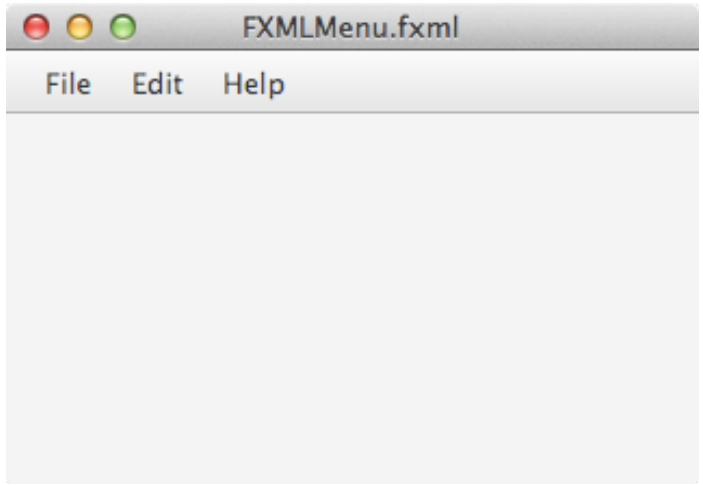
GUI mit JavaFX III

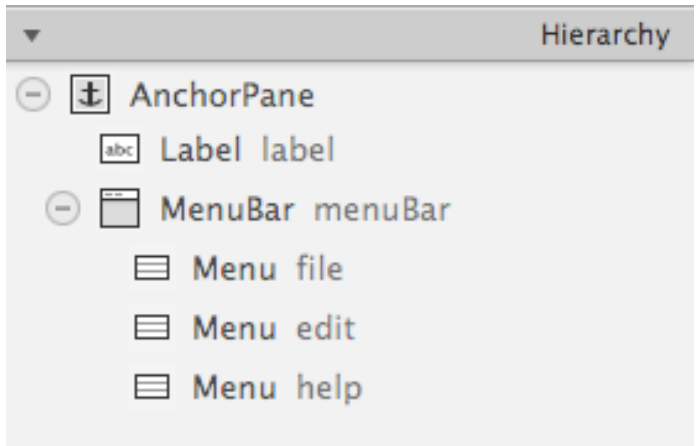
Zentrale Konzepte:

- Menüs
- *FileChooser/DirectoryChooser*
- *DatePicker*
- Mehrdimensionale Sammlungen
  - Mehrdimensionale Arrays
  - Geschachtelte Sammlungen

Menüs

- Ein *MenuBar* kann oben im Fenster eingesetzt werden (am besten im *SceneBuilder*)
- Der *MenuBar* enthält mehrere *Menu*-Objekte (können im *SceneBuilder* hinzugefügt werden)
- Einem Menu können einzelne *MenuItem*-Objekte hinzugefügt werden (im Controller per Programm hinzufügen).
- Einem Menu kann ein Untermenu in Form eines *Menu*-Objekts hinzugefügt werden.
- Es gibt die folgenden *MenuItem* Subklassen:
  - *Menu*
  - *RadioMenuItem*
  - *CheckMenuItem*
  - *SeparatorMenuItem*





Hinzufügen von *MenuItem*-Objekten im Controller

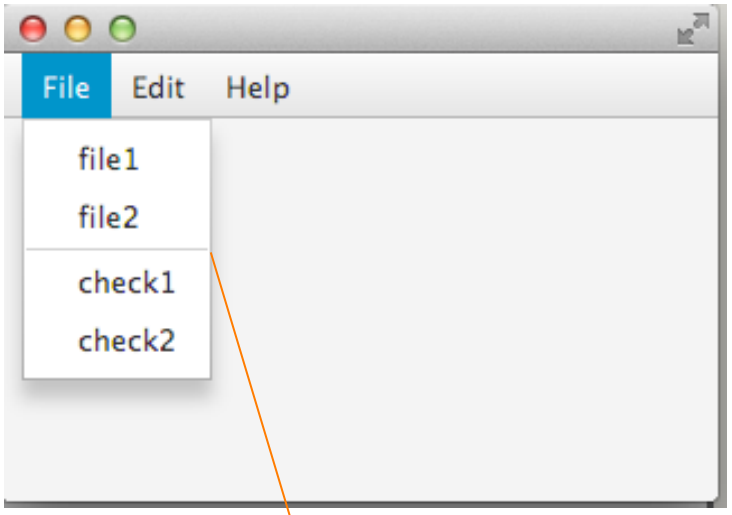
```
public class MenuController implements Initializable {

    @FXML
    private Label label;
    @FXML
    private MenuBar menuBar;
    @FXML
    private Menu file;
    @FXML
    private Menu edit;
    @FXML
    private Menu help;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        initMenu();
    }
```

Attribute mit den Menu-Objekten des MenuBar.

Initialisieren der Menüs in eigener Methode.

