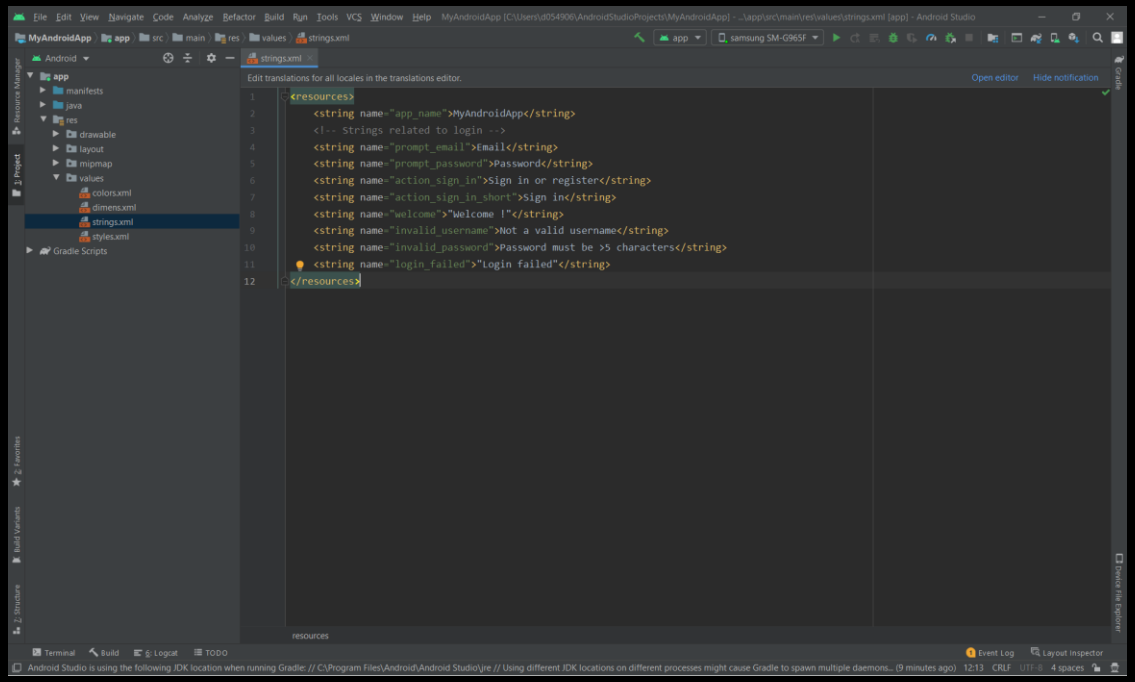
Gestaltung von Oberflächen

- Grafische Oberflächen werden in Android als Layouts bezeichnet
- Der grafische Editor ermöglicht es, Oberflächen entweder textuell oder per Drag-and-Drop zu gestalten
- Jedes Bildschirmelement besitzt verschiedene Eigenschaften wie z.B. id, text, width, height,…
- Navigation: *app – res – layout*
- **Hinweis**: Texte sollten aus Gründen der Wartbarkeit und Mehrsprachigkeit immer als String-Definitionen gepflegt werden (Properties – Text – …)



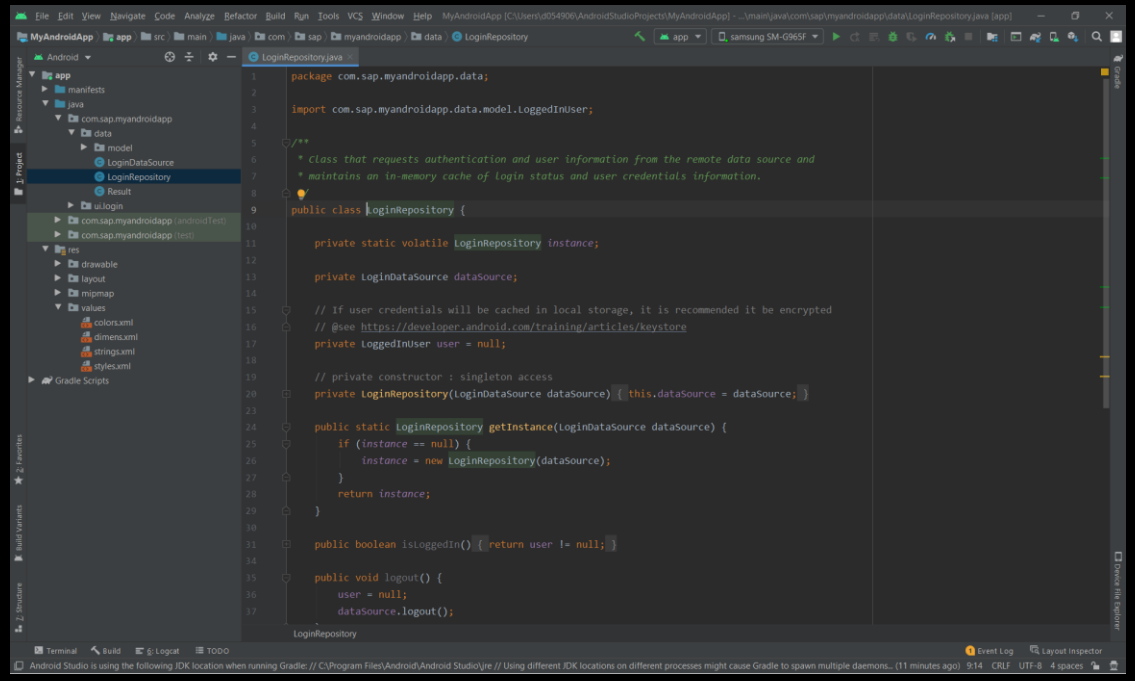
Verwaltung von Texten, Farben, etc.

- Die unterschiedlichen Value-Dateien ermöglichen die zentrale Verwaltung von
  - (sprachabhängigen) Texten (strings.xml),
  - Farben (colors.xml),
  - Farblicher Gestaltung (styles.xml) und
  - Auflösungen (dimens.xml)
- Navigation: *app – res – values*



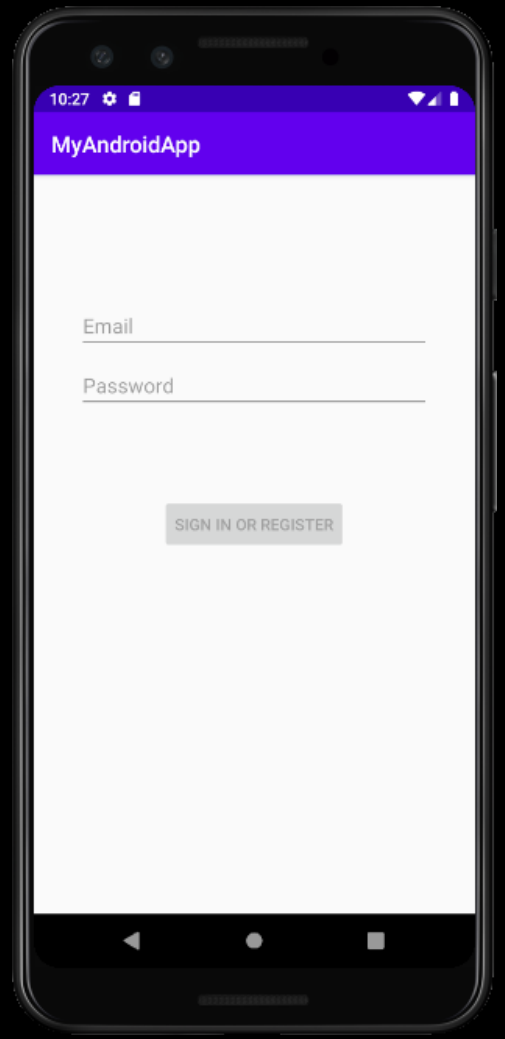
Implementierung der Benutzerinteraktion

- Zu jeder Activity gehört ein entsprechender Controller (Java-Klasse), in welcher die Benutzerinteraktion implementiert ist
- Navigation: *app – java*



Testen von Android-Apps

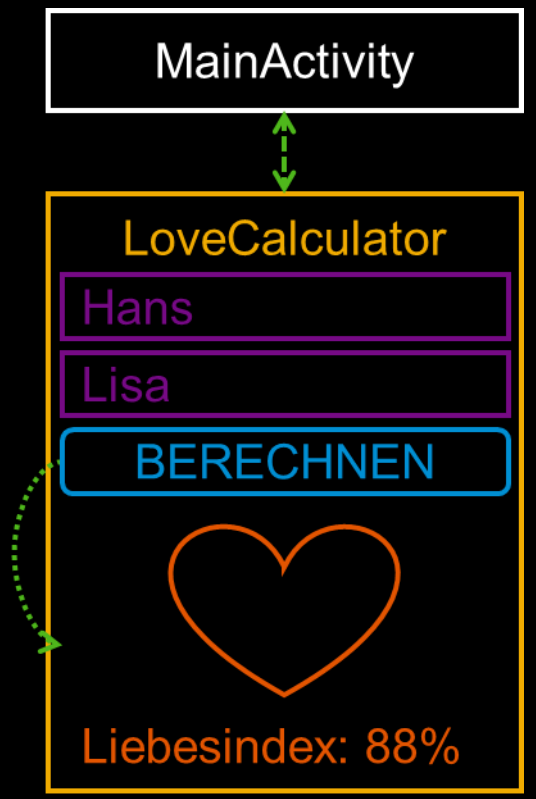
- Zum Testen von Android-Apps können über den *Android Virtual Device Manager* (AVD Manager) Android-Smartphone-Emulator erstellt werden
- Navigation: *Tools – AVD Manager*
- Zudem kann über USB ein Smartphone angeschlossen und zum Testen verwendet werden. Hierzu muss auf dem Smartphone das USB Debugging eingeschalten werden



