GUI Application (3)

Han-fen Hu

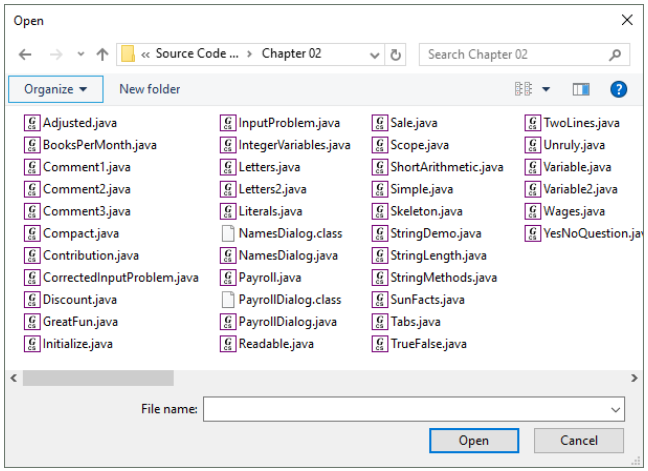
UNLV

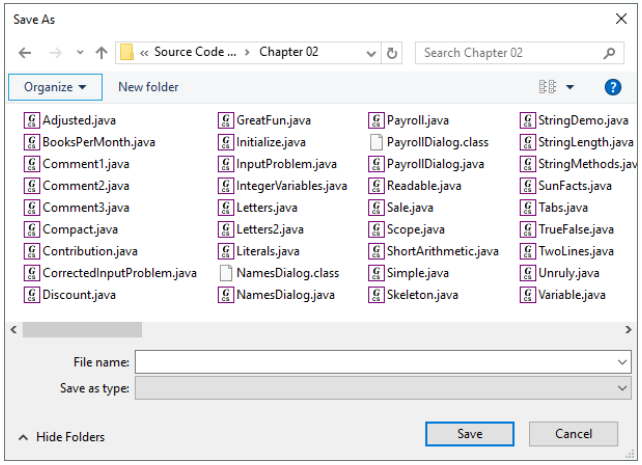
Outline

- Other Controls in JavaFX: FileChooser
- Default Button and Mnemonic
- Multiple Scene Application
- TableView Control
- JavaDoc

FileChooser (1)

- The `FileChooser` class displays a dialog box that allows the user to browse for a file and select it.
- The class can display two types of predefined dialog boxes:
  - open dialog box
  - save dialog box





FileChooser (2)

- First, create an instance of the `FileChooser` class
- Then, call either the `showOpenDialog` method, or the `showSaveDialog` method
- With either method, you pass a reference to the application's stage as an argument

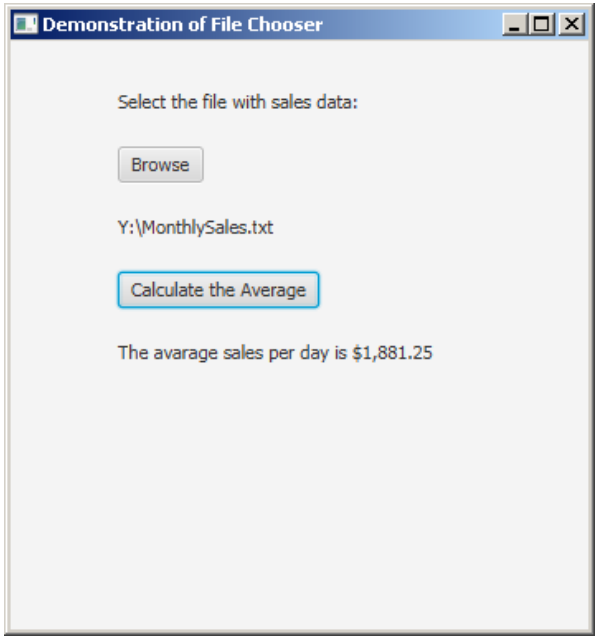
```
FileChooser fileChooser = new FileChooser();
File selectedFile = fileChooser.showOpenDialog(primaryStage);
```

FileChooser (3)

- The `showOpenDialog` and `showSaveDialog` methods return a File object (in the java.io package) containing information about the selected file.
- If the user does not select a file, the method returns null.

```
FileChooser fileChooser = new FileChooser();
File selectedFile = fileChooser.showOpenDialog(primaryStage);
if (selectedFile != null)
{
   String filename = selectedFile.getPath();
   outputLabel.setText("You selected " + filename);
}
```

Lab (1)



- ReadFile.fxml
- ReadFile.java
- ReadFileController.java
  - In this application, the user will select a file with sales data and the program calculates the average sales

UNLV

Default Button

- Default Button
  - A default Button is the button that receives a keyboard VK_ENTER press, if no other node in the scene consumes it
  - Allows the user to use keyboard to trigger the event at a button.