Hinzufügen von *MenuItem*-Objekten im Controller

- *MenuItem*-Objekte können anonym der Sammlung der MenuItems eines *Menu*-Objekts hinzugefügt werden.
- Ein *MenuItem* bekommt im Konstruktor einen Bezeichner übergeben
- Ein *SeparatorMenuItem* trennt zwei Menüpunkt-Bereiche voneinander

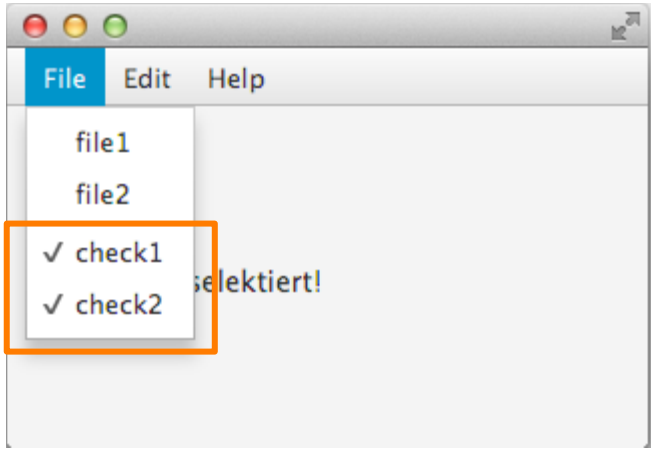



```
private void initMenu() {
    file.getItems().add(new MenuItem("file1"));
    file.getItems().add(new MenuItem("file2"));
    file.getItems().add(new SeparatorMenuItem());
    file.getItems().add(new CheckMenuItem("check1"));
    file.getItems().add(new CheckMenuItem("check2"));

    ...
}
```

CheckMenuItem

- *CheckMenuItem*-Objekte können verwendet werden, um z.B. den Modus einer Operation festzulegen.
- *CheckMenuItem*-Objekte können selektiert oder deselektiert sein.
- Den Status eines *CheckMenuItem*-Objekts kann man mit der Methode *isSelected()* abfragen.

```
private void check(CheckMenuItem it) {
   if (it.isSelected()) {
        label.setText(it.getText() + " ist selektiert!");
   } else {
        label.setText(it.getText() + " ist nicht selektiert!");
   }
}
```



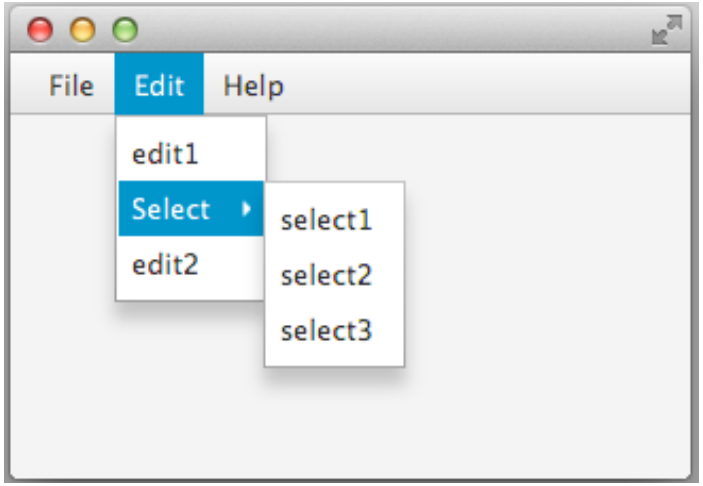
RadioMenuItem

- *RadioMenuItem*-Objekte können selektiert oder deselektiert sein und sehen aus wie *CheckMenuItem*-Objekte.
- Werden die *RadioMenuItem*-Objekte einer *ToggleGroup* hinzugefügt, so kann nur genau ein Item der Gruppe ausgewählt sein. Trifft man eine neue Auswahl, so wird die alte automatisch deselektiert.

```
private void initMenu() {
    ...
    RadioMenuItem it1 = new RadioMenuItem("radio1");
    RadioMenuItem it2 = new RadioMenuItem("radio2");

    ToggleGroup group = new ToggleGroup();
    it1.setToggleGroup(group);
    it2.setToggleGroup(group);

    help.getItems().add(it1);
    help.getItems().add(it2);
    ...
}
```

Untermenüs

- Einem *Menu*-Objekt kann als *MenuItem* auch ein *Menu*-Objekt hinzugefügt werden. So lassen sich geschachtelte Menüs erstellen.




```
private void initMenu() {
    ...
    edit.getItems().add(new MenuItem("edit1"));

    Menu untermenu = new Menu("Select");
    untermenu.getItems().add(new MenuItem("select1"));
    untermenu.getItems().add(new MenuItem("select2"));
    untermenu.getItems().add(new MenuItem("select3"));

    edit.getItems().add(untermenu);
    edit.getItems().add(new MenuItem("edit2"));
    ...
}
```