LoveCalculatorApp

Aufbau der *LoveCalculatorApp*

- Die *LoveCalculatorApp* besteht aus einer Activity, welche die Eingabe zweier Namen sowie die anschließende Berechnung und Ausgabe des Liebesindexes als Grafik und in Textform ermöglicht
- Der Liebesindex berechnet sich nach folgender (wissenschaftlich bestätigter) Formel: $L = (A + B)\ mod\ 100 + 1$
- Legende:
  - L: Liebesindex
  - A: Dezimalwert von Name 1
  - B: Dezimalwert von Name 2

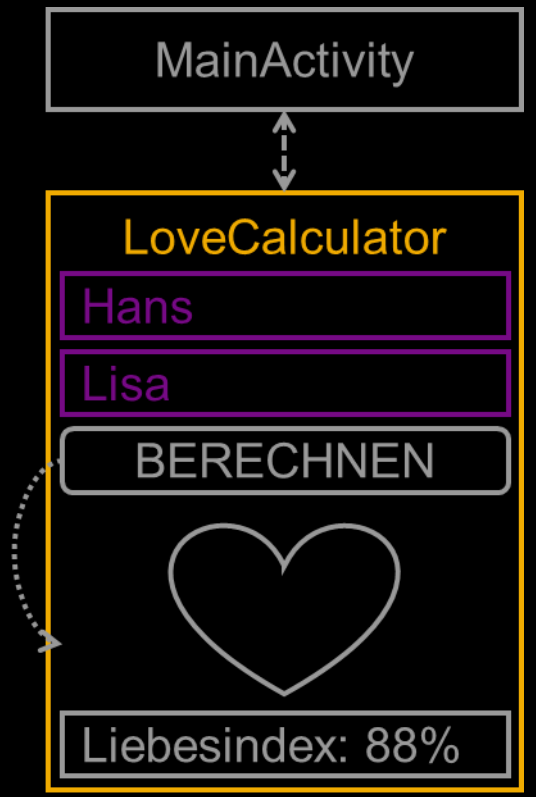


Projekt und Activity anlegen

1. Android Studio starten
2. Neues Projekt mit *Empty Activity* anlegen
   - Application Name: LoveCalculatorApp
   - Activity Name: MainActivity
3. Grafiken in Ordner *app – res – drawable-v24* kopieren

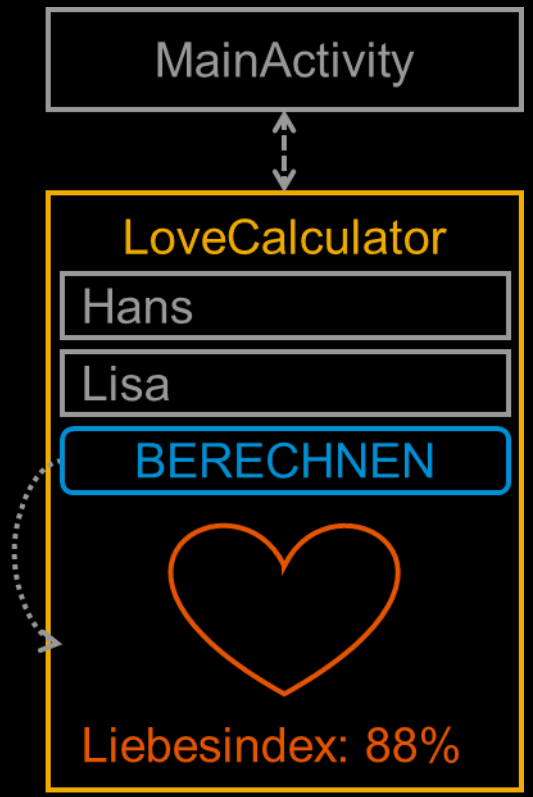
Layout *activity_main.xml* anpassen I

1. Datei *activity_main.xml* öffnen
2. Bildschirmelement *TextView* löschen
3. Layout in ein *Linear Layout (vertical)* ändern
4. Bildschirmelement *Plain Text* hinzufügen
   - id: name1_edit_text
   - text: *leer*
   - hint: Dein Name
5. Bildschirmelement *Plain Text* hinzufügen
   - id: name2_edit_text
   - text: *leer*
   - hint: Sein/Ihr Name



Layout *activity_main.xml* anpassen II

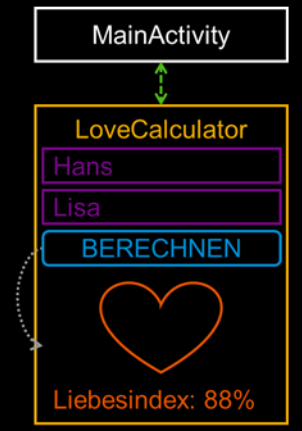
1. Bildschirmelement *Button* hinzufügen
   - id: calculate_button
   - text: Liebesindex Berechnen
2. Bildschirmelement *ImageView* hinzufügen
   - id: love_index_image_view
3. Bildschirmelement *TextView* hinzufügen
   - id: love_index_text_view
   - text: *leer*



Klasse *MainActivity* implementieren I

1. Klasse *MainActivity* öffnen
2. Attribut *name1EditText* deklarieren
3. Attribut *name2EditText* deklarieren
4. Attribut *calculateButton* deklarieren
5. Attribut *loveIndexImageView* deklarieren
6. Attribut *loveIndexTextView* deklarieren
7. Bildschirmelemente zuordnen

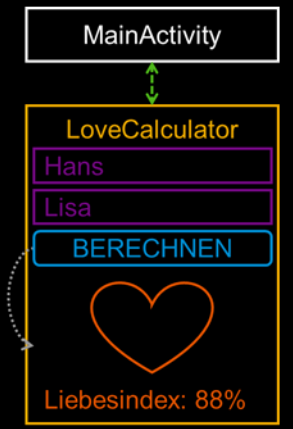
```
public class MainActivity extends AppCompatActivity {

  EditText name1EditText;
  EditText name2EditText;
  Button calculateButton;
  ImageView loveIndexImageView;
  TextView loveIndexTextView;


  public void onCreate(Bundle savedInstanceState) {
    …
    name1EditText = findViewById(R.id.name1_edit_text);
    name2EditText = findViewById(R.id.name2_edit_text);
    calculateButton = findViewById(R.id.calculate_button);
    loveIndexImageView = findViewById(R.id.love_index_image_view);
    loveIndexTextView = findViewById(R.id.love_index_text_view);
  }

}
```



Klasse *MainActivity* implementieren II

1. Ereignisbehandler festlegen
2. Variablen *name1* und *name2* deklarieren und diesen die Werte der Eingabefelder zuweisen
3. **[OPTIONAL]** Prüfen, ob beide Eingabefelder befüllt sind, im Fehlerfall einen Toast anzeigen und Methode beenden
4. Variable *text* deklarieren und dieser die zusammengeführten Werte der Attribute *name1* und *name2* zuweisen

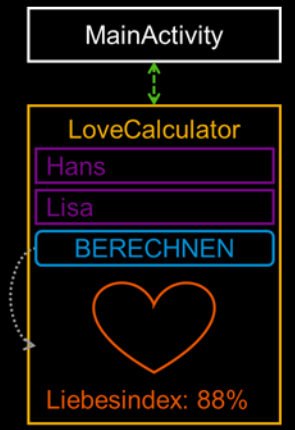
```
public class MainActivity extends AppCompatActivity {
…
  public void onCreate(Bundle savedInstanceState) {
    …
    calculateButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        String name1 = name1EditText.getText().toString();
        String name2 = name2EditText.getText().toString();
        if (name1.equals("") || name2.equals("")) {
          Toast.makeText(this, "Bitte beide Namen eingeben!",
           Toast.LENGTH_LONG).show();
          return;
        }
        String text = name1 + name2;
      }
    });
  }

}
```



Klasse *MainActivity* implementieren III

1. Attribut *total* deklarieren und dieser den berechneten „Gesamtwert“ des zusammengesetzten Namens zuweisen
2. Variable *loveIndex* deklarieren und dieser den berechneten Liebesindex zuweisen

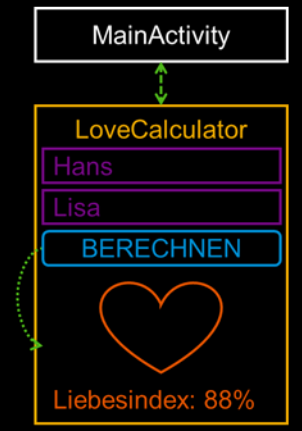
```
public class MainActivity extends AppCompatActivity {
…
  public void onCreate(Bundle savedInstanceState) {
    …
    calculateButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        …
        int total = 0;
        for (int i = 0; i < text.length(); i++) {
          total += text.charAt(i);
        }
        int loveIndex = total % 100 + 1;
      }
    });
  }

}
```