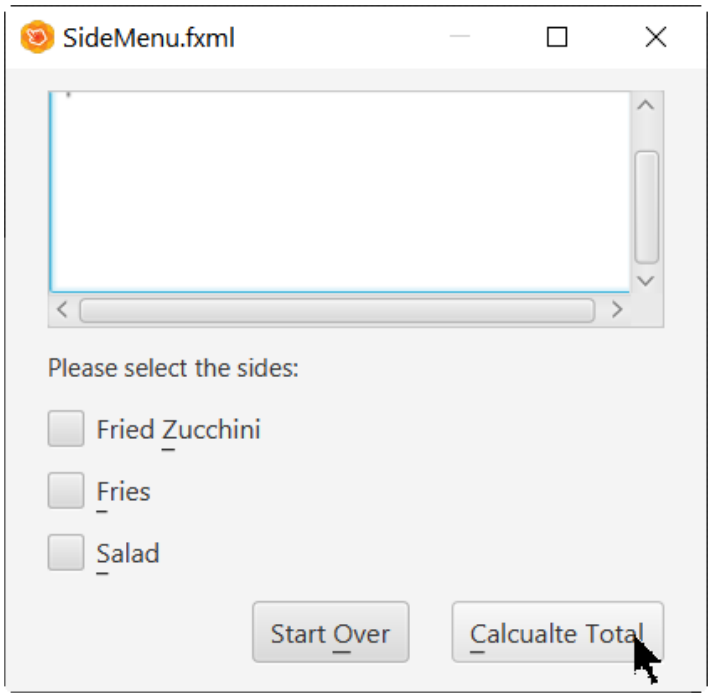
```
<Button text="Login" defaultButton="true"
                  onAction="#handleSubmitButtonAction"/>
```

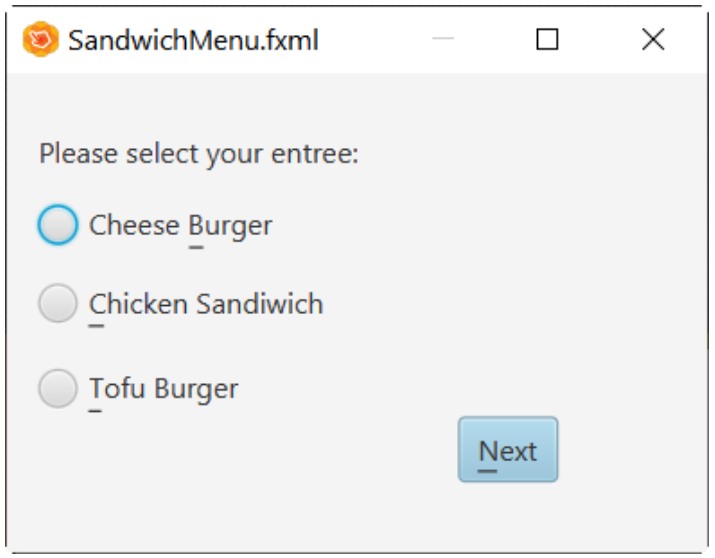
Mnemonic

- Mnemonic registers a keyboard shortcut to activate the element
  - using the letter following _ in the text + Alt
- mnemnonicParsing
  - Parse the _ in the Text as the registration of mnemnonic
  - if `mnemnonicParsing` is false. In this case the _ will also be printed normally instead of underlining the following letter.