Event Handling in Menüs

- Einem *MenuItem* kann mit Hilfe der Methode *setOnAction(..)* ein Event Handler zugeordnet werden.
- Soll allen *MenuItem*-Objekten eines Menüs derselbe Event Handler zugewiesen werden, so kann man dies am besten in einer Methode machen.

```
private void initMenu() {
    file.getItems().add(new MenuItem("file1"));
    ...
    file.getItems().add(new CheckMenuItem("check2"));
    setHandler(file, (e) -> handleAll(e));
    ...
}

private void setHandler(Menu menu, EventHandler<ActionEvent> handler){
    for (MenuItem it : menu.getItems()) {
        if(!(it instanceof Menu)) it.setOnAction(handler);
    }
}
```

Lambda Expression für Event Handler.

Setzen des Event Handlers handler für alle MenuItems des Menüs menu.

Event Handling in Menüs

- Will man die unterschiedlichen *MenuItem*-Arten eines Menüs unterschiedlich behandeln, kann man dies in der Event Handling Methode mit Hilfe von *instanceof* realisieren.

```
private void handleAll(ActionEvent e) {
    MenuItem it = (MenuItem) e.getSource();
    Menu parentMenu = it.getParentMenu();
    if (it instanceof CheckMenuItem) {
        check((CheckMenuItem) it);
    } else if (it instanceof RadioMenuItem) {
        label.setText("Neuer Wert der ToggleGroup: " + it.getText());
    } else {
        label.setText(parentMenu.getText() + ": " + it.getText());
    }
}
```

Ein Event liefert das auslösende MenuItem.

Ein MenuItem liefert das zugehörige Menü.

Unterscheidung zwischen MenuItem-Arten.

Event Handling in Menüs

- Will man die unterschiedlichen *MenuItem* Einträge eines Menüs unterschiedlich behandeln, kann man dies in der Event Handling Methode mit Hilfe von *switch* realisieren.

```
private void select(ActionEvent e) {
    MenuItem it = (MenuItem) e.getSource();

    switch(it.getText()) {
        case "select1" : methode1(); break;
        case "select2" : methode2(); break;
        case "select3" : methode3(); break;
    }
}
```

Ein Event liefert das auslösende MenuItem.

Hier wird je nach Aufschrift des gewählten Menüpunktes eine andere Methode aufgerufen.

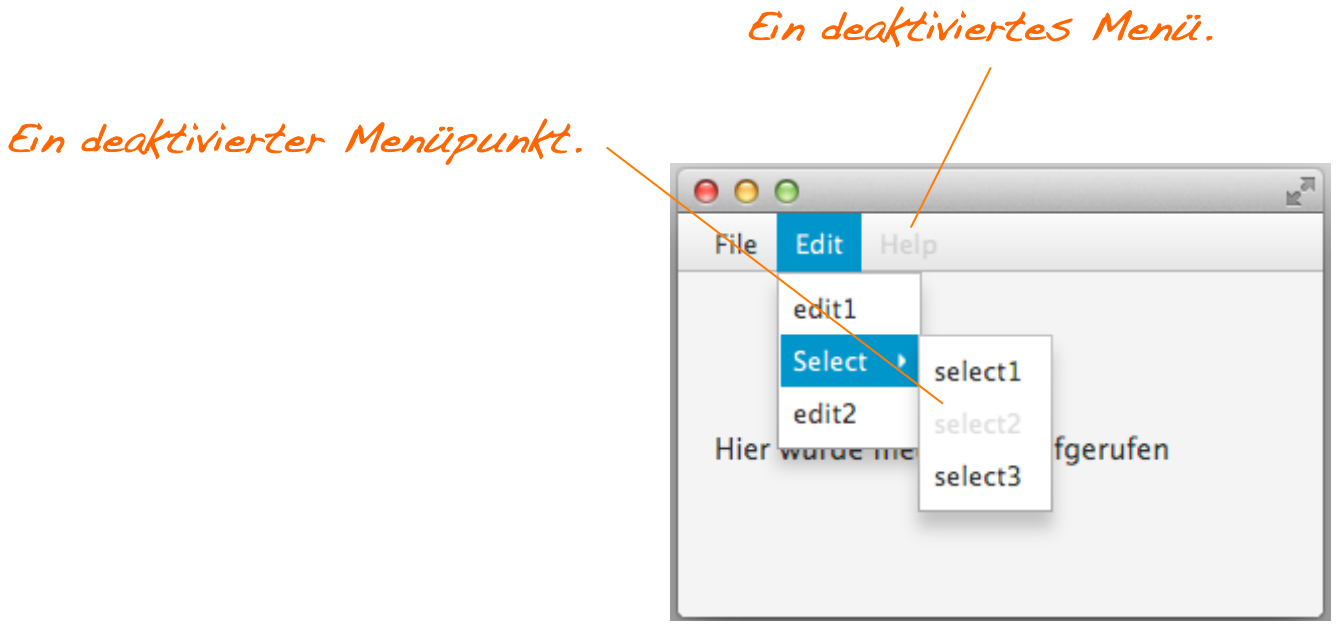
Deaktivieren von Menüs und Menüpunkten

- Man kann ein *MenuItem* oder ein ganzes Menü deaktivieren:

```
public void setDisable(boolean b)
```

