Design Pattern – Model, View, Controller/Presenter

Pemisahan Model View dan Presenter pada Aplikasi Nilai Mahasiswa

Dr. Antonius Rachmat C

Topic List

- Model View Controller/Presenter
  - Pengertian
  - How dan Why
  - Kelebihan dan Kekurangan
- Studi Kasus

Model View Controller/Presenter

Intro

Why?

Example

Pros & Cons

MVC

- Pola desain yang memisahkan antara **data, proses bisnis, dan tampilan**
- One model, many views, many controller
- **Model** adalah lapisan data. Ini termasuk objek data, kelas database, dan logika bisnis lainnya, yang berkaitan dengan penyimpanan data, pengambilan data, pembaruan data, dll.
- **View** merender data menggunakan Model dengan cara yang sesuai untuk antarmuka pengguna. Inilah yang ditampilkan kepada pengguna.
- **Controller** adalah otak dari sistem. Ini berisi semua logika sistem, dan mengontrol Model dan View. Pengguna berinteraksi dengan sistem melalui kontrol ini.

Model – View - Controller

Talks to data source to retrieve and store data

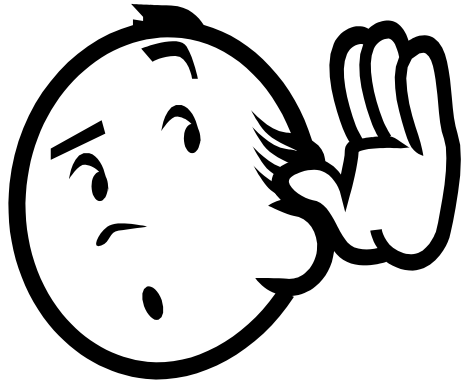
MODEL

Asks model for data and presents it in a user-friendly format

VIEW

Listens for the user to change data or state in the UI, notifying the model or view accordingly

CONTROLLER



The Principle of MVC

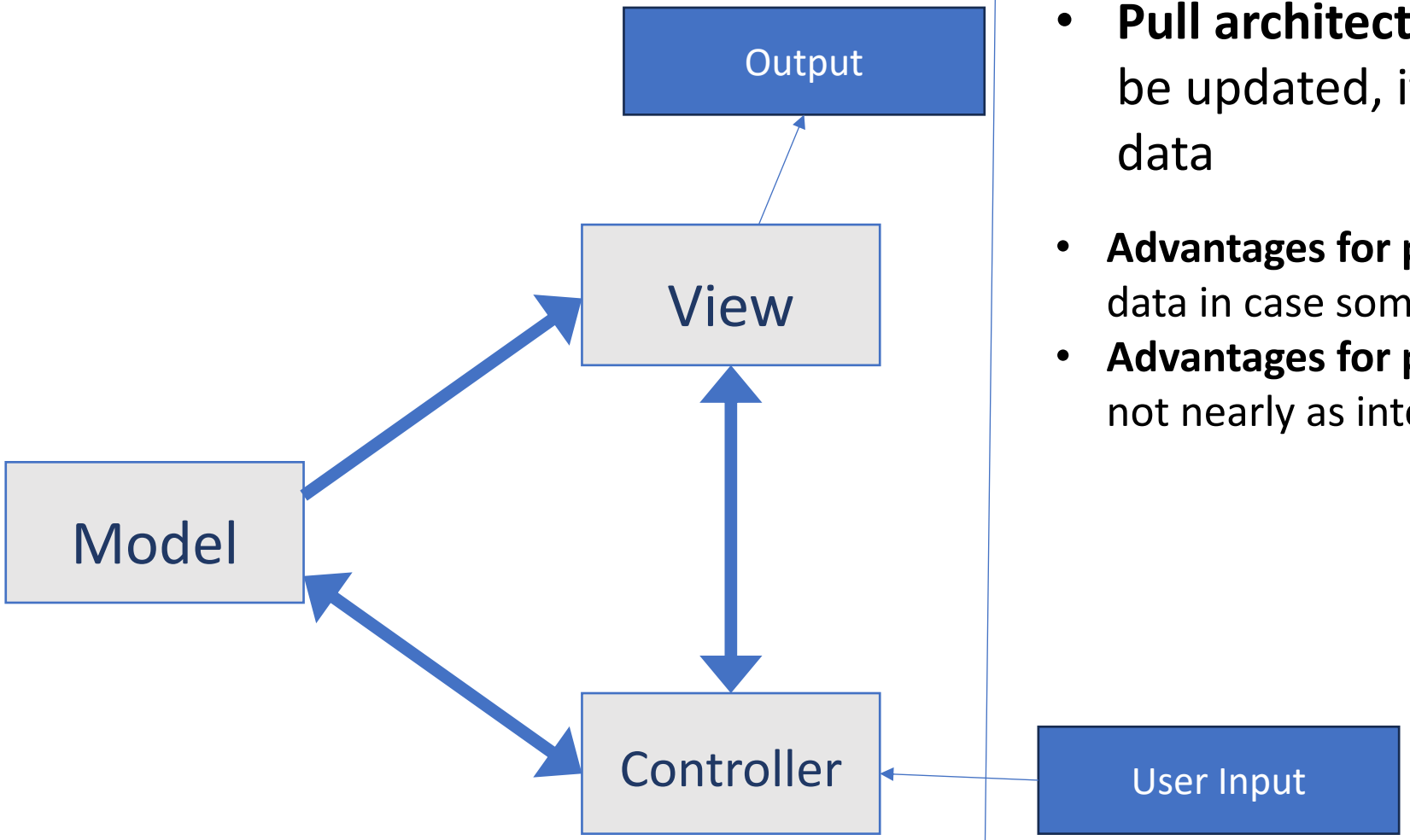
Prinsip dasar:

