Contents

Chapter 1: controller/Controller.java

```java
package controller;

import model.Pricelist;
import model.ProductGroup;
import model.Rent;
import model.Sale;
import model.enums.PaymentMethod;
import model.enums.SaleState;
import model.enums.Unit;
import model.product.Bundle;
import model.product.Container;
import model.product.Product;
import model.product.Tour;
import model.product.rentable.Rentable;
import storage.Storage;
import java.io.*;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class Controller {

    private static Controller controller;
    private static Storage storage;

    /**
     * Creates and returns a new productGroup
     *
     * @param name the name of the productGroup
     * @return created productGroup
     */
    public ProductGroup createProductGroup(String name) {
        ProductGroup productGroup = new ProductGroup(name);
        storage.addProductGroup(productGroup);
        return productGroup;
    }

    /**
     * Creates and returns a new product
     *
     * @param productGroup the productGroup of the product
     * @param name         the name of the product
     * @param description  the description of the product
     * @return created product
     */
    public Product createProductInProductGroup(ProductGroup productGroup, String name,
       ↪ String description) {
        Product product = productGroup.createProduct(name, description);
        storage.addProduct(product);
        return product;
    }

    /**
     * Creates and returns a new container
     *
     * @param productGroup the productGroup of the product
     * @param name         the name of the container
     * @param description  the description of the container
     * @return created container
     */
    public Container createContainerProductInProductGroup(ProductGroup productGroup, String
      ↪ name, String description, double size, Unit unit) {
        Container container = productGroup.createContainerProduct(name, description, size,
          ↪ unit);
        storage.addProduct(container);
        return container;
    }

    /**
     * Creates and returns a new tour
     *
     * @param productGroup the productGroup of the tour
     * @param name         the name of the tour
     * @param description  the description of the tour
     * @return created tour
     */
    public Tour createTourProductInProductGroup(ProductGroup productGroup, String name,
      ↪ String description) {
        Tour tour = productGroup.createTourProduct(name, description);
        storage.addProduct(tour);
        return tour;
    }

    /**
     * Creates and returns a new bundle
     *
     * @param productGroup the productGroup of the bundle
     * @param name         the name of the bundle
     * @param description  the description of the bundle
     * @return created bundle
     */
    public Bundle createBundleProductInProductGroup(ProductGroup productGroup, String name,
      ↪ String description) {
        Bundle bundle = productGroup.createBundleProduct(name, description);
        storage.addProduct(bundle);
        return bundle;
    }

    /**
     * Creates and returns a new pricelist
     *
     * @param name the name of the pricelist
     * @return created pricelist
     */
    public Pricelist createPricelist(String name) {
        Pricelist pricelist = new Pricelist(name);
        storage.addPriceList(pricelist);
        return pricelist;
    }

    /**
     * Adds a product with price to pricelist
     *
     * @param pricelist the pricelist to be added to
     * @param product   the product to be added
     * @param price     the price of the product to be added
     */
    public void addProductWithPriceToPricelist(Pricelist pricelist, Product product, double
       ↪ price) {
        pricelist.addProductWithPrice(product, price);
    }

    /**
     * Creates and returns a new sale
     *
     * @return created sale
     */
    public Sale createSale() {
        Sale sale = new Sale();
        storage.addSale(sale);
        return sale;
    }

    /**
     * Creates and returns a new bundle
     *
     * @param contactName          the contact name of the rent
     * @param contactInformation   the contact information of the rent
     * @param deliveryDateAndTime the delivery date and time of the rent
     * @param returnDateAndTime   the return date and time of the rent
     * @return created rent
     */
    public Rent createRent(String contactName, String contactInformation, LocalDateTime
       ↪ deliveryDateAndTime,
                           LocalDateTime returnDateAndTime) {
        Rent rent = new Rent(contactName, contactInformation, deliveryDateAndTime,
           ↪ returnDateAndTime);
        storage.addSale(rent);
        return rent;
    }

    //
       ↪ ------------------------------------------------------------------------------------
       ↪

    /**
     * Get list of product groups from storage
     *
     * @return list of product groups
     */
    public List<ProductGroup> getProductGroupList() {
        return storage.getProductGroupList();
    }

    /**
     * Get list of products from storage
     *
     * @return list of products
     */
    public List<Product> getProductList() {
        return storage.getProductList();
    }

    /**
     * Get list of pricelists from storage
     *
     * @return list of pricelist
     */
    public List<Pricelist> getPricelists() {
        return storage.getPricelists();
    }

    /**
     * Get list of sales from storage
     *
     * @return list of sales
     */
    public List<Sale> getSales() {
        return storage.getSales();
    }

    /**
     * Get list of sales from storage that is instance of Rent,
     * but does not contain product with instance of Tour in salesLines
     *
     * @return list of sales
     */
    public List<Sale> getRents() {
        return getSales().stream()
                .filter(sale -> sale instanceof Rent)
                .filter(sale -> sale.getSalesLines().stream().noneMatch(salesLine -> !(
                   ↪ salesLine.getProduct() instanceof Tour)))
                .collect(Collectors.toList());
    }

    /**
     * Get list of sales from storage that is instance of Rent,
     * and contains a salesLine with a product instance of Tour
     *
     * @return list of sales
     */
    public List<Sale> getTours() {
        return getSales().stream()
                .filter(sale -> sale instanceof Rent)
                .filter(sale -> sale.getSalesLines().stream().anyMatch(salesLine ->
                   ↪ salesLine.getProduct() instanceof Tour))
                .collect(Collectors.toList());
    }
    //
       ↪ ------------------------------------------------------------------------------------
       ↪

    /**
     * Lazy loaded singleton controller
     *
     * @return controller
     */
    public static Controller getInstance() {
        if (controller == null)
            controller = new Controller();
        return controller;
    }

    // ---------------------------------------------------------------------------------------

    public static final String PATH = "data/Serilizabledata.ser";

    public void saveToFile(String path) {
        try (FileOutputStream fileOutputStream = new FileOutputStream(path);
             ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream)) {

            objectOutputStream.writeObject(storage);
            System.out.println("Saved to file: " + PATH);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void readFromFile(String path) {
        try (FileInputStream fileInputStream = new FileInputStream(path);
             ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream)) {

            Object object = objectInputStream.readObject();
            if (object instanceof Storage)
                storage = (Storage) object;
            System.out.println("Read from file: " + PATH);

        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Could not read file, recreating data from initStorage");
            controller.initStorage();
        }
    }

    // ---------------------------------------------------------------------------------------

    /*
     For testing purposes, dont need to generate date
     */
    public void initStorageOnly() {
        storage = Storage.getInstance();
    }

    public void initStorage() {
        storage = Storage.getInstance();

        Pricelist pricelist1 = createPricelist("Fredagsbar");
        Pricelist pricelist2 = createPricelist("Butik");
        Pricelist pricelist3 = createPricelist("Rundvisninger");

        // Klippekort
        ProductGroup productGroup1 = createProductGroup("Klippekort");
        Product product1 = createProductInProductGroup(productGroup1, "Klippekort, 4 klip",
            ↪ "");

        addProductWithPriceToPricelist(pricelist1, product1, 130);
        addProductWithPriceToPricelist(pricelist2, product1, 130);

        // Flaske
        ProductGroup productGroup2 = createProductGroup("Flaske");
        Product product2 = createContainerProductInProductGroup(productGroup2, "Klosterbryg"
            ↪ , "", 60, Unit.cl);
        Product product3 = createContainerProductInProductGroup(productGroup2, "Sweet
            ↪ Georgia Brown", "", 60, Unit.cl);
        Product product4 = createContainerProductInProductGroup(productGroup2, "Extra
            ↪ Pilsner", "", 60, Unit.cl);
        Product product5 = createContainerProductInProductGroup(productGroup2, "Celebration"
            ↪ , "", 60, Unit.cl);
        Product product6 = createContainerProductInProductGroup(productGroup2, "Blondie", ""
            ↪ , 60, Unit.cl);
        Product product7 = createContainerProductInProductGroup(productGroup2, "Forårsbryg",
            ↪  "", 60, Unit.cl);
        Product product8 = createContainerProductInProductGroup(productGroup2, "India Pale
            ↪ Ale", "", 60, Unit.cl);
        Product product9 = createContainerProductInProductGroup(productGroup2, "Julebryg", "
            ↪ ", 60, Unit.cl);
        Product product10 = createContainerProductInProductGroup(productGroup2, "Juletønden"
            ↪ , "", 60, Unit.cl);
        Product product11 = createContainerProductInProductGroup(productGroup2, "Old Strong
            ↪ Ale", "", 60, Unit.cl);
        Product product12 = createContainerProductInProductGroup(productGroup2, "Fregatten
            ↪ Jylland", "", 60, Unit.cl);
        Product product13 = createContainerProductInProductGroup(productGroup2, "Imperial
            ↪ Stout", "", 60, Unit.cl);
        Product product14 = createContainerProductInProductGroup(productGroup2, "Tribute", "
            ↪ ", 60, Unit.cl);
        Product product15 = createContainerProductInProductGroup(productGroup2, "Black
            ↪ Monster", "", 60, Unit.cl);

        for (int i = 1; i < 15; i++) {
            Product product = storage.getProductList().get(i);
            addProductWithPriceToPricelist(pricelist1, product, 70);
            addProductWithPriceToPricelist(pricelist2, product, 36);
        }
        addProductWithPriceToPricelist(pricelist1, product15, 100);
        addProductWithPriceToPricelist(pricelist2, product15, 60);

        // Fadøl, 40 cl
        ProductGroup productGroup3 = createProductGroup("Fadøl, 40 cl");
        Product product16 = createContainerProductInProductGroup(productGroup3, "Klosterbryg
            ↪ ", "", 40, Unit.cl);
        Product product17 = createContainerProductInProductGroup(productGroup3, "Jazz Classic", "", 40, Unit.cl);
        Product product18 = createContainerProductInProductGroup(productGroup3, "Extra Pilsner", "", 40, Unit.cl);
        Product product19 = createContainerProductInProductGroup(productGroup3, "Celebration", "", 40, Unit.cl);
        Product product20 = createContainerProductInProductGroup(productGroup3, "Blondie", "", 40, Unit.cl);
        Product product21 = createContainerProductInProductGroup(productGroup3, "Forårsbryg", "", 40, Unit.cl);