Lab (2)

- SandwichMenu.fxml
- SideMenu.fxml





Multiple Scene Application (1)

- Create an FXML and a controller for each scene
- Keep the scene as simple as possible
  - Push different scenes to the stage, one at a time
- Variables can be passed between scenes
  - Create a method in the second scene that accepts parameters
  - From the first scene, call the method and pass the arguments

Multiple Scene Application (2)

- Calling a method in a controller class
  - Via the loader of the FXML
  - The controller class is specified in the FXML file
  - Call the method and pass the variable

```
// the FXML loader object to load the UI design
FXMLLoader loader = new FXMLLoader();
// specify the file location
loader.setLocation(getClass().getResource("SideMenu.fxml"));
// access the controller class via the loader
// in the FXML file, the controller class is specified.
SideMenuController controller = loader.getController();
// call the method in the controller class.
// Pass the string to the next scene
controller.initData(item);
```

Multiple Scene Application (3)

- Push a new scene to the window
  - A window is a stage
  - To get the current stage, we need to use the event object to get the stage

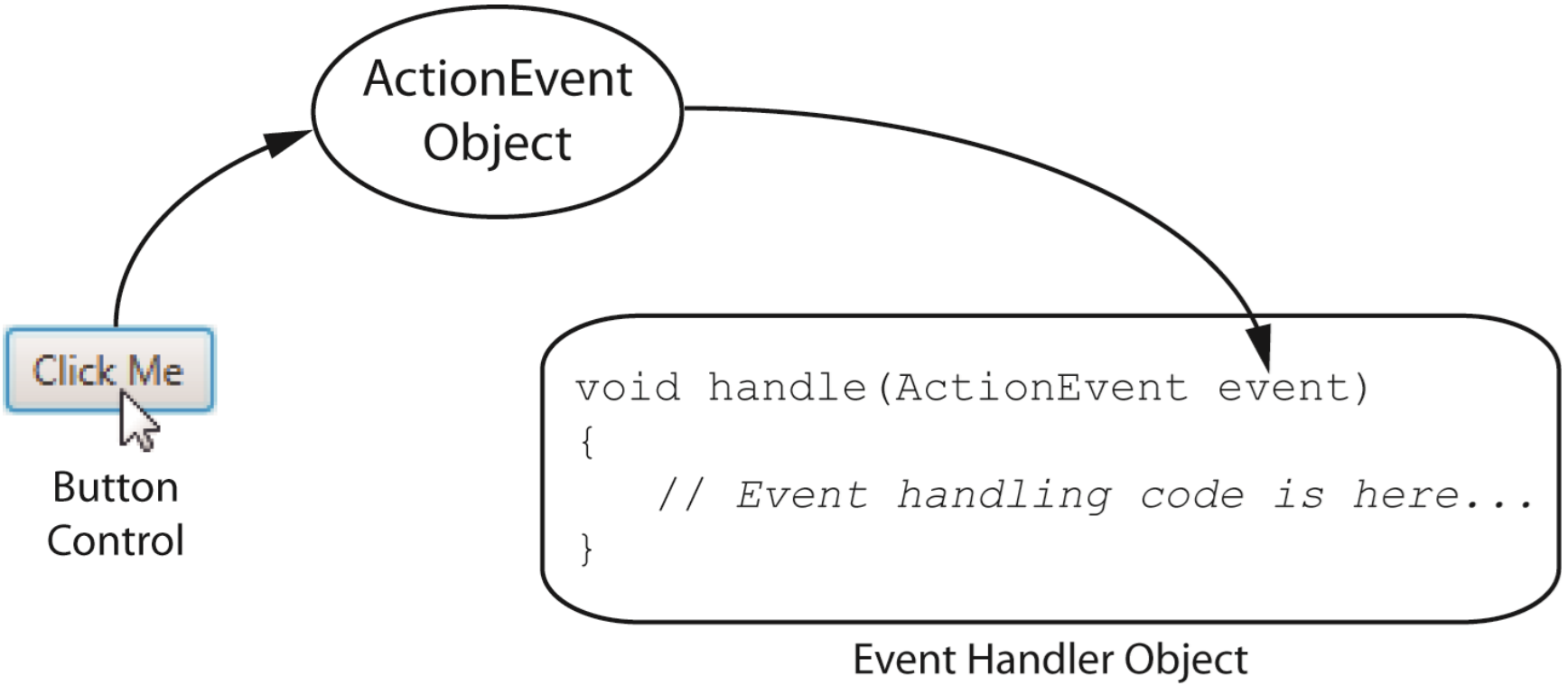
```
Stage stage =
(Stage)((Node)e.getSource()).getScene().getWindow();
```