Klasse *MainActivity* implementieren IV

1. Grafik abhängig vom Liebesindex dem Attribut *loveIndexImageView* zuweisen
2. Liebesindex dem Attribut *loveIndexTextView* zuweisen

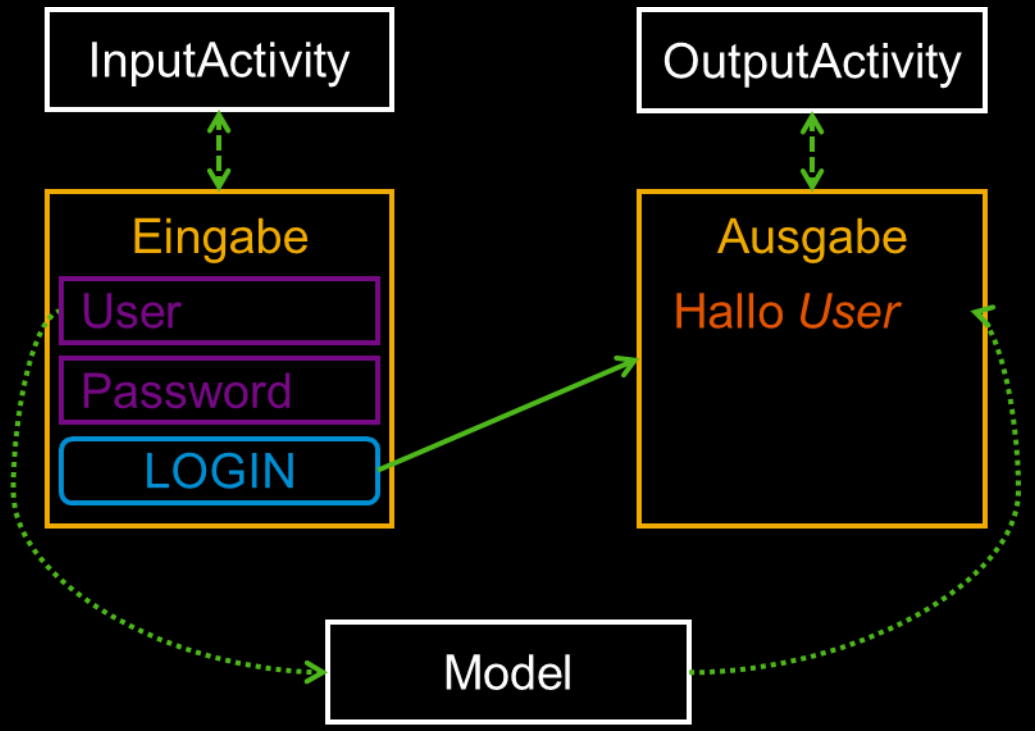
```
public class MainActivity extends AppCompatActivity {
…
  public void onCreate(Bundle savedInstanceState) {
    …
    calculateButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        …
        if (loveIndex < 34) {
          loveIndexImageView.setImageResource(R.drawable.sad);
        } else if (loveIndex < 67) {
          loveIndexImageView.setImageResource(R.drawable.neutral);
        } else {
          loveIndexImageView.setImageResource(R.drawable.happy);
        }
        loveIndexTextView.setText("Liebesindex: " + loveIndex + "%");
      }
    });
  }

}
```



LoginApp

Aufbau der *LoginApp*

- Die *LoginApp* besteht aus zwei Activities sowie einer Model-Klasse
- Die *InputActivity* ermöglicht das Eingeben und Speichern von Login-Daten im Model sowie das Starten der *OutputActivity*
- Die *OutputActivity* ermöglicht das Auslesen und Anzeigen der im Model gespeicherten Login-Daten

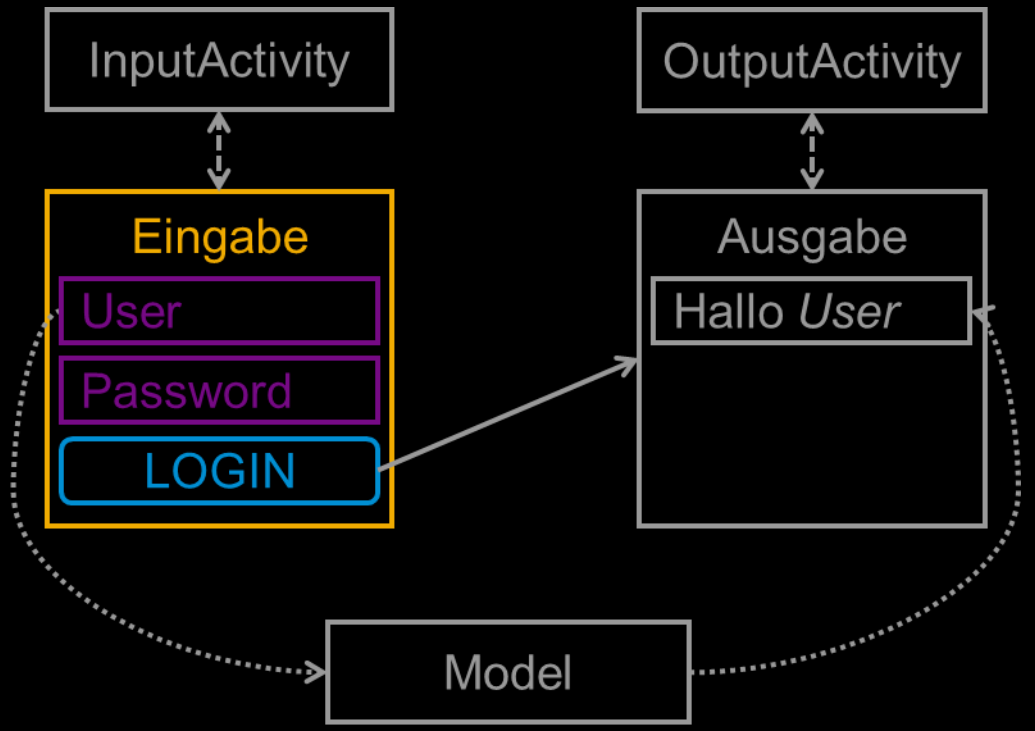


Projekt und Activities anlegen

1. Android Studio starten
2. Neues Projekt mit *Empty Activity* anlegen
   - Application Name: LoginApp
   - Activity Name: InputActivity
3. *Empty Activity* anlegen
   - Activity Name: OutputActivity

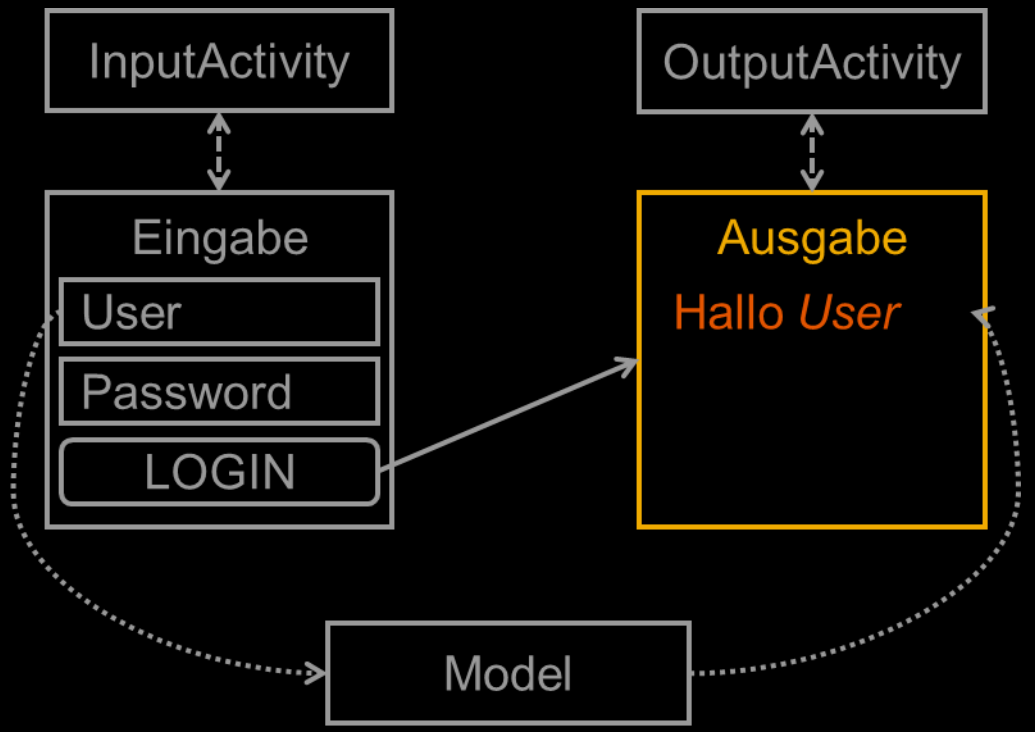
Layout *activity_input.xml* anpassen

1. Datei *activity_input.xml* öffnen
2. Bildschirmelement *TextView* löschen
3. Layout in ein *Linear Layout (vertical)* ändern
4. Bildschirmelement *Plain Text* hinzufügen
   - id: user_edit_text
   - text: *leer*
   - hint: User
5. Bildschirmelement *Plain Text* hinzufügen
   - id: password_edit_text
   - text: *leer*
   - hint: Password
6. Bildschirmelement *Button* hinzufügen
   - id: login_button
   - text: Login