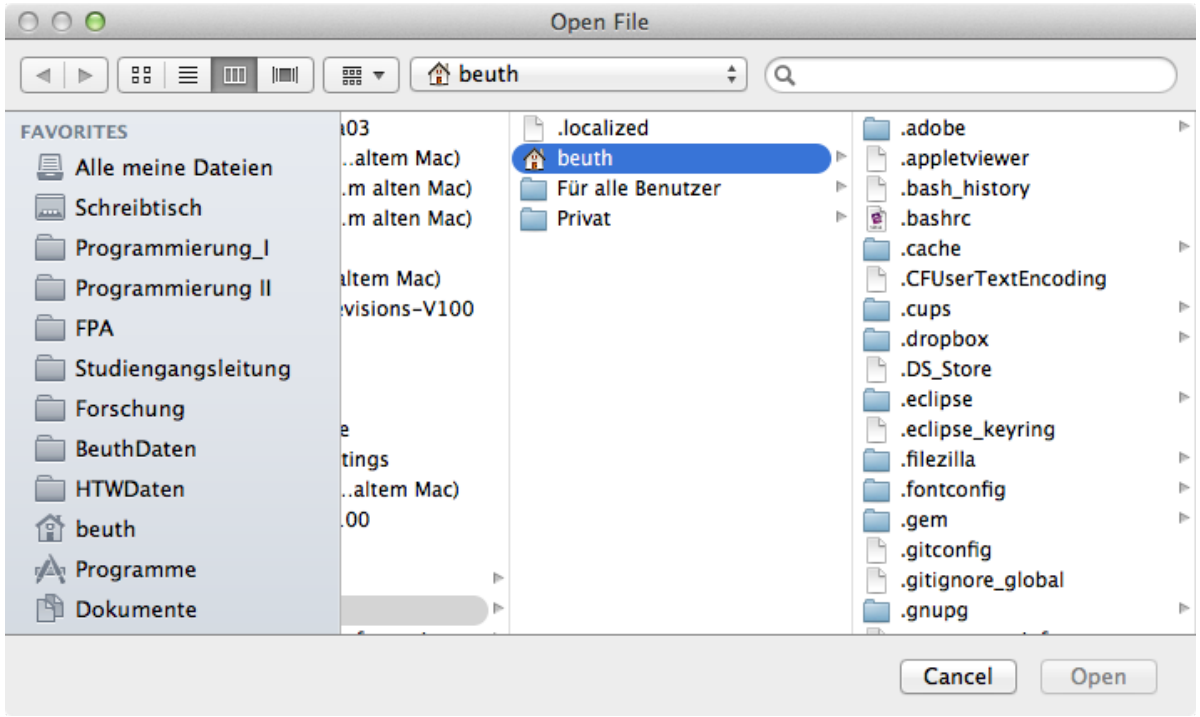
- Man kann ein *MenuItem* oder ein ganzes Menü unsichtbar machen:

```
public void setVisible(boolean b)
```



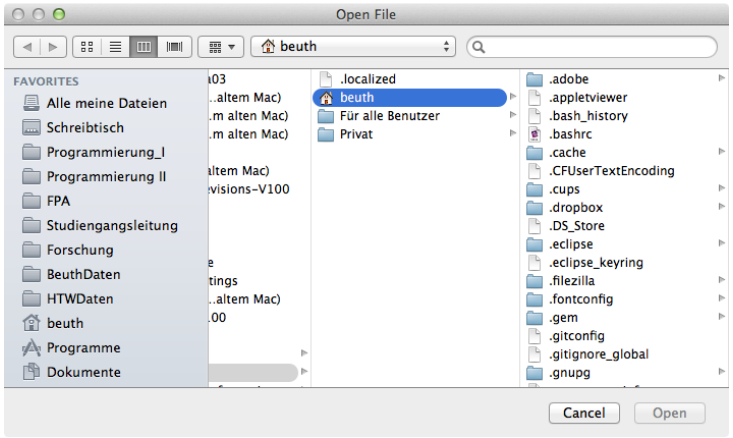
FileChooser

- Für das Auswählen einer Datei gibt es bereits einen vordefinierten Dialog, der sich leicht einsetzen lässt.
- *Der FileChooser lässt sich in verschiedenen Modi anzeigen: zum Öffnen oder zum Speichern einer Datei*