        Product product22 = createContainerProductInProductGroup(productGroup3, "India Pale Ale", "", 40, Unit.cl);
        Product product23 = createContainerProductInProductGroup(productGroup3, "Julebryg", "", 40, Unit.cl);
        Product product24 = createContainerProductInProductGroup(productGroup3, "Imperial Stout", "", 40, Unit.cl);
        Product product25 = createContainerProductInProductGroup(productGroup3, "Special", "", 40, Unit.cl);

        for (int i = 15; i < 25; i++) {
            Product product = storage.getProductList().get(i);
            addProductWithPriceToPricelist(pricelist2, product, 38);
        }

        // Spiritus
        ProductGroup productGroup4 = createProductGroup("Spiritus");
        Product product26 = createContainerProductInProductGroup(productGroup4, "Spirit of Aarhus", "", 70, Unit.cl);
        Product product27 = createContainerProductInProductGroup(productGroup4, "SOA med pind", "", 70, Unit.cl);
        Product product28 = createContainerProductInProductGroup(productGroup4, "Whisky", "", 70, Unit.cl);
        Product product29 = createContainerProductInProductGroup(productGroup4, "Liquor of Aarhus", "", 70, Unit.cl);

        addProductWithPriceToPricelist(pricelist1, product26, 300);
        addProductWithPriceToPricelist(pricelist2, product26, 300);
        addProductWithPriceToPricelist(pricelist1, product27, 350);
        addProductWithPriceToPricelist(pricelist2, product27, 350);
        addProductWithPriceToPricelist(pricelist1, product28, 500);
        addProductWithPriceToPricelist(pricelist2, product28, 500);
        addProductWithPriceToPricelist(pricelist1, product29, 175);
        addProductWithPriceToPricelist(pricelist2, product29, 175);

        // Fustage
        ProductGroup productGroup5 = createProductGroup("Fustage");
        Product product30 = createContainerProductInProductGroup(productGroup5, "Klosterbryg", "", 20, Unit.l);
        product30.setRentableStrategy(new Rentable(200));
        Product product31 = createContainerProductInProductGroup(productGroup5, "Jazz Classic", "", 25, Unit.l);
        product31.setRentableStrategy(new Rentable(200));
        Product product32 = createContainerProductInProductGroup(productGroup5, "Extra Pilsner", "", 25, Unit.l);
        product32.setRentableStrategy(new Rentable(200));
        Product product33 = createContainerProductInProductGroup(productGroup5, "Celebration", "", 20, Unit.l);
        product33.setRentableStrategy(new Rentable(200));
```

```java
        Product product34 = createContainerProductInProductGroup(productGroup5, "Blondie", "
            ↪ ", 25, Unit.l);
        product34.setRentableStrategy(new Rentable(200));
        Product product35 = createContainerProductInProductGroup(productGroup5, "Forårsbryg"
            ↪ , "", 20, Unit.l);
        product35.setRentableStrategy(new Rentable(200));
        Product product36 = createContainerProductInProductGroup(productGroup5, "India Pale
            ↪ Ale", "", 20, Unit.l);
        product36.setRentableStrategy(new Rentable(200));
        Product product37 = createContainerProductInProductGroup(productGroup5, "Julebryg,",
            ↪  "", 20, Unit.l);
        product37.setRentableStrategy(new Rentable(200));
        Product product38 = createContainerProductInProductGroup(productGroup5, "Imperial
            ↪ Stout", "", 20, Unit.l);
        product38.setRentableStrategy(new Rentable(200));

        addProductWithPriceToPricelist(pricelist2, product30, 775);
        addProductWithPriceToPricelist(pricelist2, product31, 625);
        addProductWithPriceToPricelist(pricelist2, product32, 575);
        addProductWithPriceToPricelist(pricelist2, product33, 775);
        addProductWithPriceToPricelist(pricelist2, product34, 700);
        addProductWithPriceToPricelist(pricelist2, product35, 775);
        addProductWithPriceToPricelist(pricelist2, product36, 775);
        addProductWithPriceToPricelist(pricelist2, product37, 775);
        addProductWithPriceToPricelist(pricelist2, product38, 775);

        // Kulsyre
        ProductGroup productGroup6 = createProductGroup("Kulsyre");
        Product product39 = createContainerProductInProductGroup(productGroup6, "Kulsyre", "
            ↪ ", 6, Unit.kg);
        product39.setRentableStrategy(new Rentable(1000));
        Product product40 = createContainerProductInProductGroup(productGroup6, "Kulsyre", "
            ↪ ", 4, Unit.kg);
        product40.setRentableStrategy(new Rentable(1000));
        Product product41 = createContainerProductInProductGroup(productGroup6, "Kulsyre", "
            ↪ ", 10, Unit.kg);
        product41.setRentableStrategy(new Rentable(1000));

        addProductWithPriceToPricelist(pricelist1, product39, 400);
        addProductWithPriceToPricelist(pricelist2, product39, 400);

        // Malt
        ProductGroup productGroup7 = createProductGroup("Malt");
        Product product42 = createContainerProductInProductGroup(productGroup7, "Malt sæk",
            ↪ "", 25, Unit.kg);

        addProductWithPriceToPricelist(pricelist2, product42, 300);

        // Beklædning
        ProductGroup productGroup8 = createProductGroup("Beklædning");
        Product product43 = createProductInProductGroup(productGroup8, "T-shirt", "");
        Product product44 = createProductInProductGroup(productGroup8, "Polo", "");
        Product product45 = createProductInProductGroup(productGroup8, "Cap", "");

        addProductWithPriceToPricelist(pricelist1, product43, 70);
        addProductWithPriceToPricelist(pricelist2, product43, 70);
        addProductWithPriceToPricelist(pricelist1, product44, 100);
        addProductWithPriceToPricelist(pricelist2, product44, 100);
        addProductWithPriceToPricelist(pricelist1, product45, 30);
        addProductWithPriceToPricelist(pricelist2, product45, 30);

        // Anlæg
        ProductGroup productGroup9 = createProductGroup("Anlæg");
        Product product46 = createProductInProductGroup(productGroup9, "1-hane", "");
        product46.setRentableStrategy(new Rentable(250));
        Product product47 = createProductInProductGroup(productGroup9, "2-haner", "");
        product47.setRentableStrategy(new Rentable(400));
        Product product48 = createProductInProductGroup(productGroup9, "Bar med flere haner"
            ↪ , "");
        product48.setRentableStrategy(new Rentable(500));
        Product product49 = createProductInProductGroup(productGroup9, "Bar med flere haner"
            ↪ , "");
        product49.setRentableStrategy(new Rentable(500));
        Product product50 = createProductInProductGroup(productGroup9, "Krus", "");
        product50.setRentableStrategy(new Rentable(60));

        addProductWithPriceToPricelist(pricelist2, product46, 250);
        addProductWithPriceToPricelist(pricelist2, product47, 400);
        addProductWithPriceToPricelist(pricelist2, product48, 500);
        addProductWithPriceToPricelist(pricelist2, product49, 500);
        addProductWithPriceToPricelist(pricelist2, product50, 60);

        // Glas
        ProductGroup productGroup10 = createProductGroup("Glas");
        Product product51 = createProductInProductGroup(productGroup10, "Uanset størrelse",
            ↪ "");

        addProductWithPriceToPricelist(pricelist2, product51, 15);

        // Sampakning
        ProductGroup productGroup11 = createProductGroup("Sampakning");
        Bundle product52 = createBundleProductInProductGroup(productGroup11, "Gaveæske 2 øl,
            ↪  2 glas", "");
        for (int i = 0; i < 2; i++) {
            product52.addProduct(product2);
            product52.addProduct(product51);
        }
        Bundle product53 = createBundleProductInProductGroup(productGroup11, "Gaveæske 4 øl"
            ↪ , "");
        for (int i = 0; i < 4; i++) {
            product53.addProduct(product2);
        }
        Bundle product54 = createBundleProductInProductGroup(productGroup11, "Trækasse 6 øl"
            ↪ , "");
        for (int i = 0; i < 6; i++) {
            product54.addProduct(product2);
        }
        Bundle product55 = createBundleProductInProductGroup(productGroup11, "Gavekurv 6 øl,
            ↪  2 glas", "");
        for (int i = 0; i < 6; i++) {
            product55.addProduct(product2);
        }
        product55.addProduct(product51);
        product55.addProduct(product51);
        Bundle product56 = createBundleProductInProductGroup(productGroup11, "Trækasse 6 øl,
            ↪  6 glas", "");
        for (int i = 0; i < 6; i++) {
            product56.addProduct(product2);
            product56.addProduct(product51);
        }
        Bundle product57 = createBundleProductInProductGroup(productGroup11, "Trækasse 12 øl
            ↪ ", "");
        for (int i = 0; i < 12; i++) {
            product57.addProduct(product2);
        }
        Bundle product58 = createBundleProductInProductGroup(productGroup11, "papkasse 12 øl
            ↪ ", "");
        for (int i = 0; i < 12; i++) {
            product58.addProduct(product2);
        }

        // Rundvisning
        ProductGroup productGroup12 = createProductGroup("Rundvisning");
        Product product59 = createTourProductInProductGroup(productGroup12, "Rundvisning pr
            ↪ person", "");
        product59.setRentableStrategy(new Rentable(0));

        pricelist3.addProductWithPrice(product59, 100);

        // ---------------------------

        // Snacks
        ProductGroup pgSnacks = createProductGroup("Snacks");
        Product chips = createProductInProductGroup(pgSnacks, "Chips", "");
        Product peanuts = createProductInProductGroup(pgSnacks, "Peanuts", "");
        Product chokolade = createProductInProductGroup(pgSnacks, "Chokolade", "");
        Product kapsler = createProductInProductGroup(pgSnacks, "Kapsler", "");
        Bundle bValentinskurv = createBundleProductInProductGroup(pgSnacks, "Karstens
            ↪ Valentinskurv", "");
        bValentinskurv.addProduct(chips);
        bValentinskurv.addProduct(peanuts);
        Bundle bHjemmehygge = createBundleProductInProductGroup(pgSnacks, "Hjemme hygge", ""
            ↪ );
        bHjemmehygge.addProduct(chokolade);
        bHjemmehygge.addProduct(chips);

        addProductWithPriceToPricelist(pricelist1, chips, 10);
        addProductWithPriceToPricelist(pricelist1, peanuts, 10);
        addProductWithPriceToPricelist(pricelist1, chokolade, 15);
        addProductWithPriceToPricelist(pricelist1, kapsler, 10);
        addProductWithPriceToPricelist(pricelist1, bValentinskurv, 20);
        addProductWithPriceToPricelist(pricelist1, bHjemmehygge, 20);

        Random random = new Random();
        int pricelistCount = pricelist1.getProductsWithPrice().size();

        for (int i = 0; i < 10000; i++) {
            int rngProductIndex = random.nextInt(pricelistCount);
            // Unoptimized
            Product randomProduct = (Product) pricelist1.getProductsWithPrice().keySet().
                ↪ toArray()[rngProductIndex];
            double productPrice = pricelist1.getPriceOfProduct(randomProduct);
            Sale sale = createSale();
            sale.updateSalesLine(randomProduct, 1, productPrice);
            LocalDateTime localDateTime = LocalDateTime.of(2020, random.nextInt(12) + 1,
               ↪ random.nextInt(29) + 1, 12, 0);
            sale.setTimestamp(localDateTime);
            PaymentMethod paymentMethod = PaymentMethod.values()[random.nextInt(
               ↪ PaymentMethod.values().length)];
            sale.addTransfer(paymentMethod, productPrice);
            if (i % 5000 == 0)
                sale.setSaleState(SaleState.DELAYED);
            else
                sale.setSaleState(SaleState.COMPLETED);
        }

        // Create dummy tour
        Sale tourSale = createRent("Alexander", "Ikke tilgængelig", LocalDateTime.now(),
           ↪ LocalDateTime.now());
        tourSale.updateSalesLine(product59, 42, 100);
    }
}
```