- **Model** tidak peduli dengan bagaimana data akhirnya ditampilkan kepada pengguna.
- **View** tidak mementingkan nilai data aktual dan bagaimana caranya ditampilkan, hanya yang penting nilai apa yang perlu ditampilkan.
- **Controller** bertindak sebagai perekat antara kedua komponen ini, mengatur bagaimana data harus ditampilkan
- Prinsip ini memberikan dua keuntungan: **separation of concerns and unit testability**.

MVC – Pros dan Cons

- **Pros:**
  - Organisasi kode lebih baik
  - Rapid application development
  - Code reuse / flexibility
  - Parallel development
  - Memungkinkan pengembangan banyak view (misalnya untuk mobile, web, desktop) dengan controller yang sama dan database yang sama
- **Cons:**
  - Butuh belajar lebih
  - Akan rumit jika aplikasinya sederhana

MVC Flow in Theory

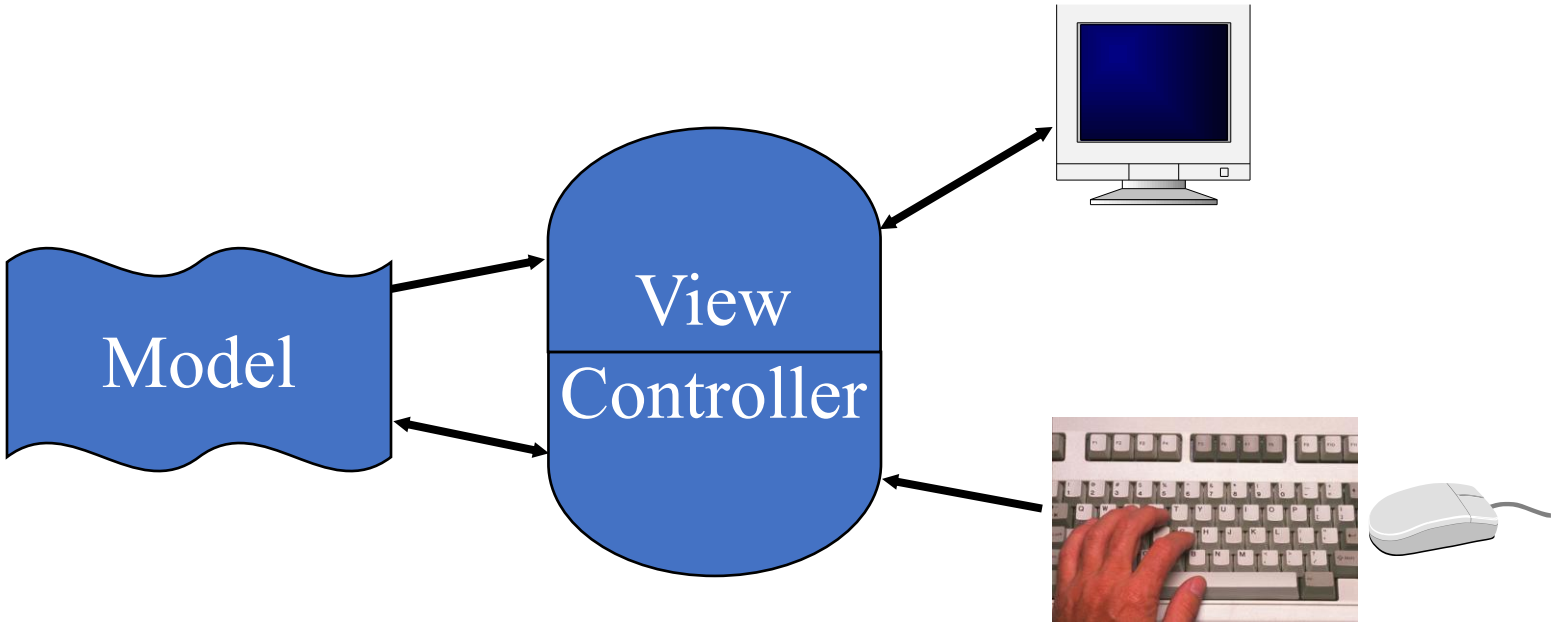


- **Push architecture:** As soon as the model changes, it notifies all of the views
- **Pull architecture:** When a view needs to be updated, it asks the model for new data