    - Every event on the GUI will create an event object and pass to the listener method

- Set the scene for the stage

```
stage.setScene(scene);
```

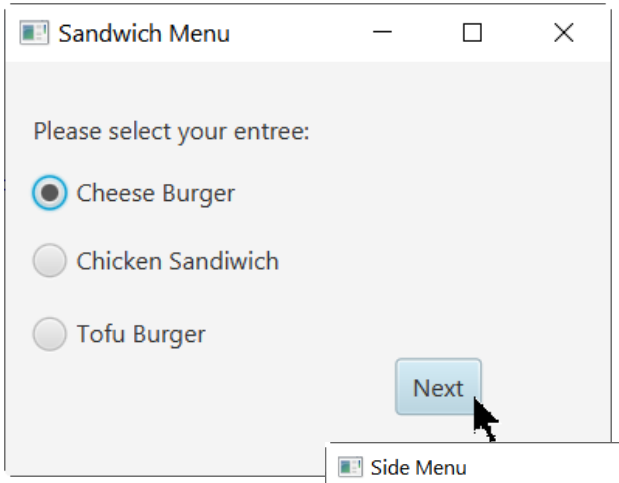
ActionEvent Object

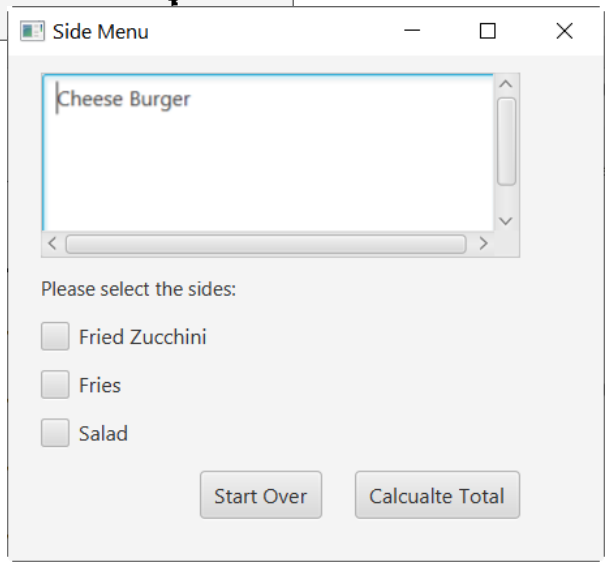


Example of ActionEvent Object

```
public void changeScenceToSideMenu(ActionEvent e) throws Exception {
    // preprare the string to be sent to the next page
    String item="";
    // check which radio button is selected. Then set the string according
    if(cheeseBurgerRadioButton.isSelected())
        item = "Cheese Burger";
    else if(chickenSandwichRadioButton.isSelected())
        item = "Chicken Sandwich";
    else if(tofuBurgerRadioButton.isSelected())
        item = "Tofu Burger";

    // the FXML loader object to load the UI design
    FXMLLoader loader = new FXMLLoader();
    // specify the file location
    loader.setLocation(getClass().getResource("SideMenu.fxml"));
    // access the controller class via the loader
    // in the FXML file, the controller class is specified.
    SideMenuController controller = loader.getController();
    // call the method in the controller class.
    // Pass the string to the next scene
    controller.initData(item);


    // load the UI
    Parent parent = loader.load();
    // set the scene
    Scene scene = new Scene(parent);

    // get the current window
    Stage stage = (Stage)((Node)e.getSource()).getScene().getWindow();
```

Lab (3)





- SandwichMenu.fxml
- SideMenu.fxml
- SandwichMenuController.java
- SideMenuController.java
- BurgerJoint.java
  - In this application, the user will select a sandwich from the first window and select the sides from the second window. The selected sandwich and sides will be display at the TextArea of the second window

UNLV

TableView Control (1)

- The most important classes for creating tables are `TableView`, `TableColumn`, and `TableCell`.
  - Populate a table by implementing the data model and by applying a cell factory.
- `TableColumn` object need to be created first and then added to `TableView`