Chapter 2: gui/administrationpane/AdministrationPane.java

```java
package gui.administrationpane;

import gui.LeftSideBar;
import gui.LeftSideBarInterface;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;

import java.util.LinkedHashMap;
import java.util.Map;

public class AdministrationPane extends BorderPane implements LeftSideBarInterface {

    private static AdministrationPane administrationPane;

    private LeftSideBar leftSideBar;

    private AdministrationPane() {
        this.setPadding(new Insets(20));

        Map<String, Pane> navigationMap = new LinkedHashMap<>();
        navigationMap.put("Produkt gruppe", NewProductGroupPane.getInstance());
        navigationMap.put("Produkt", NewProductPane.getInstance());
        navigationMap.put("Prislister", PricelistPane.getInstance());

        leftSideBar = new LeftSideBar(navigationMap, this);
        setLeft(leftSideBar);

        // Creating panes

        changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
    }

    @Override
    public void changeSelected(String nameOfButton) {
        Label title = new Label(nameOfButton);
        title.getStyleClass().add("title");
        setAlignment(title, Pos.CENTER);
        setTop(title);
        setCenter(leftSideBar.getPane(nameOfButton));
    }


    public static AdministrationPane getInstance() {
        if (administrationPane == null)
            administrationPane = new AdministrationPane();
        return administrationPane;
    }
}
```

Chapter 3: gui/administrationpane/NewProductGroupPane.java

```java
package gui.administrationpane;

import controller.Controller;
import gui.helpers.TableViewHelper;
import gui.helpers.ValidationHelper;
import gui.overviewpane.ProductOverviewPane;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Font;
import model.ProductGroup;

public class NewProductGroupPane extends GridPane {

        private static NewProductGroupPane newProductGroupPane;

        private TextField txfName;

        private NewProductGroupPane() {
                this.getStyleClass().add("gridpane");

                // Title
                Label lblTitle = new Label("Ny produktgruppe");
                lblTitle.setFont(new Font(24));
                lblTitle.setPrefWidth(200);
                this.add(lblTitle, 0, 0, 2, 1);

                this.add(new Separator(), 0, 1, 2, 1);

                // Fields
                Label lblName = new Label("Navn:");
                this.add(lblName, 0, 2);
                txfName = new TextField();
                txfName.setPromptText("Navn");
                this.add(txfName, 1, 2);

                Button btnCreateProduct = new Button("Opret Produktgruppe");
                this.add(btnCreateProduct, 1, 3);
                btnCreateProduct.setOnAction(event -> createProductGroup());

                // Table
                TableView<ProductGroup> productGroupTableView = TableViewHelper.
                    ↪ createProductGroupTable();
                this.add(productGroupTableView, 2, 0, 1, 5);
                productGroupTableView.setPrefHeight(700);
                productGroupTableView.setPrefWidth(300);
                productGroupTableView.setColumnResizePolicy(TableView.
                    ↪ CONSTRAINED_RESIZE_POLICY);
                productGroupTableView.setEditable(false);
        }

        private void createProductGroup() {
                boolean hasError = false;
                String name = txfName.getText().trim();
                if (!ValidationHelper.isStringBetween0to30characters(name)) {
                        txfName.getStyleClass().add("error");
                        hasError = true;
                }
                if (!hasError) {
                        ProductGroup productGroup = Controller.getInstance().
                            ↪ createProductGroup(name);
                        TableViewHelper.addProductGroupToObservableList(productGroup);
                        ProductOverviewPane.getInstance().update();

                        txfName.clear();
                        ValidationHelper.removeErrorClassStyle(this);
                }
        }

        public static void update() {
                newProductGroupPane = new NewProductGroupPane();
        }

        public static NewProductGroupPane getInstance() {
                if (newProductGroupPane == null)
                        newProductGroupPane = new NewProductGroupPane();
                return newProductGroupPane;
        }
}
```

Chapter 4: gui/administrationpane/NewProductPane.java

```java
package gui.administrationpane;

import controller.Controller;
import gui.helpers.TableViewHelper;
import gui.helpers.ValidationHelper;
import gui.overviewpane.ProductOverviewPane;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.text.Font;
import model.ProductGroup;
import model.enums.Unit;
import model.product.Bundle;
import model.product.Container;
import model.product.Product;
import model.product.Tour;
import model.product.rentable.Rentable;

import java.util.HashMap;
import java.util.Map;

public class NewProductPane extends GridPane {

        private static NewProductPane productPane;

        private TextField txfName;
        private TextArea txfDescription;
        private ChoiceBox<ProductGroup> productGroupChoiceBox;
        private CheckBox rentableCheckBox;
        private TextField txfSize;
        private ChoiceBox<Unit> UnitChoiceBox;

        String[] productMenuNames = { "Standard", "Standard (med størrelse)", "Sampakning",
            ↪ "Rundvisning" };
        private Map<String, ToggleButton> toggleButtonMap;

        private TextField txfDeposit;

        private TableView<Product> productTableView;

        private TableView<Product> bundleProductsTableView;

        private GridPane currentProductGridpane;

        private NewProductPane() {
                this.getStyleClass().add("gridpane");

                // Title
                Label lblTitle = new Label("Nyt produkt");
                lblTitle.setFont(new Font(24));
                lblTitle.setPrefWidth(300);
                this.add(lblTitle, 0, 0, 2, 1);

                this.add(new Separator(), 0, 1, 2, 1);

                currentProductGridpane = new GridPane();
                currentProductGridpane.getStyleClass().add(String.valueOf(this.getStyleClass
                    ↪ ()));
                this.add(currentProductGridpane, 0, 3);

                HBox hBox = new HBox();
                hBox.setAlignment(Pos.CENTER);
                hBox.setSpacing(5);
                this.add(hBox, 0, 2, 2, 1);

                toggleButtonMap = new HashMap<>();

                ToggleGroup toggleGroup = new ToggleGroup();

                for (String productMenuName : productMenuNames) {
                        ToggleButton toggleButton = new ToggleButton(productMenuName);
                        if (productMenuName.equals(productMenuNames[0])) { // Set "Standard"
                            ↪ as selected
                                toggleButton.setSelected(true);
                                createStandardProductMenu();
                        }
                        toggleButtonMap.put(productMenuName, toggleButton);
                        hBox.getChildren().add(toggleButton);
                        toggleButton.setToggleGroup(toggleGroup);
                }

                toggleButtonMap.get(productMenuNames[0]).setOnAction(event ->
                    ↪ createStandardProductMenu());
                toggleButtonMap.get(productMenuNames[1]).setOnAction(event ->
                    ↪ createContainerProductMenu());
                toggleButtonMap.get(productMenuNames[2]).setOnAction(event ->
                    ↪ createBundleProductMenu());
                toggleButtonMap.get(productMenuNames[3]).setOnAction(event ->
                    ↪ createTourProductMenu());

                Label lblProductList = new Label("Produktliste:");
                this.add(lblProductList, 2, 0);

                // Table
                bundleProductsTableView = null;
                productTableView = TableViewHelper.createProductWithDescriptionTable();
                this.add(productTableView, 2, 1, 1, 4);
                productTableView.setPrefHeight(500);
                productTableView.setPrefWidth(375);
                productTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
                productTableView.setEditable(false);
        }

        private void clearFields() {
                txfName.clear();
                txfDescription.clear();
                txfDeposit.clear();
                if (txfSize != null)
                        txfSize.clear();
                if (bundleProductsTableView != null) {
                        bundleProductsTableView.getItems().clear();
                }
                ValidationHelper.removeErrorClassStyle(currentProductGridpane);
        }

        private void update(Product product) {
                clearFields();
                TableViewHelper.addProductToObservableList(product);
                PricelistPane.getInstance().addNewProduct(product);
                ProductOverviewPane.getInstance().update();
        }

        private String getProductName() {
                return txfName.getText().trim();
        }

        private String getProductDescription() {
                return txfDescription.getText().trim();
        }

        private ProductGroup getProductGroup() {
                return productGroupChoiceBox.getSelectionModel().getSelectedItem();
        }

        private Double getDeposit() {
                return Double.parseDouble(txfDeposit.getText());
        }

        private Unit getUnit() {
                return UnitChoiceBox.getValue();
        }

        private Double getSize() {
                return Double.parseDouble(txfSize.getText());
        }

        private boolean isRentable() {
                return rentableCheckBox.isSelected();
        }

        private void createProduct() {
                if (!hasError()) {
                        Product product = Controller.getInstance().
                            ↪ createProductInProductGroup(getProductGroup(), getProductName
                            ↪ (),
                                        getProductDescription());

                        if (isRentable())
                                product.setRentableStrategy(new Rentable(getDeposit()));

                        update(product); // update the pane(s)
                }
```

```java
	}

	private void createContainerProduct() {
		if (!hasError()) {
			Container container = Controller.getInstance().
			    ↪ createContainerProductInProductGroup(getProductGroup(),
					getProductName(), getProductDescription(), getSize()
					    ↪ , getUnit());

			if (isRentable())
				container.setRentableStrategy(new Rentable(getDeposit()));

			update(container);
		}
	}

	private void createBundleProduct() {
		if (!hasError()) {
			Bundle bundle = Controller.getInstance().
			    ↪ createBundleProductInProductGroup(getProductGroup(),
					getProductName(), getProductDescription());
			for (Product product : bundleProductsTableView.getItems()) {
				bundle.addProduct(product);

			}

			if (isRentable())
				bundle.setRentableStrategy(new Rentable(getDeposit()));

			update(bundle);
		}
	}

	private void createTourProduct() {
		if (!hasError()) {
			Tour tour = Controller.getInstance().createTourProductInProductGroup
			    ↪ (getProductGroup(), getProductName(),
					getProductDescription());
			tour.setRentableStrategy(new Rentable(0d));

			update(tour); // update the pane(s)
		}

	}

	private boolean hasError() {
		boolean hasError = false;
		String name = txfName.getText().trim();
		if (!ValidationHelper.isStringBetween0to30characters(name)) {
			txfName.getStyleClass().add("error");
			hasError = true;
		}

		ProductGroup productGroup = productGroupChoiceBox.getSelectionModel().
		    ↪ getSelectedItem();
		if (productGroup == null) {
			productGroupChoiceBox.getStyleClass().add("error");
			hasError = true;
```

```
        }

        if (toggleButtonMap.get(productMenuNames[1]).isSelected()) {
            if (!ValidationHelper.isNumber(txfSize.getText().trim()) || txfSize.
                ↪ getText().isEmpty()) {
                txfSize.getStyleClass().add("error");
                hasError = true;
            }
        }

        if (rentableCheckBox.isSelected())
            if (!ValidationHelper.isNumberAndPositive(txfDeposit.getText().trim
                ↪ ())) {
                txfDeposit.getStyleClass().add("error");
                hasError = true;
            }
        return hasError;
    }

    private void createStandardProductMenu() {
        currentProductGridpane.getChildren().clear();

        // Fields
        Label lblName = new Label("Navn:");
        lblName.setPrefWidth(125);
        currentProductGridpane.add(lblName, 0, 0);
        txfName = new TextField();
        txfName.setPromptText("Navn");
        currentProductGridpane.add(txfName, 1, 0);

        Label lblDescription = new Label("Beskrivelse:");
        currentProductGridpane.add(lblDescription, 0, 1);
        txfDescription = new TextArea();
        txfDescription.setPromptText("Beskrivelse");
        txfDescription.setPrefRowCount(2);
        txfDescription.setPrefHeight(60);
        txfDescription.setPrefWidth(txfName.getWidth());
        txfDescription.setWrapText(true);
        currentProductGridpane.add(txfDescription, 1, 1);

        Label lblProductGroup = new Label("Produktgruppe:");
        currentProductGridpane.add(lblProductGroup, 0, 2);
        productGroupChoiceBox = new ChoiceBox<>();
        productGroupChoiceBox.setItems(TableViewHelper.getProductGroupObservableList
            ↪ ());
        currentProductGridpane.add(productGroupChoiceBox, 1, 2);

        Label lblRentable = new Label("Kan udlejes:");
        currentProductGridpane.add(lblRentable, 0, 3);

        HBox hBox = new HBox();
        hBox.setSpacing(5);
        hBox.setAlignment(Pos.CENTER_LEFT);
        currentProductGridpane.add(hBox, 1, 3);

        rentableCheckBox = new CheckBox();
        hBox.getChildren().add(rentableCheckBox);