- **Advantages for push**: Guaranteed to have latest data in case something goes wrong later on
- **Advantages for pull**: Avoid unnecessary updates, not nearly as intensive on the view

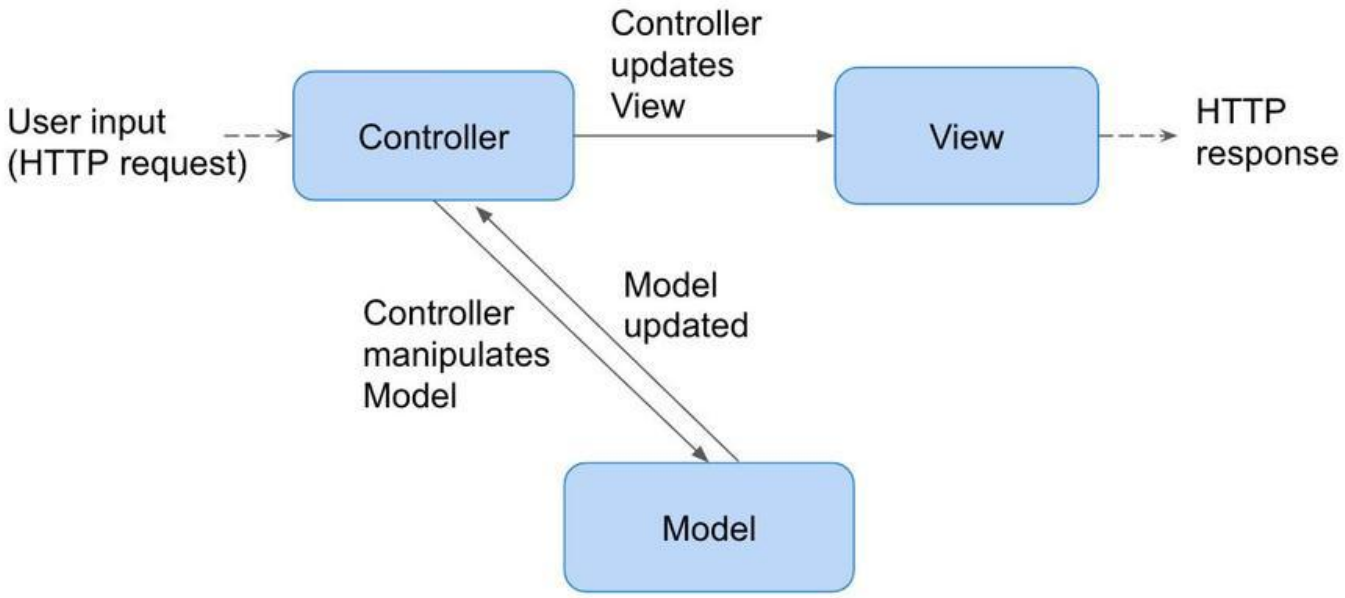
Bentuk MVC lainnya

- Combining Controller and View
  - Small programs
  - May be highly interdependent