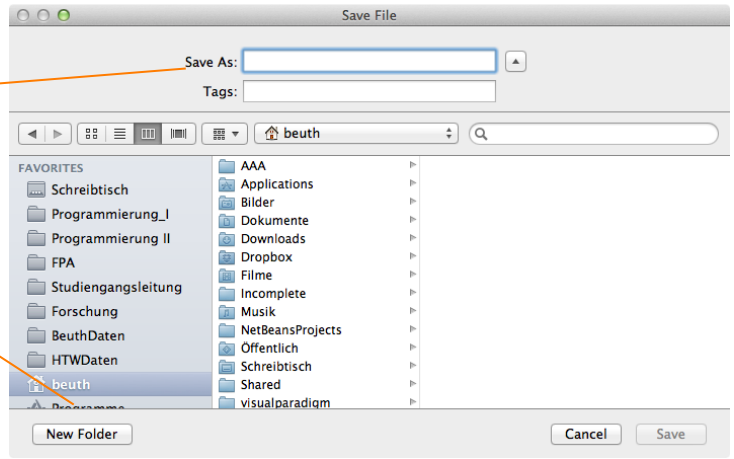
FileChooser: Open und Save Dialog

```
FileChooser chooser = new FileChooser();
chooser.setTitle("Open File");
File selection = chooser.showOpenDialog(null);
```



Eingabefeld für Namen der Datei.

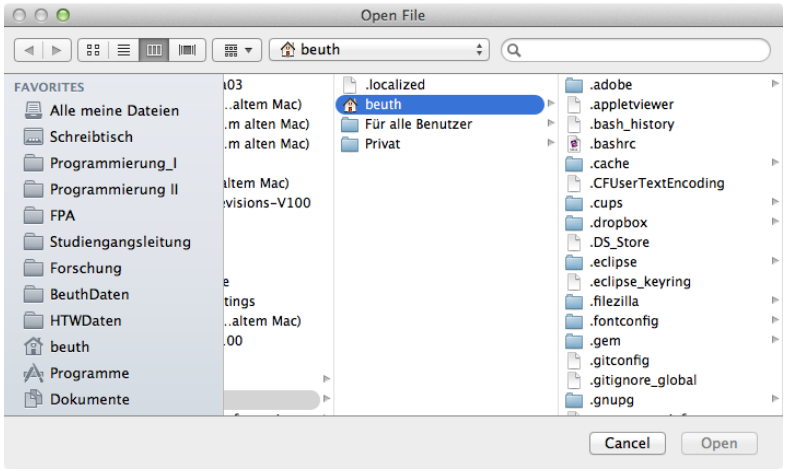
Button für neues Verzeichnis.




```
FileChooser chooser = new FileChooser();
chooser.setTitle("Save File");
File selection = chooser.showSaveDialog(null);
```

FileChooser: Einstellen des Start-Verzeichnis

- Man kann einstellen, welches Verzeichnis der *FileChooser* initial anzeigen soll.



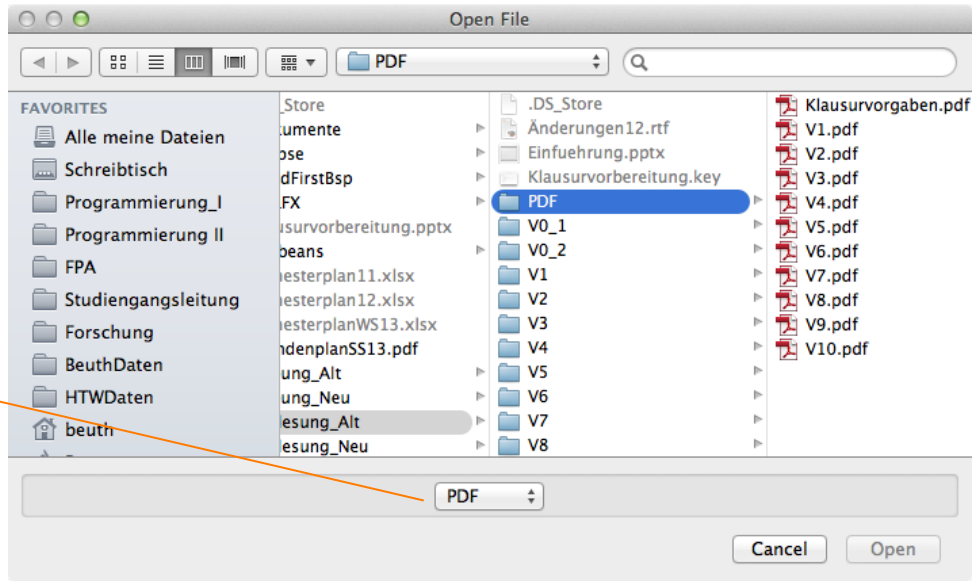

```
FileChooser chooser = new FileChooser();

chooser.setTitle("Open File");

File dir = new File(System.getProperty("user.home"));

chooser.setInitialDirectory(dir);

File selection = chooser.showOpenDialog(null);
```

z.B. Plattformunabhängige Angabe des User-Verzeichnis.