```

```java
                txfDeposit = new TextField();
                txfDeposit.setPromptText("Pant/depositum");
                txfDeposit.setDisable(true);
                hBox.getChildren().add(txfDeposit);

                rentableCheckBox.selectedProperty().addListener((observable, oldValue,
                    ↪ newValue) -> {
                        if (newValue)
                                txfDeposit.setDisable(false);
                        else {
                                txfDeposit.setText("");
                                txfDeposit.setDisable(true);
                        }
                });

                Button btnCreateProduct = new Button("Opret produkt");
                currentProductGridpane.add(btnCreateProduct, 1, 4);
                btnCreateProduct.setOnAction(event -> createProduct());
        }

        private void createContainerProductMenu() {
                currentProductGridpane.getChildren().clear();

                // Fields
                Label lblName = new Label("Navn:");
                lblName.setPrefWidth(125);
                currentProductGridpane.add(lblName, 0, 0);
                txfName = new TextField();
                txfName.setPromptText("Navn");
                currentProductGridpane.add(txfName, 1, 0);

                Label lblDescription = new Label("Beskrivelse:");
                currentProductGridpane.add(lblDescription, 0, 1);
                txfDescription = new TextArea();
                txfDescription.setPromptText("Beskrivelse");
                txfDescription.setPrefRowCount(2);
                txfDescription.setPrefHeight(60);
                txfDescription.setPrefWidth(txfName.getWidth());
                txfDescription.setWrapText(true);
                currentProductGridpane.add(txfDescription, 1, 1);

                Label lblProductGroup = new Label("Produktgruppe:");
                currentProductGridpane.add(lblProductGroup, 0, 2);
                productGroupChoiceBox = new ChoiceBox<>();
                productGroupChoiceBox.setItems(TableViewHelper.getProductGroupObservableList
                    ↪ ());
                currentProductGridpane.add(productGroupChoiceBox, 1, 2);

                Label lblRentable = new Label("Kan udlejes:");
                currentProductGridpane.add(lblRentable, 0, 3);

                HBox hBox = new HBox();
                hBox.setSpacing(5);
                hBox.setAlignment(Pos.CENTER_LEFT);
                currentProductGridpane.add(hBox, 1, 3);

                rentableCheckBox = new CheckBox();
                hBox.getChildren().add(rentableCheckBox);

                txfDeposit = new TextField();
                txfDeposit.setPromptText("Pant/depositum");
                txfDeposit.setDisable(true);
                hBox.getChildren().add(txfDeposit);

                rentableCheckBox.selectedProperty().addListener((observable, oldValue,
                    ↪ newValue) -> {
                        if (newValue)
                                txfDeposit.setDisable(false);
                        else {
                                txfDeposit.setText("");
                                txfDeposit.setDisable(true);
                        }
                });

                Label lblSize = new Label("Størrelse:");
                currentProductGridpane.add(lblSize, 0, 4);

                HBox hBoxSize = new HBox();
                hBoxSize.setSpacing(5);
                hBoxSize.setAlignment(Pos.TOP_LEFT);
                currentProductGridpane.add(hBoxSize, 1, 4);

                txfSize = new TextField();
                txfSize.setMaxWidth(50);
                hBoxSize.getChildren().add(txfSize);
                UnitChoiceBox = new ChoiceBox<>();
                for (Unit unit : Unit.values()) {
                        UnitChoiceBox.getItems().add(unit);
                }
                UnitChoiceBox.getSelectionModel().selectFirst();
                hBoxSize.getChildren().add(UnitChoiceBox);

                Button btnCreateProduct = new Button("Opret produkt");
                currentProductGridpane.add(btnCreateProduct, 1, 5);
                btnCreateProduct.setOnAction(event -> createContainerProduct());
        }

        private void createBundleProductMenu() {
                currentProductGridpane.getChildren().clear();

                // Fields
                Label lblName = new Label("Bundt navn:");
                lblName.setPrefWidth(125);
                currentProductGridpane.add(lblName, 0, 0);
                txfName = new TextField();
                txfName.setPromptText("Navn");
                currentProductGridpane.add(txfName, 1, 0);

                Label lblProductGroup = new Label("Produktgruppe:");
                currentProductGridpane.add(lblProductGroup, 0, 2);
                productGroupChoiceBox = new ChoiceBox<>();
                productGroupChoiceBox.setItems(TableViewHelper.getProductGroupObservableList
                    ↪ ());
                currentProductGridpane.add(productGroupChoiceBox, 1, 2);

                /////
```

```java

                Label lblRentable = new Label("Kan udlejes:");
                currentProductGridpane.add(lblRentable, 0, 3);

                HBox hBox = new HBox();
                hBox.setSpacing(5);
                hBox.setAlignment(Pos.CENTER_LEFT);
                currentProductGridpane.add(hBox, 1, 3);
                rentableCheckBox = new CheckBox();
                hBox.getChildren().add(rentableCheckBox);

                txfDeposit = new TextField();
                txfDeposit.setPromptText("Pant/depositum");
                txfDeposit.setDisable(true);
                hBox.getChildren().add(txfDeposit);

                rentableCheckBox.selectedProperty().addListener((observable, oldValue,
                   ↪ newValue) -> {
                        if (newValue)
                                txfDeposit.setDisable(false);
                        else {
                                txfDeposit.setText("");
                                txfDeposit.setDisable(true);
                        }
                });

                ///

                Label lblProductsInBundle = new Label("Produkter i sampakning:");
                currentProductGridpane.add(lblProductsInBundle, 0, 4);

                HBox hbProductsInBundle = new HBox();
                hbProductsInBundle.setSpacing(5);
                hbProductsInBundle.setAlignment(Pos.CENTER);
                currentProductGridpane.add(hbProductsInBundle, 0, 5, 2, 2);

                bundleProductsTableView = new TableView<>();
                bundleProductsTableView.setPrefHeight(300);
                bundleProductsTableView.setPrefWidth(250);
                bundleProductsTableView.setColumnResizePolicy(TableView.
                   ↪ CONSTRAINED_RESIZE_POLICY);
                bundleProductsTableView.setEditable(false);
                hbProductsInBundle.getChildren().add(bundleProductsTableView);

                Button btnAddToBundle = new Button("Tilføj");
                btnAddToBundle.setOnAction(event -> addToBundle());
                hbProductsInBundle.getChildren().add(btnAddToBundle);

                TableColumn<Product, String> column1 = new TableColumn<>("Navn");
                column1.setCellValueFactory(new PropertyValueFactory<>("name"));

                TableColumn<Product, String> column2 = new TableColumn<>("Produktgruppe");
                column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));

                bundleProductsTableView.getColumns().add(column1);
                bundleProductsTableView.getColumns().add(column2);

                Button btnCreateBundle = new Button("Opret sampakning");
```

```
		btnCreateBundle.setOnAction(event -> createBundleProduct());
		currentProductGridpane.add(btnCreateBundle, 1, 8);
	}

	private void addToBundle() {
		Product product = productTableView.getSelectionModel().getSelectedItem();
		if (product != null && !(product instanceof Bundle)) {
			bundleProductsTableView.getItems().add(product);
		}
	}

	private void createTourProductMenu() {
		currentProductGridpane.getChildren().clear();

		// Fields
		Label lblName = new Label("Navn:");
		lblName.setPrefWidth(125);
		currentProductGridpane.add(lblName, 0, 0);
		txfName = new TextField();
		txfName.setPromptText("Navn");
		currentProductGridpane.add(txfName, 1, 0);

		Label lblDescription = new Label("Beskrivelse:");
		currentProductGridpane.add(lblDescription, 0, 1);
		txfDescription = new TextArea();
		txfDescription.setPromptText("Beskrivelse");
		txfDescription.setPrefRowCount(2);
		txfDescription.setPrefHeight(60);
		txfDescription.setPrefWidth(txfName.getWidth());
		txfDescription.setWrapText(true);
		currentProductGridpane.add(txfDescription, 1, 1);

		Label lblProductGroup = new Label("Produktgruppe:");
		currentProductGridpane.add(lblProductGroup, 0, 2);
		productGroupChoiceBox = new ChoiceBox<>();
		productGroupChoiceBox.setItems(TableViewHelper.getProductGroupObservableList
		    ↪ ());
		currentProductGridpane.add(productGroupChoiceBox, 1, 2);

		Button btnCreateProduct = new Button("Opret rundvisning");
		currentProductGridpane.add(btnCreateProduct, 1, 4);
		btnCreateProduct.setOnAction(event -> createTourProduct());
	}

	public static NewProductPane getInstance() {
		if (productPane == null)
			productPane = new NewProductPane();
		return productPane;
	}
}
```