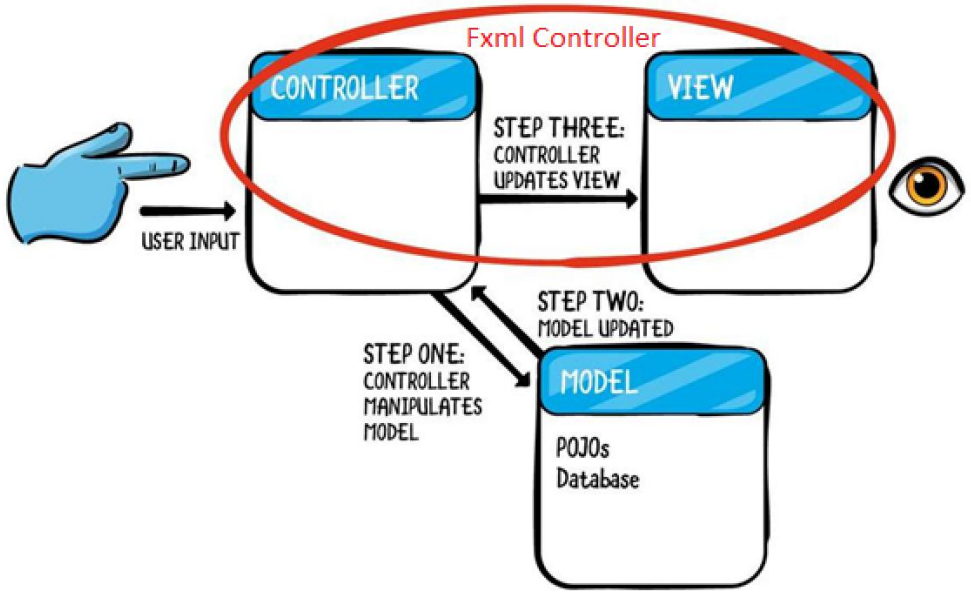
Intro – MVC vs MVP

- **Model View Presenter** adalah turunan dari desain arsitektur Model View Controller.
- **Model-View-Controller (MVC)** adalah pola desain terkenal di bidang **pengembangan web**.
- Dalam MVC, Model adalah **entitas data** dalam sebuah aplikasi. Sedangkan Controller dapat diartikan sebagai **penghubung** antara **View** dengan **Model**.



Intro – The problem of MVC

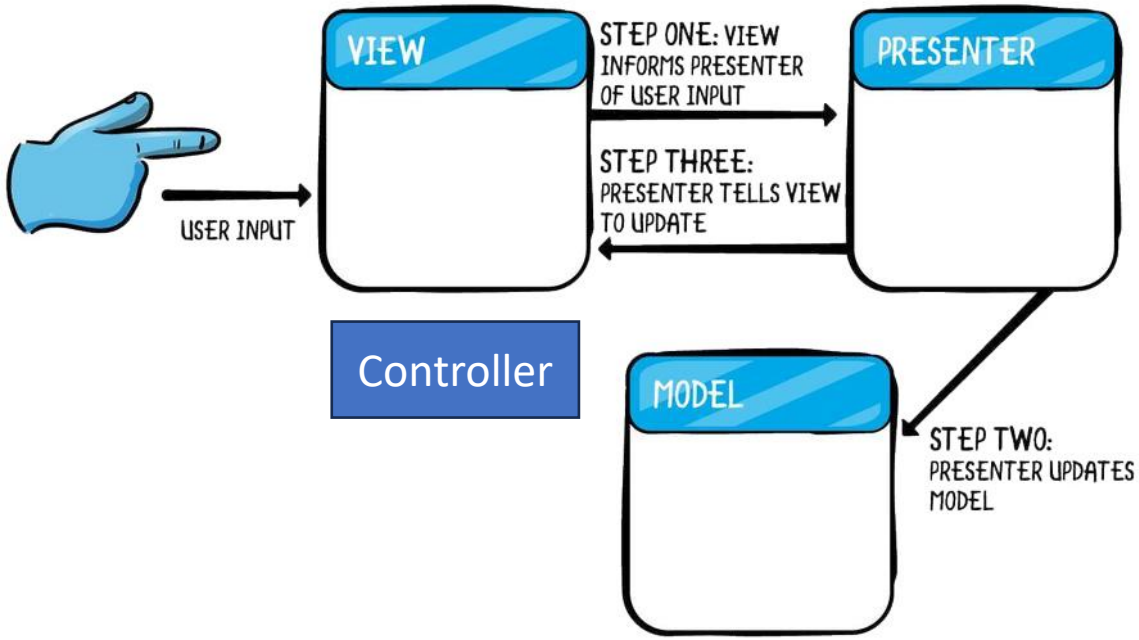
- Pada JavaFX saat **Controller mengupdate View**, View tidak dapat diperbarui jika hanya berupa layout .fxml **statis**.
- Sebagai pengontrol, **ControllerView hampir selalu harus berisi logika untuk view** sebagai respons terhadap inputan pengguna, yang melanggar prinsip controller.
- Selain itu, **tidak semua view dideploy melalui .fxml**; bagaimana jika view Controller memuat layout secara dinamis?
- Akibatnya, **View Controller** secara efektif berfungsi sebagai **Controller** dan **View**.
  - Menyalahi prinsip dan tujuan MVC