Layout *activity_output.xml* anpassen

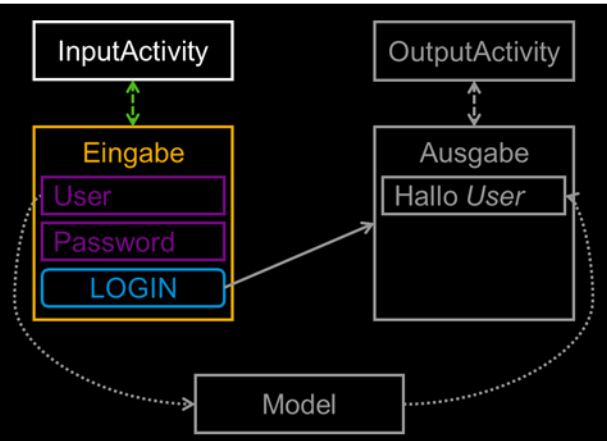
1. Datei *activity_output.xml* öffnen
2. Layout in ein *Linear Layout (vertical)* ändern
3. Bildschirmelement *TextView* hinzufügen
   - id: welcome_text_view
   - text: *leer*



Klasse *InputActivity* implementieren I

1. Klasse *InputActivity* öffnen
2. Attribute *userEditText*, *passwordEditText* und *loginButton* deklarieren
3. Bildschirmelemente zuordnen

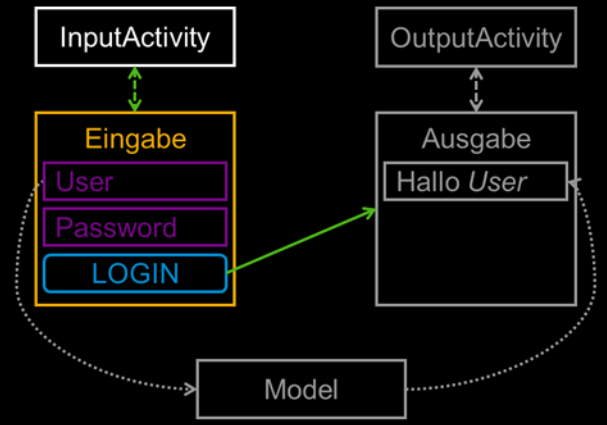
```
public class InputActivity extends AppCompatActivity {

  EditText userEditText;
  EditText passwordEditText;
  Button loginButton;

  public void onCreate(Bundle savedInstanceState) {
    …
    userEditText = findViewById(R.id.user_edit_text);
    passwordEditText = findViewById(R.id.user_edit_text);
    loginButton = findViewById(R.id.login_button);
  }

}
```



Klasse *InputActivity* implementieren II

1. Klasse *InputActivity* öffnen
2. Ereignisbehandler festlegen
3. Intent deklarieren
4. Activity *OutputActivity* starten

```
public class InputActivity extends AppCompatActivity {
…
  public void onCreate(Bundle savedInstanceState) {
    …
    loginButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Intent intent = new Intent(v.getContext(), OutputActivity.class);
        startActivity(intent);
      }
    });
  }

}
```

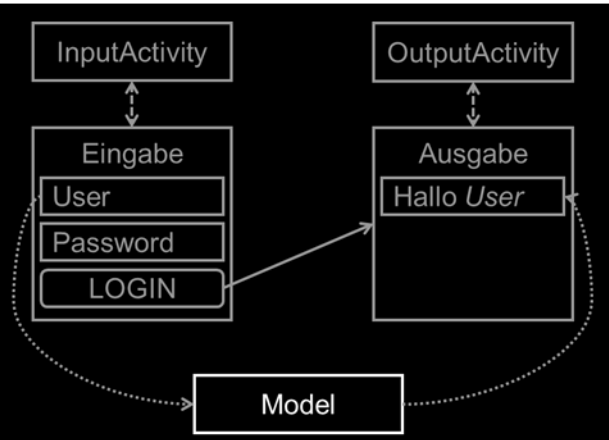


Klasse *Model* erstellen und implementieren

1. Klasse *Model* erstellen und öffnen
2. Attribute *instance*, *user* und *password* deklarieren
3. Konstruktor implementieren
4. Methode *getInstance()* implementieren
5. Setter und Getter implementieren

Hinweis: die Klasse wird als Singleton implementiert. Das Singleton-Pattern stellt sicher, dass zur entsprechenden Klasse genau ein Objekt zur Laufzeit existiert

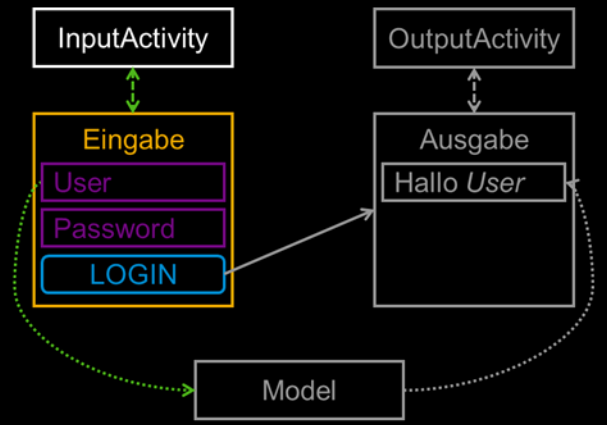
```
public class Model {

  private final static Model instance;
  private String user;
  private String password;


  private Model() { }
  public Model getInstance() {
    if (instance == null) {
      instance = new Model();
    }
    return instance;
  }
  public void setUser(String user) { this.user = user; }
  public String getUser() { return user; }
  public void setPassword(String password) { this.password = password; }
  public String getPassword() { return password; }

}
```



Klasse *InputActivity* implementieren III

1. Klasse *InputActivity* öffnen
2. Login-Daten lesen und zwischenspeichern
3. Model-Instanz lesen
4. Login-Daten im Model speichern

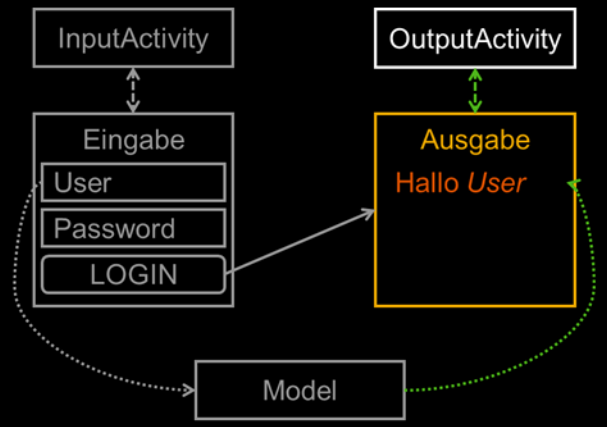
```
public class InputActivity extends AppCompatActivity {
…
  public void onCreate(Bundle savedInstanceState) {
    …
    loginButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        String user = userEditText.getText().toString();
        String password = passwordEditText.getText().toString();
        Model model = Model.getInstance();
        model.setUser(user);
        model.setPassword(password);
        …
      }
    });
  }

}
```



Klasse *OutputActivity* implementieren

1. Klasse *OutputActivity* öffnen
2. Attribute *welcomeTextView*
3. Bildschirmelement zuordnen
4. Model-Instanz lesen
5. Login-Daten aus dem Model lesen
6. Login-Daten ausgeben

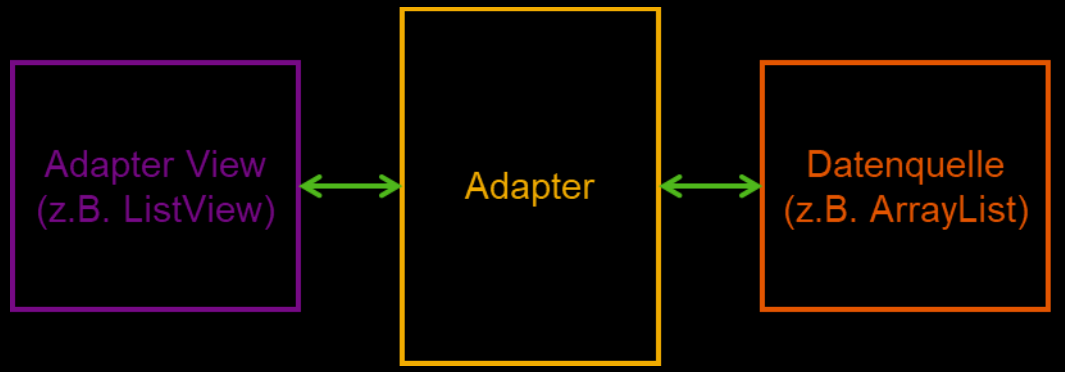
```
public class OutputActivity extends AppCompatActivity {

  TextView welcomeTextView;

  public void onCreate(Bundle savedInstanceState) {
    …
    welcomeTextView = findViewById(R.id.welcome_text_view);
    Model model = Model.getInstance();
    String user = model.getUser();
    welcomeTextView.setText(("Hallo " + user);
  }

}
```