Chapter 5: gui/administrationpane/PricelistPane.java

```java
package gui.administrationpane;

import controller.Controller;
import gui.helpers.ProductAndPriceHelper;
import gui.helpers.TableViewHelper;
import gui.helpers.ValidationHelper;
import gui.overviewpane.PricelistOverviewPane;
import gui.salepane.SelectPricelistPane;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.util.converter.DoubleStringConverter;
import model.Pricelist;
import model.product.Product;

import java.util.LinkedList;
import java.util.List;

public class PricelistPane extends GridPane {

    private static PricelistPane pricelistPane;

    private TextField txfName;

    private Pricelist selectedPricelist;

    private TableView<ProductAndPriceHelper> productInPricelistTableView;
    private static ObservableList<ProductAndPriceHelper> leftList = FXCollections.
        ↪ observableList(new LinkedList<>());
    private TableView<ProductAndPriceHelper> productNotInPricelistTableView;
    private static ObservableList<ProductAndPriceHelper> rightList = FXCollections.
        ↪ observableList(new LinkedList<>());

    private PricelistPane() {
        this.getStyleClass().add("gridpane");

        // Title
        Label lblTitle = new Label("Ny prisliste");
        lblTitle.setFont(new Font(24));
        lblTitle.setPrefWidth(200);
        this.add(lblTitle, 0, 0, 2, 1);

        this.add(new Separator(), 0, 1, 2, 1);

        Label lblName = new Label("Navn:");
        this.add(lblName, 0, 2);
        txfName = new TextField();
        txfName.setPromptText("Navn");
        this.add(txfName, 1, 2);

        Button btnCreatePricelist = new Button("Opret prisliste");
        this.add(btnCreatePricelist, 1, 3);
        btnCreatePricelist.setOnAction(event -> createPricelist());

        Label lblChoosePricelist = new Label("Vælg prisliste:");
        this.add(lblChoosePricelist, 0, 4);

        TableView<Pricelist> pricelistTableView = TableViewHelper.createPricelistTable();
        this.add(pricelistTableView, 0, 5, 2, 1);
        pricelistTableView.setPrefHeight(500);
        pricelistTableView.setPrefWidth(200);
        pricelistTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        pricelistTableView.setEditable(false);
        pricelistTableView.getSelectionModel().selectedItemProperty().addListener((obs,
            ↪ oldSelection, newSelection) -> {
            if (newSelection != null) {
                selectedPricelist = newSelection;
                updateLists();
            }
        });

        Label lblLeftTable = new Label("Prislistens indhold: (ret i pris, enter for at gemme
            ↪ )");
        this.add(lblLeftTable, 2, 0, 2, 1);

        // Left table
        productInPricelistTableView = new TableView<>();
        this.add(productInPricelistTableView, 2, 1, 2, 5);
        productInPricelistTableView.setPrefHeight(700);
        productInPricelistTableView.setPrefWidth(400);
        productInPricelistTableView.setColumnResizePolicy(TableView.
            ↪ CONSTRAINED_RESIZE_POLICY);
        createProductInPricelistTableView();

        VBox vBox = new VBox();
        vBox.setAlignment(Pos.CENTER);
        vBox.setSpacing(20);
        this.add(vBox, 4, 0, 1, 6);

        Button btnLeft = new Button("<-");
        vBox.getChildren().add(btnLeft);
        btnLeft.setOnAction(event -> moveLeft());

        Button btnRight = new Button("->");
        vBox.getChildren().add(btnRight);
        btnRight.setOnAction(event -> moveRight());

        Label lblRightTable = new Label("Produkter til tilføjelse:");
        this.add(lblRightTable, 5, 0, 2, 1);

        // Right table
        productNotInPricelistTableView = new TableView<>();
        this.add(productNotInPricelistTableView, 5, 1, 2, 5);
        productNotInPricelistTableView.setPrefHeight(700);
        productNotInPricelistTableView.setPrefWidth(400);
        productNotInPricelistTableView.setColumnResizePolicy(TableView.
            ↪ CONSTRAINED_RESIZE_POLICY);
        createProductNotInPricelistTableView();
    }

    private void createPricelist() {
        String name = txfName.getText().trim();
        if (!ValidationHelper.isStringBetween0to30characters(name)) {
            txfName.getStyleClass().add("error");
            return;
        }
        Pricelist pricelist = Controller.getInstance().createPricelist(name);
        TableViewHelper.addPricelistToObservableList(pricelist);
        SelectPricelistPane.getInstance().addPricelist(pricelist);
        txfName.clear();
        ValidationHelper.removeErrorClassStyle(this);
    }

    private void createProductInPricelistTableView() {
        productInPricelistTableView.setEditable(true);

        TableColumn<ProductAndPriceHelper, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("product"));

        TableColumn<ProductAndPriceHelper, String> column2 = new TableColumn<>("
            ↪ Produktgruppe");
        column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));

        TableColumn<ProductAndPriceHelper, Double> column3 = new TableColumn<>("Pris");
        column3.setCellValueFactory(new PropertyValueFactory<>("price"));
        column3.setCellFactory(TextFieldTableCell.forTableColumn(new DoubleStringConverter()
            ↪ ));
        column3.setOnEditCommit(event -> {
            ProductAndPriceHelper ProductAndPriceHelper = event.getRowValue();
            ProductAndPriceHelper.setPrice(event.getNewValue());
            selectedPricelist.updateProductPrice(ProductAndPriceHelper.getProduct(),
                ↪ ProductAndPriceHelper.getPrice());
            PricelistOverviewPane.getInstance().update();
        });
        column3.setEditable(true);
        column3.setMinWidth(60);
        column3.setMaxWidth(60);

        productInPricelistTableView.getColumns().add(column1);
        productInPricelistTableView.getColumns().add(column2);
        productInPricelistTableView.getColumns().add(column3);

        productInPricelistTableView.setItems(leftList);
    }

    private void createProductNotInPricelistTableView() {
        productNotInPricelistTableView.setEditable(false);

        TableColumn<ProductAndPriceHelper, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("product"));

        TableColumn<ProductAndPriceHelper, String> column2 = new TableColumn<>("
            ↪ Produktgruppe");
        column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));

        productNotInPricelistTableView.getColumns().add(column1);
        productNotInPricelistTableView.getColumns().add(column2);

        productNotInPricelistTableView.setItems(rightList);
    }

    public void updateLists() {
        if (selectedPricelist == null)
            return;

        leftList.clear();
        leftList.setAll(ProductAndPriceHelper.parseToProductAndPrice(selectedPricelist.
            ↪ getProductsWithPrice()));

        List<Product> productList = Controller.getInstance().getProductList();
        productList.removeAll(ProductAndPriceHelper.getProductList(leftList));
        rightList.setAll(ProductAndPriceHelper.parseToProductAndPrice(productList));
    }

    public void addNewProduct(Product product) {
        rightList.add(new ProductAndPriceHelper(product, 0));
    }

    private void moveLeft() {
        ProductAndPriceHelper ProductAndPriceHelper = productNotInPricelistTableView.
            ↪ getSelectionModel()
                .getSelectedItem();
        if (productNotInPricelistTableView.getSelectionModel().getSelectedItem() != null) {
            rightList.remove(ProductAndPriceHelper);
            leftList.add(ProductAndPriceHelper);
        }
        selectedPricelist.addProductWithPrice(ProductAndPriceHelper.getProduct(),
            ↪ ProductAndPriceHelper.getPrice());
        PricelistOverviewPane.getInstance().update();
    }

    private void moveRight() {
        ProductAndPriceHelper ProductAndPriceHelper = productInPricelistTableView.
            ↪ getSelectionModel().getSelectedItem();
        if (productInPricelistTableView.getSelectionModel().getSelectedItem() != null) {
            leftList.remove(ProductAndPriceHelper);
            rightList.add(ProductAndPriceHelper);
        }
        selectedPricelist.removeProductWithPrice(ProductAndPriceHelper.getProduct());
    }

    public static PricelistPane getInstance() {
        if (pricelistPane == null)
            pricelistPane = new PricelistPane();
        return pricelistPane;
    }
```

```
}
```

Chapter 6: gui/css/app.css

```css
/* ====== General Style */
.label.title {
    -fx-font-size: 32px;
}

.label.subtitle {
    -fx-font-size: 15px;
}

/* ... Error Handling*/
.text-field.error, .date-picker.error, .choice-box.error, .text-area.error {
    -fx-background-color: #EF6B71;
}

.gridpane {
    -fx-padding: 20;
    -fx-hgap: 20;
    -fx-vgap: 10;
}

/* ====== Specific elements */
.button.important {
    -fx-background-color: green;
    -fx-min-width: 150;
    -fx-min-height: 150;
}
.button.important:hover {
    -fx-background-color: lightgreen;
}

.label.clock {
    -fx-font-size: 14px;
    -fx-font-weight: bold;
    -fx-font-family: Menlo;
}

.dateinterval-picker {
    -fx-alignment: center;
}
/* Mono font for better readability */
.salesLine {
    -fx-font-family: "Courier New";
}

/* ... Tour */
.tour .month-year-pane {
    -fx-background-color: aqua;
    -fx-padding: 5 5 5 5;
}

.tour .day-name-cell {
    -fx-background-color: black;
    -fx-text-fill: aqua;
}

.tour-cell.weekend {
    -fx-background-color: #70a666;
}

.tour-cell.weekday {
    -fx-background-color: #ffda55;
}

.tour-cell.occupied {
    -fx-background-color: #913a3a;
}
```

Chapter 7: gui/helpers/Calendar.java

```java
package gui.helpers;

import com.sun.javafx.scene.control.skin.DatePickerSkin;
import javafx.scene.Node;
import javafx.scene.layout.HBox;
import model.Sale;

import java.util.List;

public class Calendar extends HBox {

    public Calendar(List<Sale> sales) {
       ProductDatePicker productDatePicker = new ProductDatePicker(sales);

       DatePickerSkin datePickerSkin = new DatePickerSkin(productDatePicker);
       Node content = datePickerSkin.getPopupContent();

       this.getChildren().add(content);
    }
}
```