Intro – MVC vs MVP

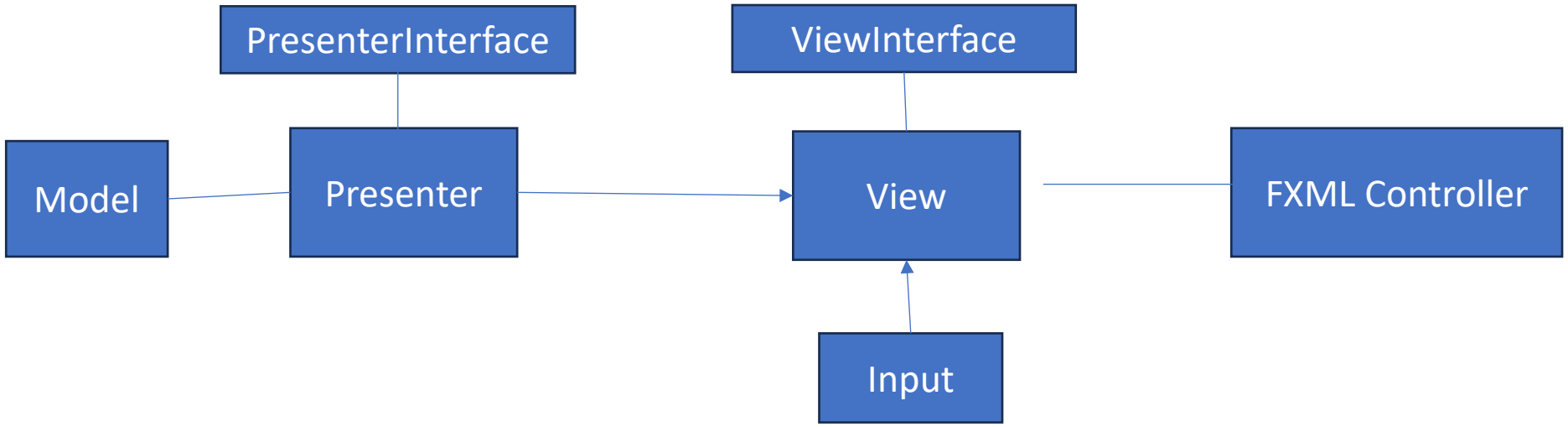
- Pada pattern MVP, **View tidak berkomunikasi langsung dengan Model** melainkan harus melalui **perantara** atau bisa juga disebut dengan **Contract** (dalam hal ini **Presenter** bertugas hanya sebagai **Interface** antara **Model** dan **View**)

  - **Model** adalah lapisan data, yang bertanggung jawab atas bisnis logic.
  - **View** menampilkan UI dan respon tindakan pengguna (biasanya berupa controller dari FXML)
  - **Presenter** mengatur logika komunikasi antara Model dan View.



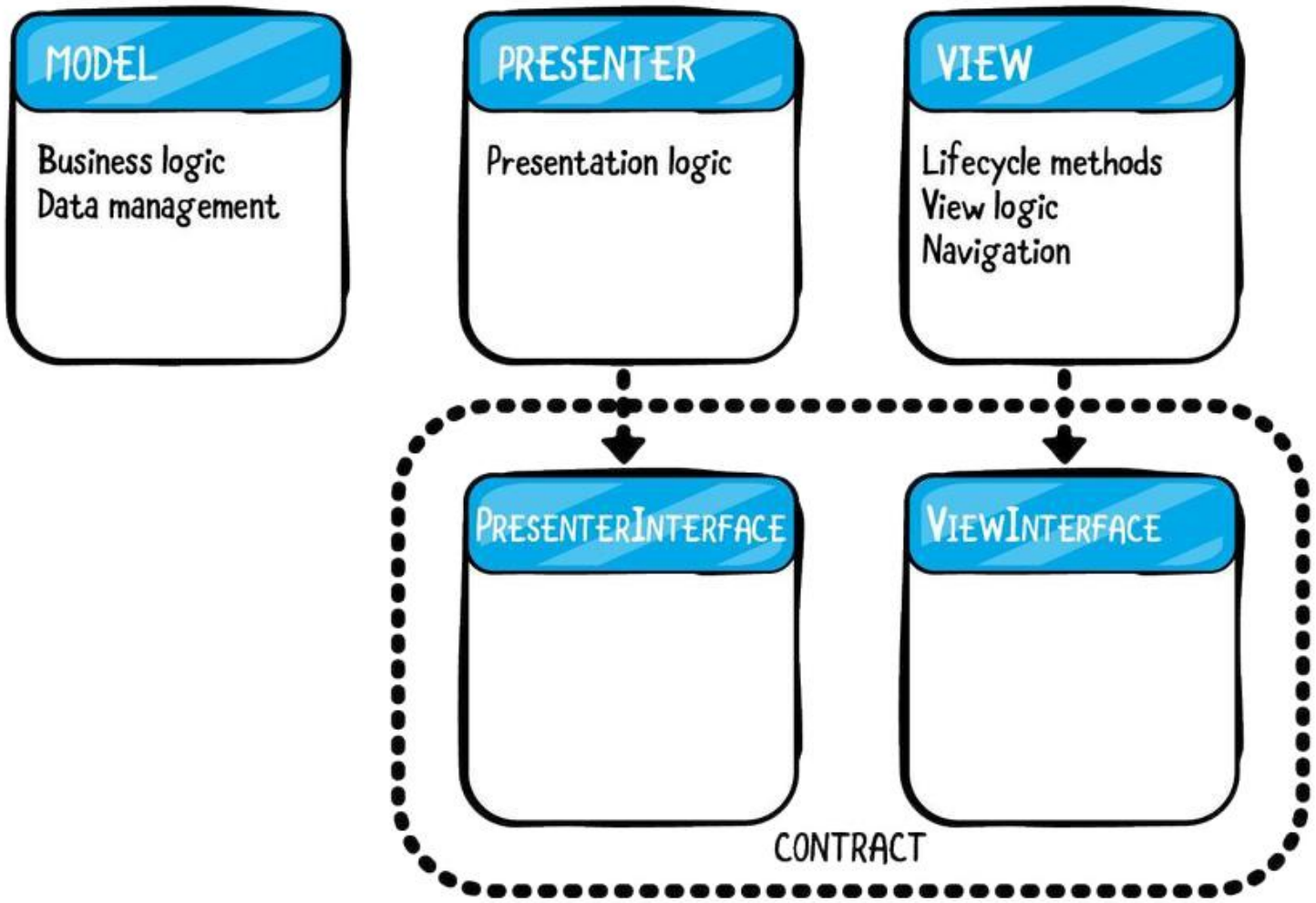
Intro – How to apply MVP ?