Setzen des Start-Verzeichnis.

FileChooser: Einstellen von Filtern für Dateiendungen

- Man kann über einen *ExtensionFilter* einstellen, welche Dateien der *FileChooser* anbieten soll.

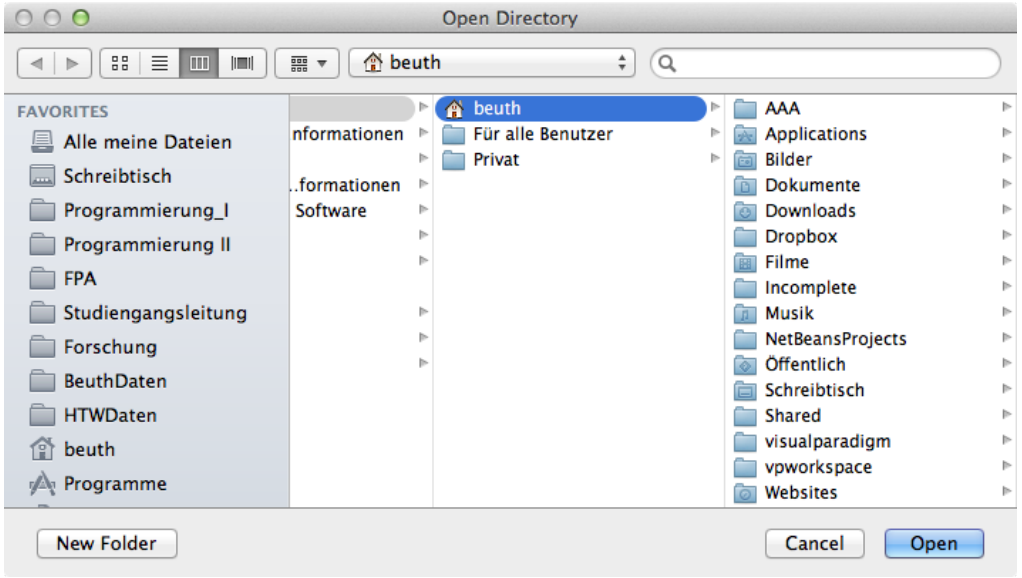


Auswahl für die gesetzten Filter.

```
chooser chooser = new FileChooser();
...
FileChooser.ExtensionFilter f = new FileChooser.ExtensionFilter("PDF", "*.pdf");
chooser.getExtensionFilters().add(f);
File selection = chooser.showOpenDialog(null);
```

DirectoryChooser: Auswahl von Verzeichnissen

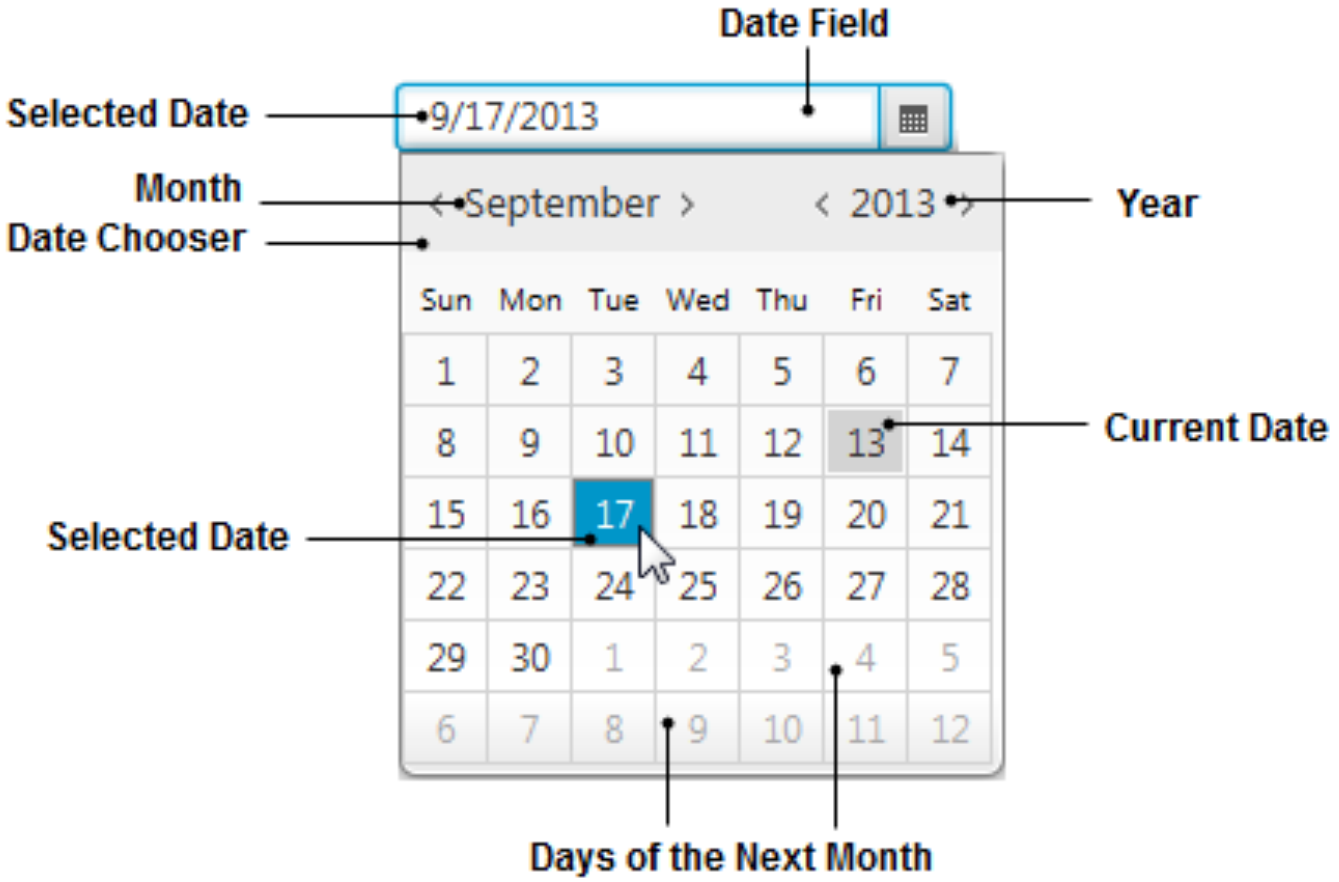
- Soll nur ein Verzeichnis ausgewählt werden können, so kann man den *DirectoryChooser* verwenden.




```
DirectoryChooser dirChooser = new DirectoryChooser();

dirChooser.setTitle("Open Directory");

File startDir = new File(System.getProperty("user.home"));

dirChooser.setInitialDirectory(startDir);

File selection = dirChooser.showDialog(null);
```