Chapter 8: gui/helpers/DateIntervalPicker.java

```java
package gui.helpers;

import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;

import java.time.LocalDate;
import java.util.Arrays;

public class DateIntervalPicker extends HBox {
    private DatePicker datePickerFrom;
    private DatePicker datePickerTo;

    public DateIntervalPicker() {
        this.getStyleClass().add("dateinterval-picker");
        Label lblFrom = new Label("Fra: ");
        Label lblTo = new Label("Til: ");
        datePickerFrom = new DatePicker();
        datePickerFrom.setValue(LocalDate.now().minusDays(30));
        datePickerTo = new DatePicker();
        datePickerTo.setValue(LocalDate.now());

        datePickerFrom.valueProperty().addListener(observable ->{
            if (datePickerFrom.getValue().isAfter(datePickerTo.getValue()))
                datePickerFrom.setValue(datePickerTo.getValue().minusDays(30));
        });

        datePickerTo.valueProperty().addListener(observable -> {
            if (datePickerTo.getValue().isBefore(datePickerFrom.getValue()))
                datePickerTo.setValue(datePickerFrom.getValue().plusDays(30));
        });

        this.getChildren().addAll(Arrays.asList(lblFrom, datePickerFrom, lblTo, datePickerTo
            ↪ ));
    }

    public LocalDate getDateFrom() {
        return datePickerFrom.getValue();
    }

    public LocalDate getDateTo() {
        return datePickerTo.getValue();
    }
}
```

Chapter 9: gui/helpers/ProductAndPriceHelper.java

```java
package gui.helpers;

import model.ProductGroup;
import model.product.Product;

import java.util.LinkedList;
import java.util.List;
import java.util.Map;


/**
 * Helper class for GUI - Makes TableViews possible and easier
 */
public class ProductAndPriceHelper {
    private Product product;
    private double price;

    public ProductAndPriceHelper(Product product, double price) {
        this.product = product;
        this.price = price;
    }

    public String getName(){
        return product.getName();
    }

    public Product getProduct() {
        return product;
    }

    public ProductGroup getProductGroup() {
        return product.getProductGroup();
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getDeposit() {
        double deposit = product.getDeposit();
        return deposit > -1 ? String.valueOf(deposit) : ""; // Else return empty string
    }

    public static List<ProductAndPriceHelper> parseToProductAndPrice(Map<Product, Double>
        ↪ pricelist) {
        List<ProductAndPriceHelper> productAndPriceList = new LinkedList<>();
        for (Product product : pricelist.keySet()) {
            productAndPriceList.add(new ProductAndPriceHelper(product, pricelist.get(product
                ↪ )));
        }
        return productAndPriceList;
    }

    public static List<ProductAndPriceHelper> parseToProductAndPrice(List<Product>
        ↪ productList) {
        List<ProductAndPriceHelper> productAndPriceList = new LinkedList<>();
        for (Product product : productList) {
            productAndPriceList.add(new ProductAndPriceHelper(product, 0));
        }
        return productAndPriceList;
    }

    public static List<Product> getProductList(List<ProductAndPriceHelper> productAndPrices)
        ↪  {
        List<Product> products = new LinkedList<>();
        for (ProductAndPriceHelper productAndPrice : productAndPrices) {
            products.add(productAndPrice.getProduct());
        }
        return products;
    }

    public static void removeProductGroupFromList(List<ProductAndPriceHelper> list,
        ↪ ProductGroup productGroup) {
        list.removeIf(productAndPriceHelper -> productAndPriceHelper.getProductGroup().
            ↪ equals(productGroup));
    }
}
```

Chapter 10: gui/helpers/ProductDatePicker.java

```java
package gui.helpers;

import javafx.scene.control.DateCell;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Tooltip;
import javafx.util.Callback;
import javafx.util.StringConverter;
import model.Rent;
import model.Sale;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoField;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class ProductDatePicker extends DatePicker {
    Map<LocalDate, List<Rent>> dateSalesMap;

    public ProductDatePicker(List<Sale> sales) {
        this.showWeekNumbersProperty().setValue(true);
        dateSalesMap = new HashMap<>();

        for (Sale sale : sales) {
                LocalDate tourDate = ((Rent) sale).getDeliveryDateAndTime().toLocalDate();
                if (!dateSalesMap.containsKey(tourDate)){
                    dateSalesMap.put(tourDate, new LinkedList<>());
                }
                dateSalesMap.get(tourDate).add((Rent) sale);
        }
        this.setDayCellFactory(new Callback<DatePicker, DateCell>() {
            @Override
            public DateCell call(DatePicker param) {
                return new DateCell() {
                    @Override
                    public void updateItem(LocalDate item, boolean empty) {
                        super.updateItem(item, empty);
                        getStyleClass().add("tour-cell");
                        if (empty || item == null) {
                            setText(null);
                            setGraphic(null);
                        } else {
                            // If date is before today: disable
                            if (item.isBefore(LocalDate.now()))
```

```
                                setDisable(true);

                            DayOfWeek day = DayOfWeek.of(item.get(ChronoField.DAY_OF_WEEK));
                            if (dateSalesMap.containsKey(item)) { // if day has Tour
                                getStyleClass().add("occupied");
                                setText(getText());
                                setTooltip(new Tooltip(dateSalesMap.get(item).stream()
                                        .map(rent -> "- " + rent.getContactName() + " (" +
                                           ↪ rent.getContactInformation() + ")")
                                        .collect(Collectors.joining("\n"))));
                            } else if (day.equals(DayOfWeek.SATURDAY) || day.equals(
                               ↪ DayOfWeek.SUNDAY)) { // if weekend
                                getStyleClass().add("weekend");
                            } else { // else must be weekday
                                getStyleClass().add("weekday");
                            }
                        }
                    }
                };
            }
        });

        /*
         * This is purely to forcefully format date to European standard.
         */
        this.setConverter(new StringConverter<LocalDate>() {
            private DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("dd/MM
               ↪ /yyyy");

            @Override
            public String toString(LocalDate localDate) {
                return (localDate == null) ? "" : dateTimeFormatter.format(localDate);
            }

            @Override
            public LocalDate fromString(String string) {
                return (string == null || string.trim().isEmpty()) ? null : LocalDate.parse(
                   ↪ string, dateTimeFormatter);
            }
        });
    }


}
```

Chapter 11: gui/helpers/TableViewHelper.java

```java
package gui.helpers;

import controller.Controller;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import model.Pricelist;
import model.ProductGroup;
import model.product.Product;

public class TableViewHelper {

    private static Controller controller = Controller.getInstance();

    private static ObservableList<ProductGroup> productGroupObservableList = FXCollections.observableList(controller.getProductGroupList());
    private static ObservableList<Product> productObservableList = FXCollections.observableList(controller.getProductList());
    private static ObservableList<Pricelist> pricelistObservableList = FXCollections.observableList(controller.getPricelists());

    // -------------------------------------------------------------------------------------

    public static TableView<ProductGroup> createProductGroupTable() {
        TableView<ProductGroup> productGroupTableView = new TableView<>();
        TableColumn<ProductGroup, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("name"));
        productGroupTableView.getColumns().add(column1);
        productGroupTableView.setItems(productGroupObservableList);
        return productGroupTableView;
    }

    public static TableView<Product> createProductWithDescriptionTable() {
        TableView<Product> productTableView = new TableView<>();
        TableColumn<Product, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("name"));
        TableColumn<Product, String> column2 = new TableColumn<>("Produktgruppe");
        column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));
        TableColumn<Product, String> column3 = new TableColumn<>("Beskrivelse");
        column3.setCellValueFactory(new PropertyValueFactory<>("description"));
        productTableView.getColumns().add(column1);
        productTableView.getColumns().add(column2);
        productTableView.getColumns().add(column3);
        productTableView.setItems(productObservableList);
        return productTableView;
    }

    public static TableView<Pricelist> createPricelistTable() {
        TableView<Pricelist> pricelistTableView = new TableView<>();
        TableColumn<Pricelist, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("name"));
        pricelistTableView.getColumns().add(column1);
        pricelistTableView.setItems(pricelistObservableList);
        return pricelistTableView;
    }

    //
        ↪ ----------------------------------------------------------------------------
        ↪

    public static void addProductGroupToObservableList(ProductGroup productGroup) {
        productGroupObservableList.add(productGroup);
    }

    public static void addProductToObservableList(Product product) {
        productObservableList.add(product);
    }

    public static void addPricelistToObservableList(Pricelist pricelist) {
        pricelistObservableList.add(pricelist);
    }

    //
        ↪ ----------------------------------------------------------------------------
        ↪

    public static ObservableList<ProductGroup> getProductGroupObservableList() {
        return productGroupObservableList;
    }

}
```

Chapter 12: gui/helpers/TimePicker.java

```java
package gui.helpers;

import javafx.scene.control.Spinner;
import javafx.scene.control.SpinnerValueFactory;
import javafx.scene.layout.HBox;

import java.time.LocalTime;
import java.util.Arrays;

public class TimePicker extends HBox {
    private Spinner<Integer> sHours;
    private Spinner<Integer> sMinutes;

    public TimePicker() {
        sHours = new Spinner<Integer>();
        sHours.setValueFactory(new SpinnerValueFactory.IntegerSpinnerValueFactory(0,24, 12))
           ↪ ;

        sMinutes = new Spinner<Integer>();
        sMinutes.setValueFactory(new SpinnerValueFactory.IntegerSpinnerValueFactory(0,60,0)
           ↪ {
            @Override
            public void decrement(int steps) {
                if(getValue() == 0) {
                    sHours.decrement();
                    setValue(45);
                } else  {
                    setValue(getValue()-15);
                }

            }
            @Override
            public void increment(int steps) {
                if (getValue() == 60) {
                    sHours.increment();
                    setValue(0);
                } else {
                    setValue(getValue()+15);
                }
            }
        });
        this.getChildren().addAll(Arrays.asList(sHours, sMinutes));
    }

    public LocalTime getValue() {
       return LocalTime.of(sHours.getValue(),sMinutes.getValue());
    }
}
```