ToDoApp

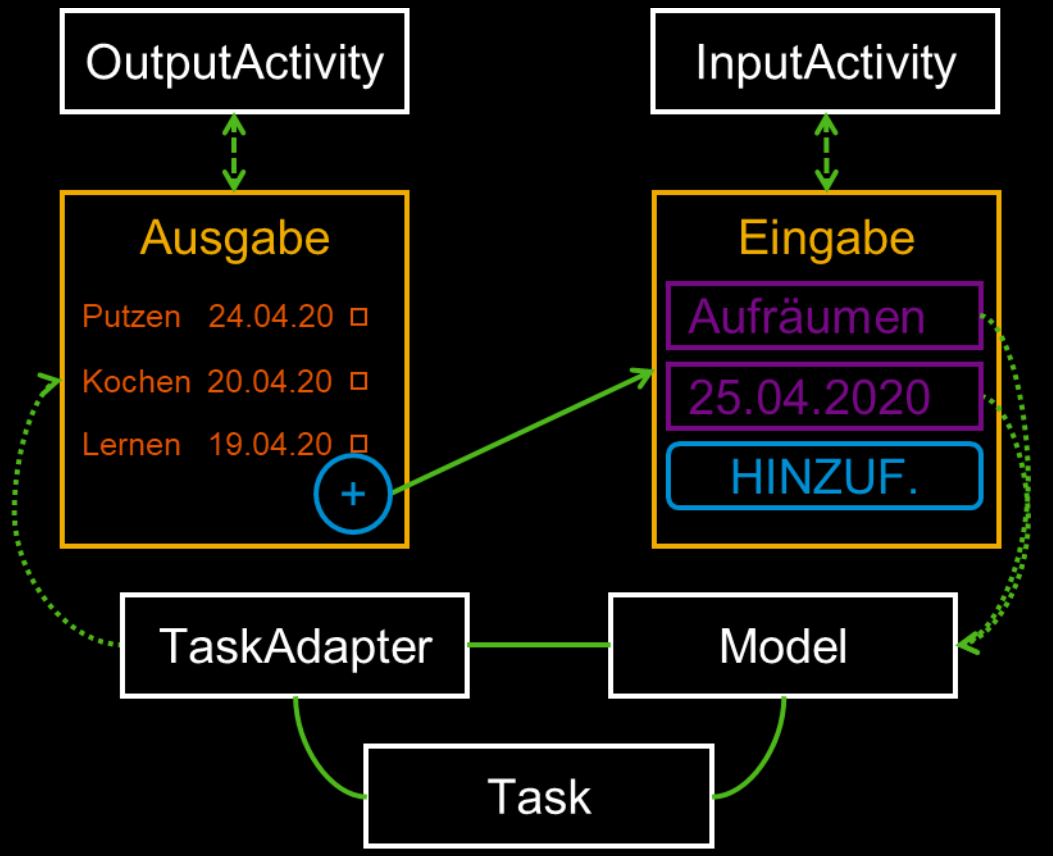
Listenansichten

- Listenansichten sind Ansichtsgruppen, die mehrere Elemente aus einer Datenquelle wie einem Array oder einer Datenbank zusammenfassen und in einer scrollbaren Liste anzeigen
- Die Daten in einer Listansicht werden mit Hilfe einer Adapterklasse an die Listendarstellung gebunden



Aufbau der *ToDoApp*

- Die *ToDoApp* besteht aus zwei Activities, einer Model-Klasse, einer Datenklasse sowie einer Adapter-Klasse
- Die *OutputActivity* zeigt alle Aufgaben in Form einer ListView an und ermöglicht das Starten der InputActivity
- Die *InputActivity* dient zum Erzeugen und Hinzufügen einer neuen Aufgabe

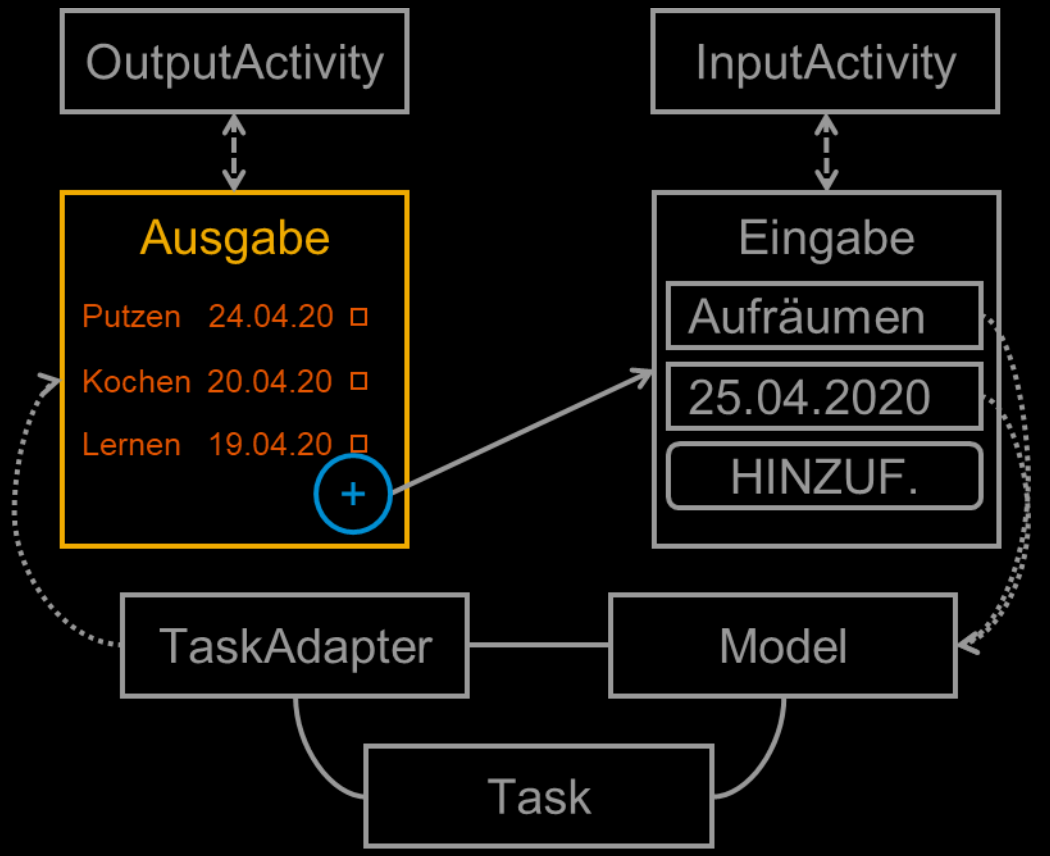


Projekt und Activities anlegen

1. Android Studio starten
2. Neues Projekt mit *Basic Activity* anlegen
   - Application Name: ToDoApp
   - Activity Name: OutputActivity
3. Layout-Datei anlegen
   - File name: item_output
4. *Empty Activity* anlegen
   - Activity Name: InputActivity

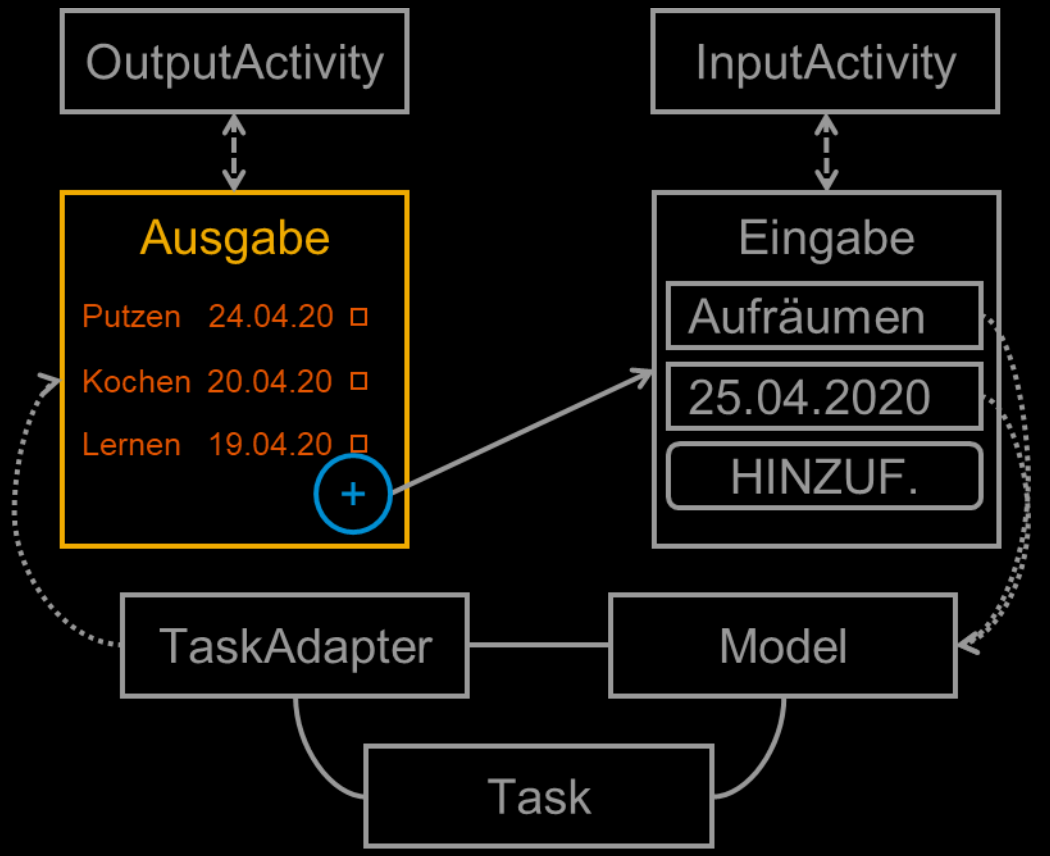
Layout *content_output.xml* anpassen

1. Datei *content_output.xml* öffnen
2. Bildschirmelement *TextView* löschen
3. Bildschirmelement *ListView* hinzufügen
   - id: tasks_list_view