DatePicker

- ⊙ Für das Auswählen eines Datums gibt es bereits einen vordefinierten Dialog, der sich leicht einsetzen lässt.
- ⊙ Arbeitet mit der neuen Java Date-Time-API.



LocalDate

- Repräsentiert ein Datum mit Tag/Monat/Jahr Informationen.
- Zugriff auf das aktuelle Datum

```
LocalDate date = LocalDate.now();
```

- Zugriff auf ein bestimmtes Datum (jahr/monat/tag)

```
LocalDate date = LocalDate.of(2000, 11, 20);
```

- Addieren einer Zeitspanne

```
LocalDate morgen = LocalDate.now().plusDays(1);
LocalDate inEinemMonat = LocalDate().now().plusMonths(1);
LocalDate inEinemJahr = LocalDate().now().plusYears(1);
```

LocalTime

- Repräsentiert eine Uhrzeit mit Stunde/Minute/Sekunde Informationen.
- Zugriff auf das aktuelle Datum

```
LocalTime time = LocalTime.now();
```

- Zugriff auf eine bestimmte Uhrzeit (std/min bzw. std/min/sek)

```
LocalTime time = LocalTime.of(13, 45);
LocalTime time = LocalTime.of(13, 45, 20);
```

- Addieren einer Zeitspanne

```
LocalDate inEinerStunde = LocalTime.now().plusHours(1);
LocalDate inEinerMinute = LocalTime.now().plusMinutes(1);
LocalDate inEinerSekunde = LocalTime.now().plusSeconds(1);
```

Formatierte Ein-/Ausgabe

- Mit Hilfe der Klasse *DateTimeFormatter* lässt sich ein Datum formatiert ausgeben oder aus einem String parsen in ein *LocalDate* oder *LocalTime* Objekt.
- Bsp. Formatierte Ausgabe *LocalDate*

```
LocalDate date = LocalDate.now();

DateTimeFormatter f1 = DateTimeFormatter.ofPattern("d MMM yy");

DateTimeFormatter f2 = DateTimeFormatter.ofPattern("EEEE dd MMMM yyyy");

System.out.println(date.format(f1));        3 Nov 14

System.out.println(date.format(f2));        Donnerstag 03 November 2014
```

- Bsp. Parsen in *LocalTime*

```
DateTimeFormatter f1 = DateTimeFormatter.ofPattern("HH:mm");

String time = "23:30";

LocalTime time = LocalTime.parse(time, format1);
```

DatePicker

- Einrichten mit Startdatum

```
datePicker.setValue(LocalDate.now());
```

- Einrichten ohne Kalenderwochen

```
datePicker.setShowWeekNumbers(false);
```

- Anmelden eines Event Handlers, der reagiert wenn ein Datum ausgwählt wurde

```
picker.setOnAction((e) -> handleInputDate());
```