Chapter 13: gui/helpers/ValidationHelper.java

```java
package gui.helpers;

import javafx.scene.layout.Pane;

/**
 * For GUI validation
 */
public class ValidationHelper {

    /**
     * Removes CSS class attributes "error" from all children of pane
     *
     * @param pane to loop through
     */
    public static void removeErrorClassStyle(Pane pane) {
        pane.getChildren().stream().forEach(node -> {
            // Rekursivt kald
            if (node instanceof Pane)
                removeErrorClassStyle((Pane) node);

            node.getStyleClass().remove("error");
        });
    }

    /**
     * Checks whether a given string length is: - less than or equal to 30 - bigger
     * than 0 True if valid else false.
     *
     * @param string to be checked
     * @return boolean
     */
    public static boolean isStringBetween0to30characters(String string) {
        return string.matches(".+") || string.length() >= 30;
    }

    /**
     * Tests if string is a number and doesn't overflow double/integer max
     * @param string to be checked if number
     * @return true if number
     */
    public static boolean isNumber(String string) {
        return string.matches("\\d+") && string.length() < 9;
    }

    public static boolean isNumberAndPositive(String string) {
        return isNumber(string) && Double.parseDouble(string) >= 0;
    }

    /**
     *
     * @param string to be checked if percentage number
     * @return true if string represent a percentage number
     */
    public static boolean isPercentNumber(String string) {
        String subsubstring = string.substring(0, string.length() - 1);
        return string.lastIndexOf('%') == string.length() - 1 && isNumber(subsubstring)
                && Double.parseDouble(subsubstring) <= 100;
    }
}
```

Chapter 14: gui/HomePane.java

```java
package gui;

import gui.administrationpane.AdministrationPane;
import gui.overviewpane.OverviewPane;
import gui.images.LoadImage;
import gui.salepane.SaleOptionsPane;
import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Tooltip;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

import java.util.HashMap;
import java.util.Map;

public class HomePane extends BorderPane {

    private static HomePane homePane = getInstance(); // eager loaded

    private static final String IMAGE_FOLDER = "res/images/homepane";

    public HomePane() {
        Map<String, Image> imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);

        HBox hBoxButtons = new HBox();
        this.setCenter(hBoxButtons);
        hBoxButtons.setAlignment(Pos.CENTER);
        hBoxButtons.setSpacing(100);

        String[] btnNames = {"Oversigt", "Administration", "Salg", "Statistikker"};
        Map<String, Button> buttons = new HashMap<>();

        for (String btnName : btnNames) {
            Button button = new Button(btnName);
            buttons.put(btnName, button);
            button.setPrefHeight(100);
            button.setPrefWidth(100);
            button.setTooltip(new Tooltip(btnName));

            VBox vBox = new VBox();
            vBox.setSpacing(5);
            vBox.setAlignment(Pos.CENTER);
            vBox.getChildren().add(button);
            vBox.getChildren().add(new Label(btnName));

            hBoxButtons.getChildren().add(vBox);
            if (imageMap != null && imageMap.containsKey(btnName)) {
                ImageView imageView = new ImageView(imageMap.get(btnName));
                button.setGraphic(imageView);
                button.setText(""); // Remove text and use button instead
            }
        }

        buttons.get(btnNames[0]).setOnAction(event -> MainApp.changePane(OverviewPane.
           ↪ getInstance(), true));
        buttons.get(btnNames[1]).setOnAction(event -> MainApp.changePane(AdministrationPane.
           ↪ getInstance(), true));
        buttons.get(btnNames[2]).setOnAction(event -> MainApp.changePane(SaleOptionsPane.
           ↪ getInstance(), true));
        buttons.get(btnNames[3]).setOnAction(event -> MainApp.changePane(StatisticPane.
           ↪ getInstance(), true));
    }

    public static HomePane getInstance() {
        if (homePane == null)
            homePane = new HomePane();
        return homePane;
    }
}
```

Chapter 15: gui/LeftSideBarInterface.java

```java
package gui;

public interface LeftSideBarInterface {
    void changeSelected(String nameOfPane);
}
```

Chapter 16: gui/LeftSideBar.java

```java
package gui;

import javafx.geometry.Pos;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.text.TextAlignment;

import java.util.HashMap;
import java.util.Map;

public class LeftSideBar extends VBox {
    private Map<String, Pane> nameAndPaneMap;

    public LeftSideBar(Map<String, Pane> nameAndPaneMap, LeftSideBarInterface pane) {
        this.nameAndPaneMap = nameAndPaneMap;
        this.setAlignment(Pos.CENTER_LEFT);
        this.setSpacing(20);

        Map<String, ToggleButton> toggleButtons = new HashMap<>();
        ToggleGroup toggleGroup = new ToggleGroup();

        for (String nameOfButton : nameAndPaneMap.keySet()) {
            ToggleButton toggleButton = new ToggleButton(nameOfButton);
            toggleButtons.put(nameOfButton, toggleButton);
            toggleButton.setPrefHeight(75);
            toggleButton.setPrefWidth(75);
            toggleButton.setWrapText(true);
            toggleButton.setTextAlignment(TextAlignment.CENTER);
            toggleButton.setTooltip(new Tooltip(toggleButton.getText()));
            toggleButton.setToggleGroup(toggleGroup);
            this.getChildren().add(toggleButton);
        }

        int i = 0;
        for (String nameOfButton : toggleButtons.keySet()) {
            int finalI = i;
            toggleButtons.get(nameOfButton).setOnAction(event -> pane.changeSelected(
                ↪ nameOfButton));
            i++;
        }
    }

    public Pane getPane(String nameOfPane){
        return nameAndPaneMap.get(nameOfPane);
    }
}
```

Chapter 17: gui/MainApp.java

```java
package gui;

import controller.Controller;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

public class MainApp extends Application {

    private static final String TITLE = "Aarhus Bryghus - KASse System";
    private static final int HEIGHT = 720;
    private static final int WIDTH = HEIGHT * 16 / 9;

    private static BorderPane borderPane;

    public static void main(String[] args) {
        Application.launch();
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle(TITLE);

        borderPane = new BorderPane();
        this.initContent(borderPane);

        Scene scene = new Scene(borderPane, WIDTH, HEIGHT);
        scene.getStylesheets().add(getClass().getResource("css/app.css").toExternalForm());
        stage.setScene(scene);
        stage.setResizable(false); // Ikke sikkert skal sættes
        stage.show();
    }

    @Override
    public void stop() {
        Controller.getInstance().saveToFile("data/Serilizabledata.ser");
    }

    public void initContent(BorderPane borderPane) {
        Controller.getInstance().readFromFile("data/Serilizabledata.ser");
        borderPane.setTop(Navigation.getInstance());
        borderPane.setCenter(HomePane.getInstance()); // Set start pane
    }

    public static BorderPane getBorderPane() {
        return borderPane;
    }

    public static void changePane(Pane pane, boolean remember) {
        if (remember)
            Navigation.addPaneToBackList((Pane) borderPane.getCenter());
        Navigation.disableForwardButton();

        borderPane.setCenter(pane);

        Navigation.checkBackButton();
    }
}
```

Chapter 18: gui/Navigation.java

```java
package gui;

import gui.images.LoadImage;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ToolBar;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.util.Duration;

import java.time.LocalTime;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

/**
 * The top bar for the application.
 * Contains: home, back, forward and clock (etc).
 */
public class Navigation extends ToolBar {
    private static Navigation navigation = getInstance(); // eager loaded

    private static final String IMAGE_FOLDER = "res/images/toolbar";

    private static Button backButton;
    private static Button forwardButton;
    private static Label lblClock;

    private static List<Pane> backList = new LinkedList<>();
    private static List<Pane> forwardList = new LinkedList<>();

    private Navigation(){
       Map<String, Image> imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);
       BorderPane borderPane = MainApp.getBorderPane();

        Button homeButton = new Button();
        homeButton.setOnAction(event -> {
            if (borderPane.getCenter() != HomePane.getInstance())
                MainApp.changePane(HomePane.getInstance(), true);
        });
        if (imageMap != null && imageMap.containsKey("home")) {
            ImageView imageView = new ImageView(imageMap.get("home"));
            imageView.setFitWidth(20);
            imageView.setFitHeight(20);
            homeButton.setGraphic(imageView);
            homeButton.setText("");
        }

        backButton = new Button("<-");

        backButton.setDisable(true);
        backButton.setOnAction(event -> {
            forwardList.add((Pane) borderPane.getCenter());
            forwardButton.setDisable(false);
            int last = backList.size() - 1;
            borderPane.setCenter(backList.get(last));
            backList.remove(last);
            if (backList.isEmpty())
                backButton.setDisable(true);
        });
        if (imageMap != null && imageMap.containsKey("arrowLeft")) {
            ImageView imageView = new ImageView(imageMap.get("arrowLeft"));
            imageView.setFitWidth(20);
            imageView.setFitHeight(20);
            backButton.setGraphic(imageView);
            backButton.setText("");
        }

        forwardButton = new Button("->");

        forwardButton.setDisable(true);
        forwardButton.setOnAction(event -> {
            backList.add((Pane) borderPane.getCenter());
            backButton.setDisable(false);
            int last = forwardList.size() - 1;
            borderPane.setCenter(forwardList.get(last));
            forwardList.remove(last);
            if (forwardList.isEmpty())
                forwardButton.setDisable(true);
        });
        if (imageMap != null && imageMap.containsKey("arrowRight")) {
            ImageView imageView = new ImageView(imageMap.get("arrowRight"));
            imageView.setFitWidth(20);
            imageView.setFitHeight(20);
            forwardButton.setGraphic(imageView);
            forwardButton.setText("");
        }

        lblClock = new Label();
        lblClock.getStyleClass().add("clock");

        Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(1), ev -> {
            LocalTime time = LocalTime.now();
            String timeString = String.format("%s:%s:%s", time.getHour(),
                    time.getMinute() > 9 ? time.getMinute() : "0" + time.getMinute(),
                    time.getSecond() > 9 ? time.getSecond() : "0" + time.getSecond());
            lblClock.setText(timeString);
        }));
        timeline.setCycleCount(Animation.INDEFINITE);
        timeline.play();

        Region region = new Region();
        HBox.setHgrow(region, Priority.ALWAYS);
        this.getItems().addAll(Arrays.asList(homeButton, backButton, forwardButton, region,
            ↪ lblClock));
    }

    public static Navigation getInstance() {
        if (navigation == null)
            navigation = new Navigation();
        return navigation;
    }

    public static void addPaneToBackList(Pane pane) {
        backList.add(pane);
    }

    public static void removeLastBackPane() {
        int last = backList.size() - 1;
        backList.remove(last);
    }

    public static void disableForwardButton() {
        forwardButton.setDisable(true);
        forwardList.clear();
    }

    public static void checkBackButton() {
        if (!backList.isEmpty())
            backButton.setDisable(false);
    }
}
```