- Buat **interface** untuk **Presenter** dan **View**, kedua interface dibuat dalam satu kelas yang disebut kelas **Kontrak**.
- **Presenter** mengimplementasikan **PresenterInterface** dan **fxml controller** mengimplementasikan **ViewInterface**.
- Kelas **Presenter** kemudian akan memiliki referensi ke sebuah instance interface **ViewInterface**, bukan referensi langsung ke fxml controller.



Intro – How to apply MVP ?

- Class **contract** hanya berisi relasi antara **Presenter** dan **View**



Intro – applying MVP example

View > Nilai MahasiswaContract

- Mendefinisikan **Contract** antara Fxml controller dan Presenter

```
interface NilaiMhsContract {
    interface View {

        void showError(String errorMessage);

        void onSuccess(String message);

        ObservableList<Mahasiswa> getObservableList();
    }

    interface Presenter {

        void geAllNilaiMahasiswa();

        void updateNilaiMahasiswa(Mahasiswa mahasiswaOld, Mahasiswa mahasiswaNew);

        void addNilaiMahasiswa(Mahasiswa mahasiswa);

        void deleteNilaiMahasiswa(Mahasiswa selectedItem);
    }
}
```

Intro –applying MVP example

- Membuat class **Presenter** dan mengimplementasikan **Contract**

```
public class NilaiMhsPresenter implements NilaiMhsContract.Presenter {
    private final NilaiMhsContract.View view;
    private Dao<Mahasiswa,String> mahasiswaRepository = new
MahasiswaRepository();

    public NilaiMhsPresenter(NilaiMhsController view) {
        this.view = view;
    }

    @Override
    public void geAllNilaiMahasiswa() {
        view.getObservableList().addAll(mahasiswaRepository.findAll());
    }

    @Override
    public void updateNilaiMahasiswa(Mahasiswa mahasiswaOld, Mahasiswa
mahasiswaNew) {
        if (mahasiswaRepository.update(mahasiswaOld, mahasiswaNew)) {
            int iOldMahasiswa =
view.getObservableList().indexOf(mahasiswaOld);
            view.getObservableList().set(iOldMahasiswa, mahasiswaNew);
            view.onSuccess("Nilai Mahasiswa berhasil diupdate!");
        }
    } …..
```

Intro –applying MVP example

- Mengimplementasikan **Contract** pada **View** (Fxml controller)
- Panggil fungsi-fungsi dari presenter, misalnya:

```
public class NilaiMhsController implements Initializable, NilaiMhsContract.View {

private NilaiMhsContract.Presenter presenter;

@Override
public void initialize(URL location, ResourceBundle resources) {
    presenter = new NilaiMhsPresenter(this);

    presenter.geAllNilaiMahasiswa();
```

Studi Kasus NilaiMahasiswa

- Pisahkan menjadi MVP:
  - Data object
    - Tetap (OOP)
  - Controller
    - Tetap (menggunakan Validator (CoR))
  - Model
    - Tetap (menggunakan Singleton)
  - Repository
    - Tetap (menggunakan Data Access Object)
  - Util
    - Tetap
  - Manager
    - Tetap (menggunakan Observer)
  - **View**
    - **LineChart, PieChart, NilaiMhs**

- Class controller sebelumnya mengimplementasikan interface View
- PresenterImpl mengimplementasikan interface Presenter

View > ChartContract

```
interface ChartContract {
    interface View <DataType>{

        void showError(String errorMessage);

        void onSuccess(String message);

        void preparedDataSucceed(List<Object> dataList);
    }

    interface Presenter {
        void prepareData();
    }
}
```