Mehrdimensionale Sammlungen

Viele Informationen sind so strukturiert, dass man sie in mehrdimensionalen Sammlungen verwalten muss:

- Tabellen
  - Schachbrett
  - Notenlisten
  - Stundenplan
- Terminkalender: Sammlung mit Zugriff über ein Datum auf eine Liste von Terminen
- Musik-Sammlung: Sammlung von Alben eines Künstlers abgespeichert nach dem Namen des Künstlers und Namen des Albums

Zweidimensionale Arrays

Tabellenartige Datenstrukturen sind in vielen Bereichen des täglichen Lebens anzutreffen, z.B.:

- Schachbrett
- Notenlisten
- Stundeplan

Deklarieren und Initialisieren eines zweidimensionalen Arrays


```
int[][] verkaeufeWoche = new int[4][7]; // 4 Läden, 7 Tage

String[][] schachbrett = new String[8][8]; // 8 Zeilen, 8 Spalten
```

Zugriff auf zweidimensionale Arrays

```
int[][] verkaeufeWoche = new int[4][7]; // 4 Läden, 7 Tage
```

Verkaufszahl im 3. Laden am Dienstag

```
int verkaeufeDienstagLaden3 = verkaeufeWoche[2][1];
```

Durchschnittlicher Wochenverkauf im 2.Laden

```
int verkauf = 0;
int anzahlTage = verkaeufeWoche[1].length;
for(int tag = 0; tag < anzahlTage; tag++)
        verkauf = verkauf + verkaeufeWoche[1][tag];
verkauf = verkauf/anzahlTage;
```

Länge der Zeile

2. Laden (konstant)

Alle Verkaufszahlen aus dem 4. Laden

```
int[] verkaeufeLaden4 = verkaeufeWoche[3];
```

Liefert die 4. Zeile, d.h. ein eindimensionales Array

Zugriff auf zweidimensionale Arrays

```
int[][] verkaeufeWoche = new int[4][7]; // 4 Läden, 7 Tage
```

Geschachtelte For-Schleife: Alle Verkäufe aller Läden an allen Tagen summieren

```
int summe = 0;

for(int laden = 0; laden < verkaeufeWoche.length; laden++)

  for(int tag = 0; tag < verkaeufeWoche[laden].length; tag++)

            summe = summe + verkaeufeWoche[laden][tag];
```

Anzahl der Zeilen: `verkaeufeWoche.length`

Anzahl der Spalten: `verkaeufeWoche[i].length`

Angabe der Zeile, deren Spaltenanzahl man auslesen möchte. Bei regelmäßigen Arrays kann dies eine feste Zahl sein, z.B. 0

Geschachtelte Sammlungen

- Terminkalender: Sammlung mit Zugriff über ein Datum auf eine Liste von Terminen

```
HashMap<LocalDate, ArrayList<Termin>>
```

Schlüssel: Datum

Wert: Liste mit Termin-Objekten

- Musik-Sammlung: Sammlung von Alben eines Künstlers abgespeichert nach dem Namen des Künstlers und Namen des Albums

Schlüssel: Name des Albums

Wert: Album-Objekt

```
HashMap<String, HashMap<String, Album>>
```

Schlüssel: Name des Künstlers

Wert: HashMap mit Alben des Künstlers

Bsp: Projekt Terminkalender

```
public class TerminVerwaltung {

   private TreeMap<LocalDate, ArrayList<Termin>> termineDate;



   public TerminVerwaltung() {
      initialisieren();
   }

   protected void initialisieren() {

        termineDate = new TreeMap<LocalDate, ArrayList<Termin>>();

   }
```

Zugriff auf alle Termine eines Tages

```
public List<Termin> getTermineTag(LocalDate date) {
    ArrayList<Termin> liste = termineDate.get(date);
    return liste;
}
```

Holt die Liste zu dem Datum

Entfernen eines Termins

```
public void removeTermin(Termin t) {
    ArrayList<Termin> liste = termineDate.get(t.getDatum());
    liste.remove(t);
    if (liste.isEmpty()) {
        termineDate.remove(t.getDatum());
    }
}
```

Entfernt den Termin aus der Liste des zug. Tages

Entfernt die leere Liste aus der HashMap

Hinzufügen eines Termins

```
public void addTermin(Termin t) throws TerminUeberschneidungException {
   Termin alt = checkTerminUeberschneidung(t);
   if (alt == null) addTerminDate(t);
        else throw new TerminUeberschneidungException(alt);
}

private void addTerminDate(Termin t) {
   ArrayList termine = termineDate.get(t.getDatum());
   if (termine == null) {
        termine = new ArrayList<Termin>();
        termineDate.put(t.getDatum(), termine);
   }
   termine.add(t);
}
```

Holt die Liste zu dem Datum

Wenn die Liste leer ist, muss eine neue erstellt werden.

Fügt die Liste für das Datum in die HashMap ein.

Fügt der Liste den Termin hinzu.