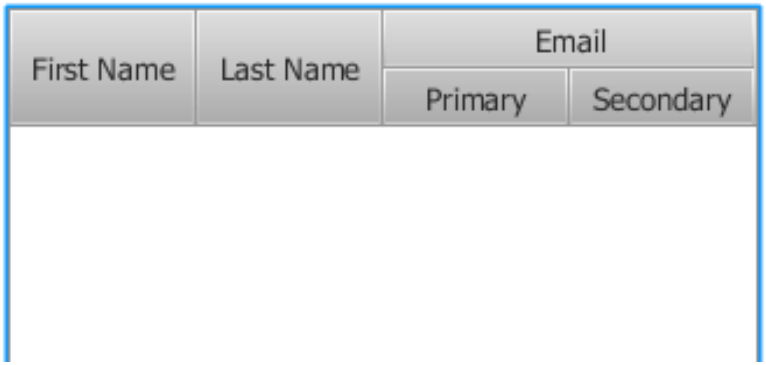
```
TableView table = new TableView();

TableColumn firstNameCol = new TableColumn("First Name");
TableColumn lastNameCol = new TableColumn("Last Name");
TableColumn emailCol = new TableColumn("Email");

Table.getColumns().addAll(firstNameCol, lastNameCol,
emailCol);
```

UNLV

TableView Control (2)

- Table with Nested Columns

```
TableColumn firstNameCol = new TableColumn("First Name");
TableColumn lastNameCol = new TableColumn("Last Name");
TableColumn emailCol = new TableColumn("Email");

TableColumn firstEmailCol = new TableColumn("Primary");
TableColumn secondEmailCol = new TableColumn("Secondary");

emailCol.getColumns().addAll(firstEmailCol, secondEmailCol);
```

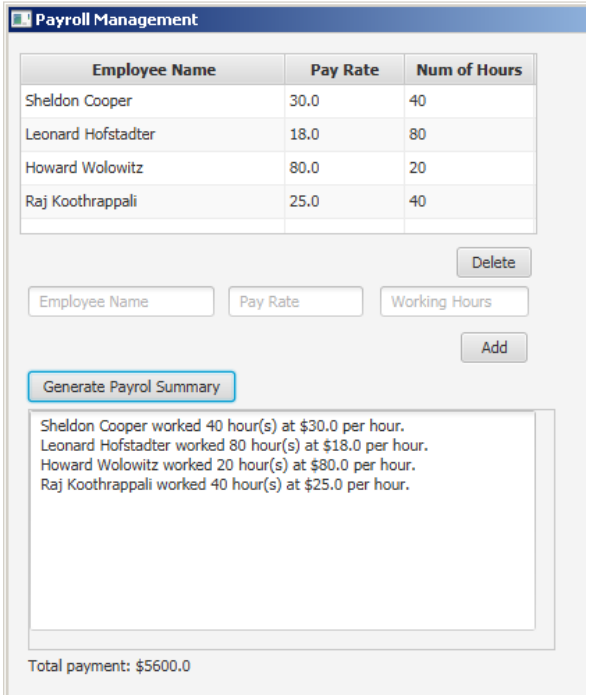


TableView Control (3)

- Defining the Data Model
  - It is a best practice to implement a class that defines the data model as an object class
  - The class provides methods and fields to further work with the table
- Then associate the data with the table columns

```
firstNameCol.setCellValueFactory( new
PropertyValueFactory<Person,String>("firstName") );
lastNameCol.setCellValueFactory( new
PropertyValueFactory<Person,String>("lastName") );
emailCol.setCellValueFactory( new
PropertyValueFactory<Person,String>("email") );
```

Lab (4)



- Payroll.java
- PayrollManagement.java
  - In this example, we will create an application that manages the payroll for an organization.
- PayrollManagementController.java
- PayrollManagement.fxml

TableView Control (4)

- Once the data model is outlined in an object class, an `ObservableList` array can be defined to represent rows to be show in the table
  - An `ObservableList` is similar to an ArrayList but is used in GUI applications.
- To add a row, we can add an object to the end of the `ObservableList` array which representing the existing rows
  - The getItems() method of `TableView` returns an `ObservableList`

```
// create a Payroll object
Payroll payroll = new Payroll();
// set the values
payroll.setEmpName(nameTextField.getText());
payroll.setPayRate(rateTextField.getText());
payroll.setNumOfHours(hoursTextField.getText());

// get all the items from the table as a list, then add the new object to it
// add it to the table
tableView.getItems().add(payroll);
```


UNLV

Lab (5)

- PayrollManagementController.java
- PayrollManagement.fxml

TableView Control (5)

- To remove a row from TableView, we can use the **getSelectedIndex()** method to retrieve the selected item
- `ObservableList` class provides the remove() method that accepts an index for removing a record.