View > Chart > LineChartMhsPresenter

```
public class LineChartMhsPresenter implements ChartContract.Presenter {
    private final ChartContract.View<Mahasiswa> view;
    private Dao mahasiswaRepository = new MahasiswaRepository();

    public LineChartMhsPresenter(LineChartMhsController view) {
        this.view = view;
    }

    @Override
    public void prepareData() {
        List<Mahasiswa> result = mahasiswaRepository.findAll();
        if (result.isEmpty()) {
            view.showError("data is empty");
            return;
        }
        List<Object> result2 = new ArrayList<Object>(result);
        view.preparedDataSucceed(result2);
    }

}
```

View > Chart > LineChartMhsController (1)

```
public class LineChartMhsController implements Initializable, ChartContract.View<Mahasiswa> {
    @FXML
    private LineChart lineChart;
    private ChartContract.Presenter presenter;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        presenter = new LineChartMhsPresenter(this);
        presenter.prepareData();
    }

    public void showError(String errorMessage) {
        Alert alert = new Alert(Alert.AlertType.ERROR, errorMessage);
        alert.show();
    }

    @Override
    public void onSuccess(String message) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION, message);
        alert.show();
    }
```

View > Chart > LineChartMhsController (2)

```
@Override
public void preparedDataSucceed(List<Object> dataList) {
    XYChart.Series series = new XYChart.Series();
    series.setName("Persebaran Nilai Mahasiswa");
    for (Object data : dataList) {
        Mahasiswa data2 = (Mahasiswa) data;
        series.getData().add(new XYChart.Data<>(data2.getNim(), data2.getNilai()));
    }
    lineChart.getData().add(series);
}
```

```java
public class PieChartMhsPresenter implements ChartContract.Presenter {
    private final ChartContract.View<JumlahMhsByMark> view;
    private Dao mahasiswaRepository = new MahasiswaRepository();

    public PieChartMhsPresenter(PieChartMhsController view) {
        this.view = view;
    }

    @Override
    public void prepareData() {
        List<Object> result = mahasiswaRepository.getJumlahMhsByMark();
        if (result.isEmpty()) {
            view.showError("data is empty");
            return;
        }
        view.preparedDataSucceed(result);
    }

}
```

View > Chart > **PieChartPresenter**

View > Chart > PieChartController (1)

```
public class PieChartMhsController implements Initializable,
ChartContract.View<JumlahMhsByMark> {
    @FXML
    private PieChart pieChart;

    ChartContract.Presenter presenter;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        presenter = new PieChartMhsPresenter(this);
        presenter.prepareData();
    }

    @Override
    public void showError(String errorMessage) {
        Alert alert = new Alert(Alert.AlertType.ERROR, errorMessage);
        alert.show();
    }
```

View > Chart > PieChartController (2)

```
@Override
public void onSuccess(String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION, message);
    alert.show();
}

@Override
public void preparedDataSucceed(List<Object> dataList) {
    // do your GUI stuff here
    ObservableList<PieChart.Data> pieChartData = FXCollections.observableArrayList();
    for (Object data : dataList) {
        JumlahMhsByMark data2 = (JumlahMhsByMark) data;
        pieChartData.add(new PieChart.Data(data2.getMark(), data2.getJumlah()));
    }
    pieChart.setData(pieChartData);
    pieChart.setTitle("Jumlah Nilai berdasarkan Mark");
    pieChart.setClockwise(true);
    pieChart.setLabelLineLength(50);
    pieChart.setLabelsVisible(true);
    pieChart.setStartAngle(180);
}
```

View > NilaiMhsContract

```
interface NilaiMhsContract {
    interface View {

        void showError(String errorMessage);

        void onSuccess(String message);

        ObservableList<Mahasiswa> getObservableList();
    }

    interface Presenter {

        void geAllNilaiMahasiswa();

        void updateNilaiMahasiswa(Mahasiswa mahasiswaOld, Mahasiswa mahasiswaNew);

        void addNilaiMahasiswa(Mahasiswa mahasiswa);

        void deleteNilaiMahasiswa(Mahasiswa selectedItem);
    }
}
```

View > NilaiMhsPresenter (1)

```
public class NilaiMhsPresenter implements NilaiMhsContract.Presenter {
    private final NilaiMhsContract.View view;
    private Dao<Mahasiswa,String> mahasiswaRepository = new MahasiswaRepository();

    public NilaiMhsPresenter(NilaiMhsController view) {
        this.view = view;
    }

    @Override
    public void geAllNilaiMahasiswa() {
        view.getObservableList().addAll(mahasiswaRepository.findAll());
    }

    @Override
    public void updateNilaiMahasiswa(Mahasiswa mahasiswaOld, Mahasiswa mahasiswaNew) {
        if (mahasiswaRepository.update(mahasiswaOld, mahasiswaNew)) {
            int iOldMahasiswa = view.getObservableList().indexOf(mahasiswaOld);
            view.getObservableList().set(iOldMahasiswa, mahasiswaNew);
            view.onSuccess("Nilai Mahasiswa berhasil diupdate!");
        }
    }
```