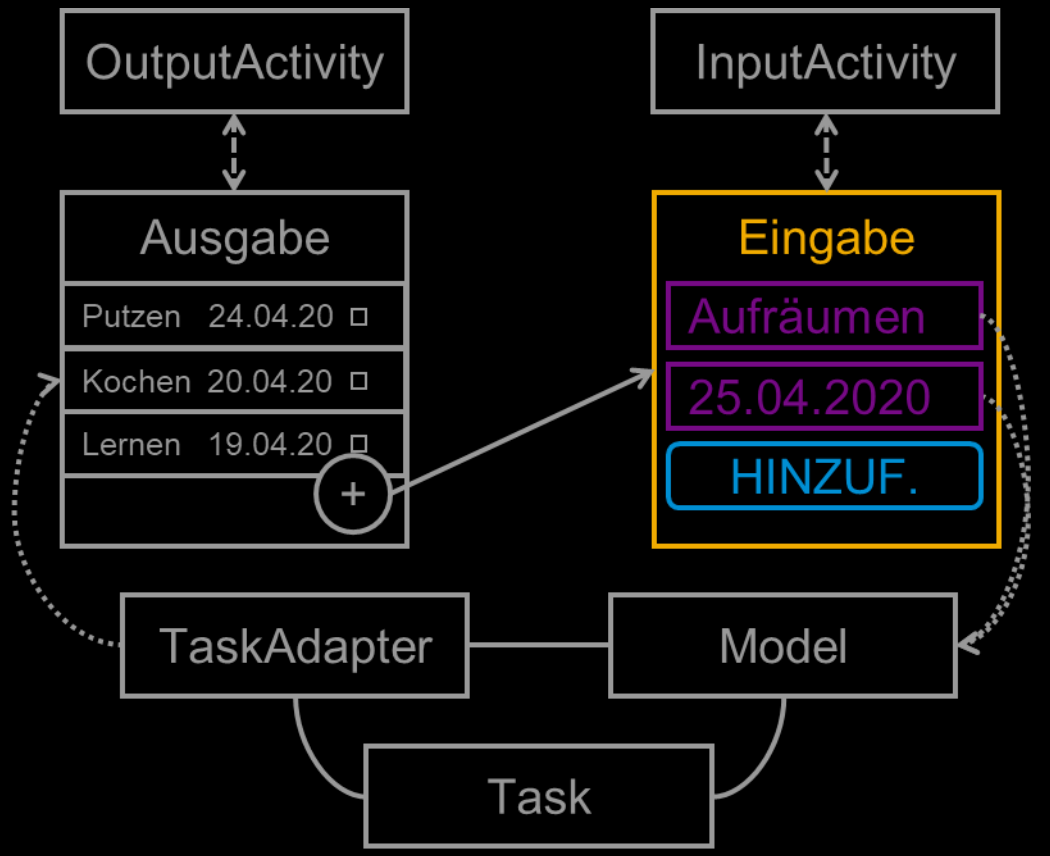
Layout *item_output.xml* anpassen

1. Datei *item_output.xml* öffnen
2. Layout in ein *Linear Layout (vertical)* ändern
3. Bildschirmelement *TextView* hinzufügen
   - id: description_text_view
   - text: *leer*
4. Bildschirmelement *TextView* hinzufügen
   - id: due_date_text_view
   - text: *leer*
5. Bildschirmelement *ChechkBox* hinzufügen
   - id: is_done_check_box
   - text: Erledigt



Layout *activity_input.xml* anpassen

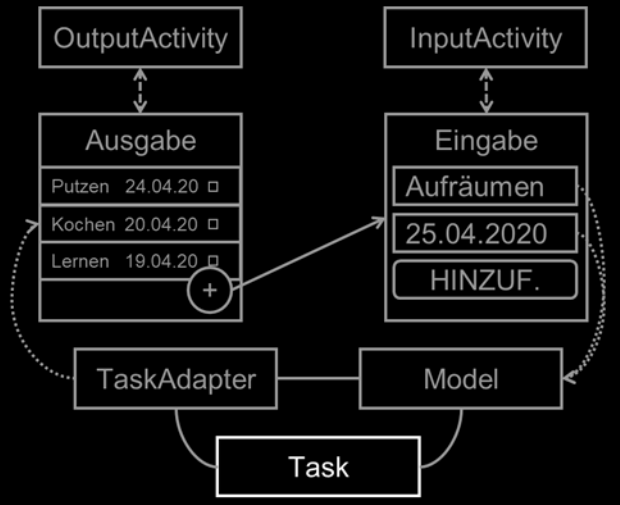
1. Datei *activity_input.xml* öffnen
2. Layout in ein *Linear Layout (vertical)* ändern
3. Bildschirmelement *Plain Text* hinzufügen
   - id: description_edit_text
   - text: *leer*
   - hint: Aufgabenbeschreibung
4. Bildschirmelement *Date* hinzufügen
   - id: due_date_edit_text
   - hint: Fälligkeitsdatum
5. Bildschirmelement *Button* hinzufügen
   - id: add_task_button
   - text: Hinzufügen



Klasse *Task* erstellen und implementieren

1. Klasse *Task* erstellen und öffnen
2. Attribute *description*, *dueDate* und *isDone* deklarieren
3. Konstruktor implementieren
4. Setter und Getter implementieren

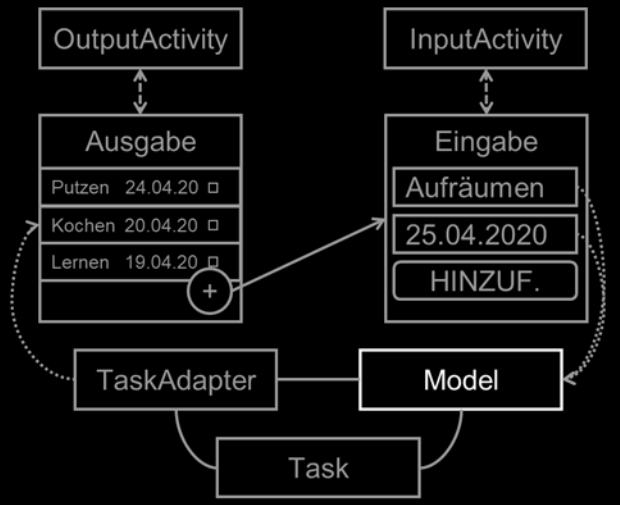
```
public class Task {

  private String description;
  private String dueDate;
  private boolean isDone;

  public Task(String description, String dueDate) {
    this.description = description;
    this.dueDate = dueDate;
  }
  public void setDone(boolean isDone) { this.isDone = isDone; }
  public String getDescription() { return description; }
  public String getDueDate() { return dueDate; }
  public boolean isDone() { return isDone; }

}
```



Klasse *Model* erstellen und implementieren

1. Klasse *Model* erstellen und öffnen
2. Attribute *instance* und *tasks* deklarieren
3. Konstruktor implementieren
4. Methode *getInstance()* implementieren
5. addTask- und getTasks-Methode implementieren

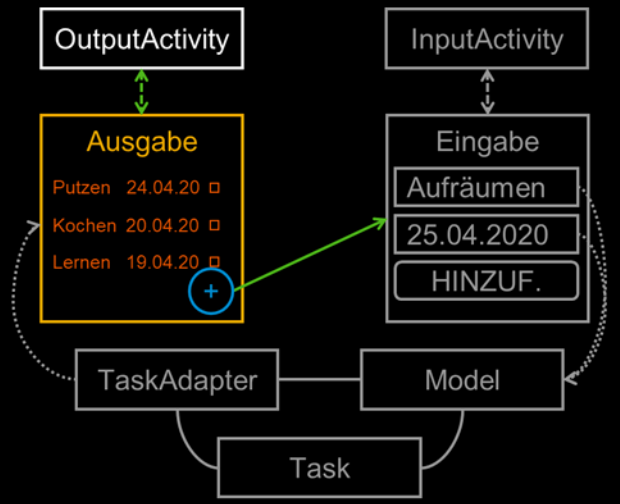
```
public class Model {

  private final static Model instance;
  private ArrayList<Task> tasks = new ArrayList<>();

  private Model() { }
  public Model getInstance() {
    if (instance == null) {
      instance = new Model();
    }
    return instance;
  }
  public void addTask(Task task) { tasks.add(task); }
  public ArrayList<Task> getTasks() { return tasks; }

}
```