Lab (6)

- PayrollManagementController.java
- PayrollManagement.fxml

UNLV

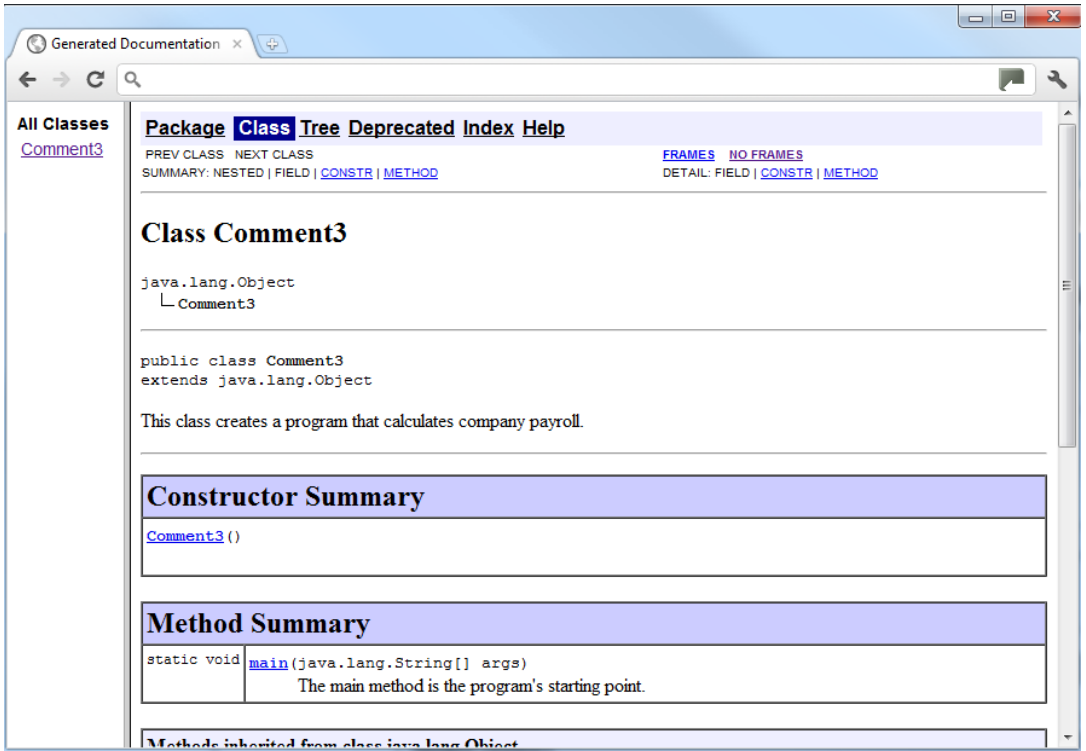
Javadoc (1)

- Java provides three methods for commenting code.

Comment Style	Description
//	Single line comment. Anything after the // on the line will be ignored by the compiler.
/* … */	Block comment. Everything beginning with /* and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.
/** … */	Javadoc comment. This is a special version of the previous block comment that allows comments to be documented by the javadoc utility program. Everything beginning with the /** and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.

Javadoc (2)

- Javadoc comments can be built into HTML documentation.
- The **`javadoc`** program will create **`index.html`** and several other documentation files in the same directory as the input file.



`@param` Tag in Documentation Comments

- You can provide a description of each parameter in your documentation comments by using the `@param` tag.
- General format

```
@param parameterName Description
```

- All `@param` tags in a method's documentation comment must appear after the general description. The description can span several lines.

`@return` Tag in Documentation Comments

- You can provide a description of the return value in your documentation comments by using the **`@return`** tag.
- General format

```
@return Description
```

- The **`@return`** tag in a method's documentation comment must appear after the general description. The description can span several lines.

Other Tags in Documentation Comments

- @author
- @version

Lab (7)

- SavingsAcocunt.java
- RetailItem.java