View > NilaiMhsPresenter (2)

```
@Override
    public void addNilaiMahasiswa(Mahasiswa mahasiswa) {
        if (mahasiswaRepository.save(mahasiswa)) {
            view.getObservableList().add(mahasiswa);
            view.onSuccess("Nilai Mahasiswa berhasil ditambah!");
        }
    }

    @Override
    public void deleteNilaiMahasiswa(Mahasiswa mahasiswa) {
        if (mahasiswaRepository.delete(mahasiswa)) {
            view.getObservableList().remove(mahasiswa);
            view.onSuccess("Nilai Mahasiswa berhasil dihapus!");
        }
    }

}
```

View > NilaiMhsController (1)

```
public class NilaiMhsController implements Initializable, NilaiMhsContract.View {

    …. (sama)
    private NilaiMhsContract.Presenter presenter;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        presenter = new NilaiMhsPresenter(this);
        mahasiswaFilteredList = new FilteredList<>(FXCollections.observableList(FXCollections.observableArrayList()));
        table.setItems(mahasiswaFilteredList);
        searchBox.textProperty().addListener(
                (observableValue, oldValue, newValue) -> mahasiswaFilteredList.setPredicate(createPredicate(newValue))
        );
        presenter.geAllNilaiMahasiswa();

        table.getSelectionModel().selectedItemProperty().addListener(new ChangeListener<Mahasiswa>() {
            @Override
            public void changed(ObservableValue<? extends Mahasiswa> observableValue, Mahasiswa course, Mahasiswa t1) {
                if (observableValue.getValue() != null) {
                    txtNama.setText(observableValue.getValue().getNama());
                    txtNim.setText(observableValue.getValue().getNim());
                    txtNilai.setText("" + observableValue.getValue().getNilai());
                    if (observableValue.getValue().getFoto() != null) {
                        Image image = new Image(new ByteArrayInputStream(observableValue.getValue().getFoto()));
                        imgvFoto.setImage(image);
                    } else {
                        imgvFoto.setImage(null);
                    }
                }
            }
        });

        nim.setCellValueFactory(new PropertyValueFactory<>("Nim"));
        nama.setCellValueFactory(new PropertyValueFactory<>("Nama"));
        nilai.setCellValueFactory(new PropertyValueFactory<>("Nilai"));
    }
```

View > NilaiMhsController (2)

```
private Predicate<Mahasiswa> createPredicate(String searchText) {
    //sama
}

public ObservableList<Mahasiswa> getObservableList() {
    //sama
}

private void bersihkan() {
    //sama
}

private boolean isNilaiMahasiswaUpdated() {
    //sama
}
```

View > NilaiMhsController (3)

```
@FXML
protected void onBtnAddClick() {
    Mahasiswa mahasiswa;
    if (isNilaiMahasiswaUpdated()) {

presenter.updateNilaiMahasiswa(table.getSelectionMode
l().getSelectedItem(),
                new Mahasiswa(txtNim.getText(),
txtNama.getText(),
Double.parseDouble(txtNilai.getText()),
newProfileImageBytes));
    } else {
        mahasiswa = new Mahasiswa(txtNim.getText(),
txtNama.getText(),
Double.parseDouble(txtNilai.getText()),
newProfileImageBytes);
        presenter.addNilaiMahasiswa(mahasiswa);
    }
    bersihkan();
}
```

```
@FXML
protected void onBtnHapusClick() {

presenter.deleteNilaiMahasiswa(table.g
etSelectionModel().getSelectedItem());
}

@FXML
protected void onBtnCloseClick() {
    //sama
}

private void simpan(String fileName)
throws FileNotFoundException {
    //sama
}
```

View > NilaiMhsController (4)

```
@FXML
protected void onBtnSaveFileClick() throws FileNotFoundException {
    //sama
}

@FXML
protected void onOpenBtnClick() throws IOException {
    //sama
}

@FXML
public void handleClearSearchText(Actio
    //sama
}

@FXML
public void onGrafikCLicked() throws IO
    //sama
}

@FXML
public void onPieChartClicked() throws
    //sama
}

@FXML
protected void addImage() throws FileNo
    //sama
}

private boolean searchFindsMahasiswa(Mahasiswa mahasiswa, String searchText) {
    //sama
}
```

```
@Override
public void showError(String errorMessage) {
    Alert alert = new Alert(Alert.AlertType.ERROR, errorMessage);
    alert.show();
    bersihkan();
}

@Override
public void onSuccess(String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION, message);
    alert.show();
    bersihkan();
}
```

Next

- Pelajari dan coba Modul Praktikum
- Unguided Group 2
- UAS
  - Belajarlah JavaFX, Maven, Database
  - Belajarlah desain pattern Singleton
  - Belajarlah menggunakan IntelliJ
  - Belajarlah menggunakan Github