Klasse *OutputActivity* implementieren I

1. Intent deklarieren
2. Activity *InputActivity* starten

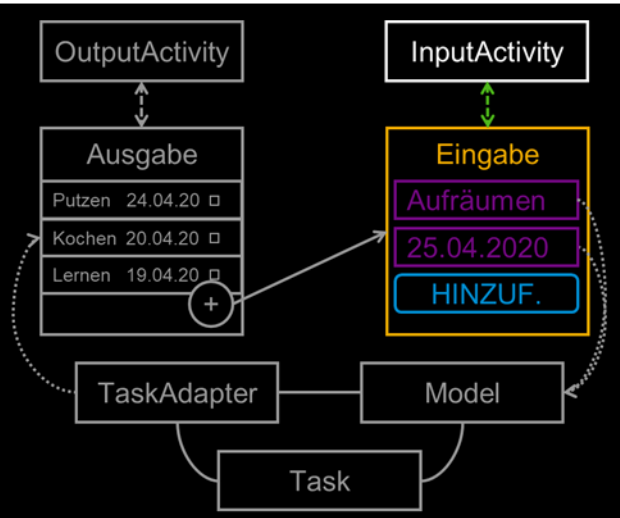
```
public class OutputActivity extends AppCompatActivity {
…
  protected void onCreate(Bundle savedInstanceState) {
    …
    fab.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Intent intent = new Intent(v.getContext(), InputActivity.class);
        startActivity(intent);
      }
    });
  }

}
```



Klasse *InputActivity* implementieren I

1. Attribute *descriptionEditText*, *dueDateEditText* und *addTaskButton* deklarieren
2. Bildschirmelemente zuordnen

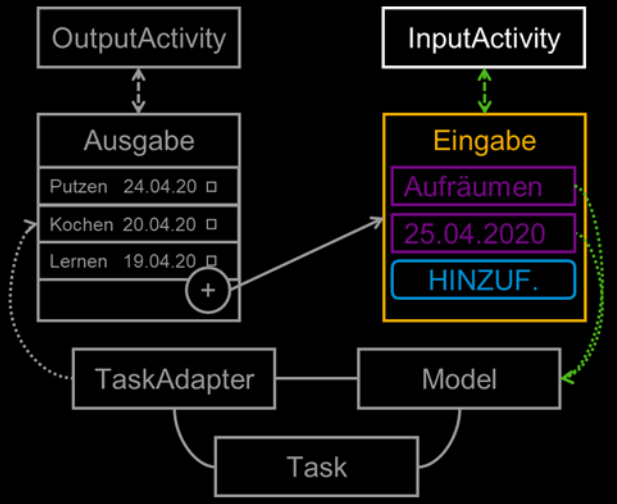
```
public class InputActivity extends AppCompatActivity {

  EditText descriptionEditText;
  EditText dueDateEditText;
  Button addTaskButton;

  protected void onCreate(Bundle savedInstanceState) {
    …
    descriptionEditText = findViewById(R.id.description_edit_text);
    dueDateEditText = findViewById(R.id.due_date_edit_text);
    addTaskButton = findViewById(R.id.add_task_button);
  }

}
```



Klasse *InputActivity* implementieren II

1. Ereignisbehandler festlegen
2. Model-Instanz lesen
3. Aufgaben-Daten lesen und zwischenspeichern
4. Aufgabe erzeugen und der Aufgabenliste im Model hinzufügen
5. Activity schließen

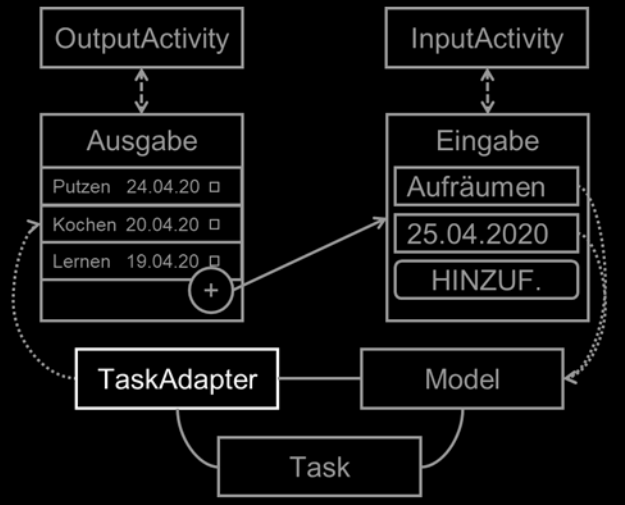
```
public class InputActivity extends AppCompatActivity {
…
  protected void onCreate(Bundle savedInstanceState) {
    …
    addTaskButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Model model = Model.getInstance();
        String description = descriptionEditText.getText().toString();
        String dueDate = dueDateEditText.getText().toString();
        model.addTask(new Item(description, dueDate);
        finish();
      }
    });
  }

}
```



Klasse *TaskAdapter* erstellen und implementieren I

1. Klasse *TaskAdapter* als Unterklasse von *ArrayAdapter<Task>* erstellen und öffnen
2. Konstruktor implementieren
3. Methode *getView(int, View, ViewGroup)* implementieren
4. Layout zuordnen

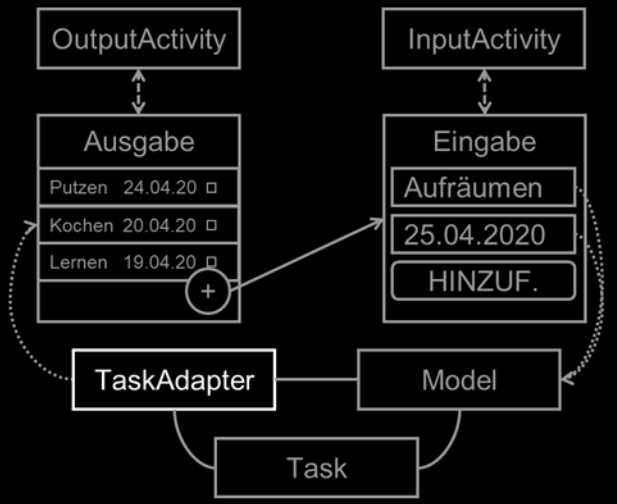
```
public class TaskAdapter extends ArrayAdapter<Task> {

  public TaskAdapter(ArrayList<Task > data) {
    super(data);
  }

  public View getView(int position, View convertView, ViewGroup parent) {
    if (convertView == null) {
      convertView = LayoutInflater.from(parent.getContext()).
       inflate(R.layout.item_output, parent, false);
    }
    return convertView;
  }

}
```



Klasse *TaskAdapter* erstellen und implementieren II

1. Variablen *descriptionTextView, dueDateTextView* und *isDoneCheckBox* deklarieren
2. Bildschirmelemente zuordnen
3. Aufgabe lesen und Daten anzeigen

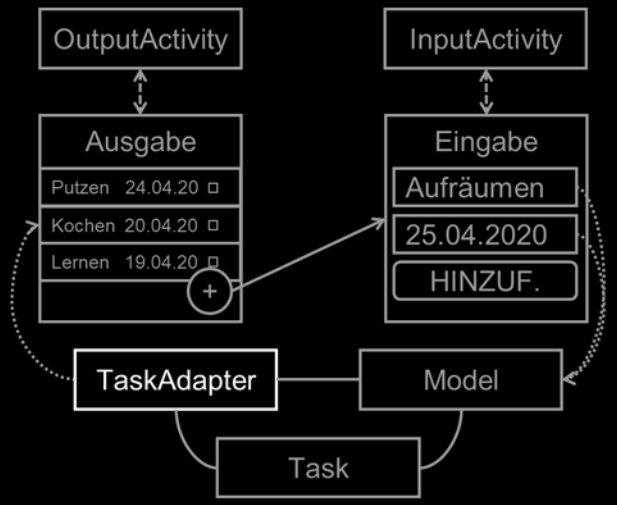
```
public class TaskAdapter extends ArrayAdapter<Task> {
…
  public View getView(int position, View convertView, ViewGroup parent) {
    …
    TextView descriptionTextView =
     convertView.findViewById(R.id.description_text_view);
    TextView dueDateTextView =
     convertView.findViewById(R.id.due_date_text_view);
    CheckBox isDoneCheckBox =
     convertView.findViewById(R.id.is_done_check_box);

    final Task task = getItem(position);
    descriptionTextView.setText(task.getDescription());
    dueDateTextView.setText(task.getDueDate());
    isDoneCheckBox.setChecked(task.isDone());
    …
  }

}
```



Klasse *TaskAdapter* erstellen und implementieren III

1. Ereignisbehandler festlegen
2. Aufgabenstatus ändern

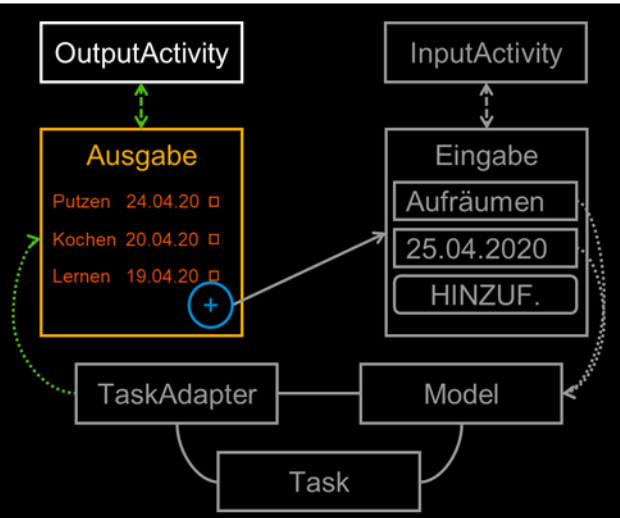
```
public class TaskAdapter extends ArrayAdapter<Task> {
…
  public View getView(int position, View convertView, ViewGroup parent) {
    …
    isDoneCheckBox.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        task.setDone(isDoneCheckBox.isChecked());
      }
    });
    …
  }

}
```



Klasse *OutputActivity* implementieren II

1. Attribut *tasksListView* deklarieren
2. Bildschirmelement zuordnen
3. Model-Instanz lesen
4. Aufgabenliste deklarieren und vom Model lesen
5. Prüfen, ob Aufgaben vorhanden sind, im Erfolgsfall Variable *taskAdapter* deklarieren und dieser die Aufgabenliste zuweisen
6. Dem Attribut *tasksListView* den Adapter *taskAdapter* zuweisen

```
public class OutputActivity extends AppCompatActivity {

  ListView tasksListView;

  protected void onCreate(Bundle savedInstanceState) {
    …
    tasksListView = findViewById(R.id.tasks_list_view);
    Model model = Model.getInstance();
    ArrayList<Item> tasks = model.getTasks();
    if (tasks != null) {
      TaskAdapter taskAdapter = new TaskAdapter(tasks);
      tasksListView.setAdapter(taskAdapter);
    }
  }

}
```



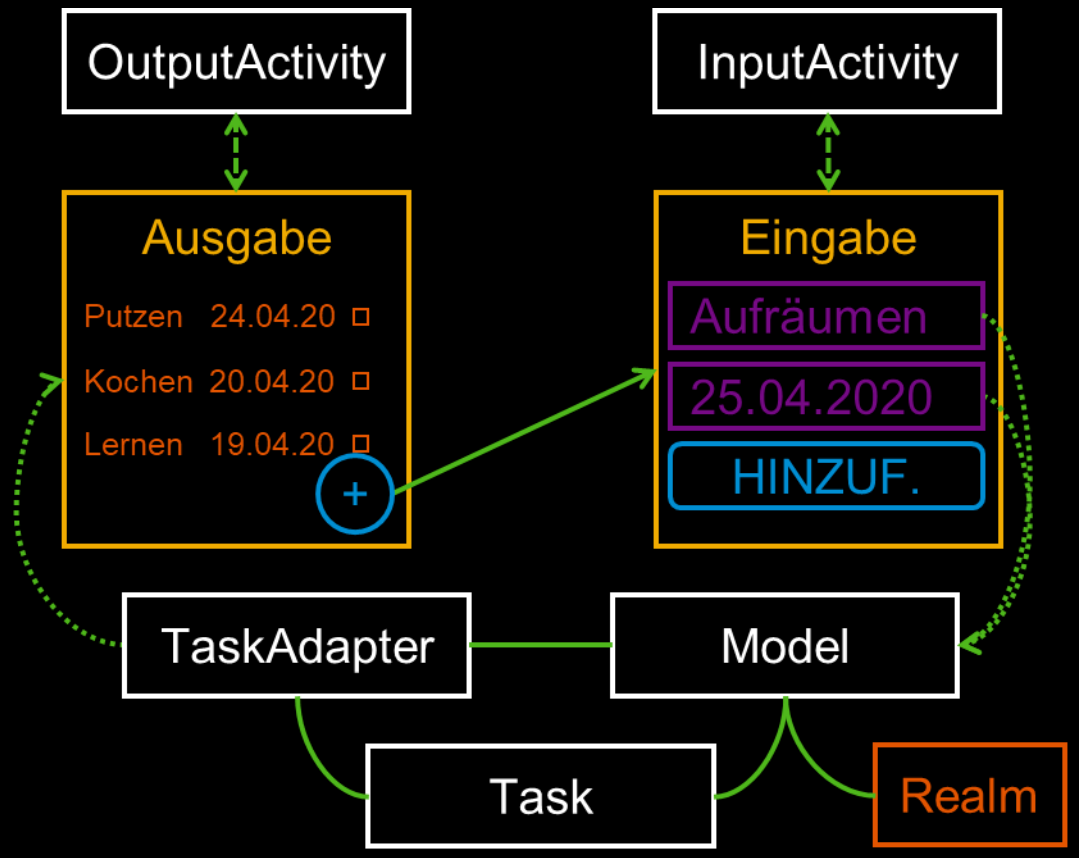
ToDoAppExtended

realm Database

- realm Database stellt eine objektorientierte Datenbank für Android-, iOS- sowie Web-Apps dar
- realm Database ermöglicht eine persistente (also dauerhafte) Speicherung von Daten
- Der Zugriff auf die Daten erfolgt dabei nicht mit Hilfe von SQL, sondern mit Hilfe entsprechender Methoden
- Die zentrale Komponente einer realm Database stellt die Klasse *Realm* dar
- Weiter Infos zur realm Database findet man unter ***realm.io***

Aufbau der *ToDoAppExtended*

- Die *ToDoAppExtended* erweitert die *ToDoApp* um die Möglichkeit, die Aufgaben dauerhaft in einer realm-Datenbank zu speichern



Gradle-Skript *build.gradle (Project: ListViewApp)* konfigurieren

1. Abhängigkeit hinzufügen

```
buildscript {
  …
  dependencies {
    classpath 'io.realm:realm-gradle-plugin:<Version>'
    …
  }

}
…
```

Gradle-Skript *build.gradle (Module: app)* konfigurieren

1. Plugin aktivieren
2. Abhängigkeit hinzufügen
3. Projekt synchronisieren

```
apply plugin: 'com.android.application'
apply plugin: 'realm-android'
…
dependencies {
  implementation 'io.realm:android-adapters:<Version>'
  …
}
…
```

Klasse *Task* anpassen

1. Klasse *Task* von Klasse *RealmObject* ableiten
2. Attribut *id* ergänzen
3. Standardkonstruktor implementieren
4. Konstruktor anpassen
5. Getter ergänzen

```
public class Task extends RealmObject {

  @PrimaryKey
  private int id;
  private String description;
  private String dueDate;
  private boolean isDone;

  public Task() { }
  public Task(String description, String dueDate) {
    Model model = Model.getInstance();
    id = model.getNextTaskId();
    this.description = description;
    this.dueDate = dueDate;
  }
  public void setDone(boolean isDone) { this.isDone = isDone; }
  public int getId() { return id; }
  public String getDescription() { return description; }
  public String getDueDate() { return dueDate; }
  public boolean isDone() { return isDone; }

}
```

Klasse *Model* anpassen

1. Attribut *tasks* löschen
2. Attribut *realm* deklarieren, Realm-Instanz lesen und zuweisen
3. addTask- und getTasks-Methode anpassen
4. getNextTaskId-Methode implementieren

```
public class Model {

  private final static Model instance;
  private ArrayList<Task> tasks = new ArrayList<>();
  private Realm realm = Realm.getDefaultInstance();

  private Model() { }
  public Model getInstance() {
    …
  }
  public void addTask(Task task) {
    try {
      realm.beginTransaction();
      realm.insert(task);
      realm.commitTransaction();
    } catch (RealmPrimaryKeyConstraintException e) { }
  }
  public RealmResults<Task> getTasks() {
    RealmResults<Task> tasks = realm.where(Task.class)
     .equalTo("isDone", false).sort("dueDate").findAll();
    return tasks;
  }
  public int getNextTaskId() {
    Number maxId = realm.where(Task.class).max("id");
    int id = (maxId == null) ? 1 : maxId.intValue() + 1;
    return id;
  }
}
```

Klasse *TaskAdapter* anpassen

1. Superklasse *ArrayAdapter<Task>* durch *RealmBaseAdapater<Task>* ersetzen

2. *ArrayList<Task>* durch *OrderedRealmCollection<Task>* ersetzen

```
public class TaskAdapter extends RealmBaseAdapter<Task> {

  public TaskAdapter(OrderedRealmCollection<Task> data) {
    super(data);
  }

  public View getView(int position, View convertView, ViewGroup parent) {
    …
  }
}
```

Klasse *OutputActivity* anpassen

1. Realm initialisieren
2. Realm-Konfiguration erstellen und zuweisen
3. *ArrayList<Task>* durch *RealmResults<Task>* ersetzen

```
public class OutputActivity extends AppCompatActivity {

  ListView tasksListView;

  protected void onCreate(Bundle savedInstanceState) {
    …
    Realm.init(this);
    RealmConfiguration config = new RealmConfiguration.Builder()
     .deleteRealmIfMigrationNeeded().build();
    Realm.setDefaultConfiguration(config);
    tasksListView = findViewById(R.id.tasks_list_view);
    Model model = Model.getInstance();
    RealmResults<Task> tasks = model.getTasks();
    if (tasks != null) {
      TaskAdapter taskAdapter = new TaskAdapter(tasks);
      tasksListView.setAdapter(taskAdapter);
    }
  }

}
```