Chapter 19: gui/overviewpane/OverviewPane.java

```java
package gui.overviewpane;

import gui.LeftSideBar;
import gui.LeftSideBarInterface;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;

import java.util.LinkedHashMap;
import java.util.Map;

public class OverviewPane extends BorderPane implements LeftSideBarInterface {

    private static OverviewPane overviewPane;

    private LeftSideBar leftSideBar;

    public OverviewPane() {
        this.setPadding(new Insets(20));

        Label lblOverview = new Label("Oversigt");
        lblOverview.getStyleClass().add("title");
        this.setTop(lblOverview);

        Map<String, Pane> navigationMap = new LinkedHashMap<>();
        navigationMap.put("Produkter", ProductOverviewPane.getInstance());
        navigationMap.put("Prislister", PricelistOverviewPane.getInstance());
        navigationMap.put("Salg", SaleOverviewPane.getInstance());
        navigationMap.put("Udlejninger", RentOverviewPane.getInstance());
        navigationMap.put("Rundvisninger", TourOverviewPane.getInstance());
        leftSideBar = new LeftSideBar(navigationMap, this);
        setLeft(leftSideBar);

        changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
    }

    @Override
    public void changeSelected(String nameOfButton) {
        Label title = new Label(nameOfButton);
        title.getStyleClass().add("title");
        setAlignment(title, Pos.CENTER);
        setTop(title);
        setCenter(leftSideBar.getPane(nameOfButton));
    }

    public static OverviewPane getInstance() {
        if (overviewPane == null)
            overviewPane = new OverviewPane();
        return overviewPane;
    }
}
```

Chapter 20: gui/overviewpane/PricelistOverviewPane.java

```java
package gui.overviewpane;

import controller.Controller;
import gui.helpers.ProductAndPriceHelper;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeTableColumn;
import javafx.scene.control.TreeTableView;
import javafx.scene.control.cell.TreeItemPropertyValueFactory;
import javafx.scene.layout.GridPane;
import model.Pricelist;
import model.product.Product;

public class PricelistOverviewPane extends GridPane {

    private static PricelistOverviewPane pricelistOverviewPane;
    private TreeTableView<Object> pricelistTreeTable;
    private final int treeTableColIndex = 0;
    private final int treeTableRowIndex = 0;

    public PricelistOverviewPane() {
        getStyleClass().add("gridpane");

        pricelistTreeTable = getPricelistTreeTable();

        this.add(pricelistTreeTable, treeTableColIndex, treeTableRowIndex);
    }

    private TreeTableView<Object> getPricelistTreeTable() {
        TreeTableView<Object> treeTableView = new TreeTableView<>();
        treeTableView.setEditable(false);
        treeTableView.setPrefHeight(700);
        treeTableView.setPrefWidth(1100);
        treeTableView.setColumnResizePolicy(TreeTableView.CONSTRAINED_RESIZE_POLICY);


        TreeTableColumn<Object, String> column1 = new TreeTableColumn<>("Navn");
        column1.setCellValueFactory(new TreeItemPropertyValueFactory<>("name"));
        column1.setMinWidth(200);

        TreeTableColumn<Object, String> column2 = new TreeTableColumn<>("Pris");
        column2.setCellValueFactory(new TreeItemPropertyValueFactory<>("price"));
        column2.setMinWidth(50);

        TreeTableColumn<Object, String> column3 = new TreeTableColumn<>("Produktgruppe");
        column3.setCellValueFactory(new TreeItemPropertyValueFactory<>("productGroup"));

        treeTableView.getColumns().add(column1);
        treeTableView.getColumns().add(column2);
        treeTableView.getColumns().add(column3);

        TreeItem<Object> root = new TreeItem<>(new Object());
        treeTableView.setRoot(root);
        treeTableView.setShowRoot(false);

        for (Pricelist pricelist : Controller.getInstance().getPricelists()) {
            TreeItem<Object> pricelistTreeItem = new TreeItem<>(pricelist);
            pricelistTreeItem.setExpanded(false);
            root.getChildren().add(pricelistTreeItem);
            for (Product product : pricelist.getProductsWithPrice().keySet()) {
                ProductAndPriceHelper productAndPriceHelper = new ProductAndPriceHelper(
                    ↪ product, pricelist.getProductsWithPrice().get(product));
                if (pricelist.getProductsWithPrice().containsKey(product)) {
                    TreeItem<Object> productAndPriceHelperTreeItem = new TreeItem<>(
                        ↪ productAndPriceHelper);
                    pricelistTreeItem.getChildren().add(productAndPriceHelperTreeItem);
                }
            }
        }

        return treeTableView;
    }

    public void update() {
        pricelistTreeTable = getPricelistTreeTable();
        this.getChildren().remove(treeTableColIndex, treeTableRowIndex);
        this.add(pricelistTreeTable, treeTableColIndex, treeTableRowIndex);
    }

    public static PricelistOverviewPane getInstance() {
        if (pricelistOverviewPane == null)
            pricelistOverviewPane = new PricelistOverviewPane();
        return pricelistOverviewPane;
    }
}
```

Chapter 21: gui/overviewpane/ProductOverviewPane.java

```java
package gui.overviewpane;

import controller.Controller;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeTableColumn;
import javafx.scene.control.TreeTableView;
import javafx.scene.control.cell.TreeItemPropertyValueFactory;
import javafx.scene.layout.GridPane;
import model.ProductGroup;
import model.enums.Unit;
import model.product.Bundle;
import model.product.Container;
import model.product.Product;
import model.product.rentable.Rentable;

public class ProductOverviewPane extends GridPane {

    private static ProductOverviewPane productOverviewPane;
    private TreeTableView<Object> productOverviewTreeTable;
    private final int treeTableColIndex = 0;
    private final int treeTableRowIndex = 0;

    public ProductOverviewPane() {
        getStyleClass().add("gridpane");

        productOverviewTreeTable = getProductOverviewTreeTable();
        this.add(productOverviewTreeTable, treeTableColIndex, treeTableRowIndex);
    }

    private TreeTableView<Object> getProductOverviewTreeTable() {
        TreeTableView<Object> treeTableView = new TreeTableView<>();
        treeTableView.setEditable(false);
        treeTableView.setPrefHeight(700);
        treeTableView.setPrefWidth(1100);
        treeTableView.setColumnResizePolicy(TreeTableView.CONSTRAINED_RESIZE_POLICY);

        TreeTableColumn<Object, String> column1 = new TreeTableColumn<>("Navn");
        column1.setCellValueFactory(new TreeItemPropertyValueFactory<>("name"));
        column1.setMinWidth(200);

        TreeTableColumn<Object, String> column2 = new TreeTableColumn<>("ID");
        column2.setCellValueFactory(new TreeItemPropertyValueFactory<>("id"));

        TreeTableColumn<Object, String> column3 = new TreeTableColumn<>("Beskrivelse");
        column3.setCellValueFactory(new TreeItemPropertyValueFactory<>("description"));
        column3.setMinWidth(300);

        TreeTableColumn<Object, Boolean> column4 = new TreeTableColumn<>("Udlejes");
        column4.setCellValueFactory(new TreeItemPropertyValueFactory<>("rentable"));

        TreeTableColumn<Object, String> column5 = new TreeTableColumn<>("Udlejningsafgift");
        column5.setCellValueFactory(new TreeItemPropertyValueFactory<>("productDeposit"));

        TreeTableColumn<Object, String> column6 = new TreeTableColumn<>("Størrelse");
        column6.setCellValueFactory(new TreeItemPropertyValueFactory<>("size"));

        TreeTableColumn<Object, String> column7 = new TreeTableColumn<>("Enhed");
        column7.setCellValueFactory(new TreeItemPropertyValueFactory<>("unit"));

        TreeTableColumn<Object, String> column8 = new TreeTableColumn<>("Sampakningsindhold"
            ↪ );
        column8.setCellValueFactory(new TreeItemPropertyValueFactory<>("bundleList"));

        treeTableView.getColumns().add(column1);
        treeTableView.getColumns().add(column2);
        treeTableView.getColumns().add(column3);
        treeTableView.getColumns().add(column4);
        treeTableView.getColumns().add(column5);
        treeTableView.getColumns().add(column6);
        treeTableView.getColumns().add(column7);
        treeTableView.getColumns().add(column8);

        TreeItem<Object> root = new TreeItem<>(new Object());
        treeTableView.setRoot(root);
        treeTableView.setShowRoot(false);

        for (ProductGroup productGroup : Controller.getInstance().getProductGroupList()) {

            TreeItem<Object> productGroupTreeItem = new TreeItem<>(productGroup);
            productGroupTreeItem.setExpanded(true);
            root.getChildren().add(productGroupTreeItem);

            for (Product product : Controller.getInstance().getProductList()) {

                ProductExtended productExtended;
                if (product instanceof Container) {
                    productExtended = new ProductExtended(product, ((Container) product).
                        ↪ getSize(),
                            ((Container) product).getUnit());
                } else if (product instanceof Bundle) {
                    productExtended = new ProductExtended(product, product.toString());
                } else {
                    productExtended = new ProductExtended(product);
                }

                if (product.getProductGroup().equals(productGroup)) {
                    TreeItem<Object> productExtendedTreeItem = new TreeItem<>(
                        ↪ productExtended);
                    productGroupTreeItem.getChildren().add(productExtendedTreeItem);
                }
            }
        }
        return treeTableView;
    }

    public void update() {
        productOverviewTreeTable = getProductOverviewTreeTable();
        this.getChildren().remove(treeTableColIndex, treeTableRowIndex);
        this.add(productOverviewTreeTable, treeTableColIndex, treeTableRowIndex);
    }

    public static ProductOverviewPane getInstance() {
        if (productOverviewPane == null)
            productOverviewPane = new ProductOverviewPane();
        return productOverviewPane;
    }

    //
        ↪ ------------------------------------------------------------------------------------
        ↪

    /*
     * Private class to display table Not pretty, but does the job - Cannot extend
     * since we use iD's
     */

    protected static class ProductExtended {

        private Product product;

        private double size;
        private Unit unit;
        private String bundleList;

        public ProductExtended(Product product) {
            this.product = product;
            this.size = -1;
        }

        public ProductExtended(Product product, double size, Unit unit) {
            this.product = product;
            this.unit = unit;
            this.size = size;
        }

        public ProductExtended(Product product, String bundleList) {
            this.product = product;
            this.size = -1;
            this.bundleList = bundleList;
        }

        public String getName() {
            return product.getName();
        }

        public int getId() {
            return product.getId();
        }

        public String getDescription() {
            return product.getDescription();
        }

        public boolean getRentable() {
            return product.getRentableStrategy() instanceof Rentable;
        }

        public String getProductDeposit() {
            return getRentable() ? String.valueOf(product.getRentableStrategy().getDeposit()
                ↪ ) : "";
        }

        public String getSize() {
            return size > -1 ? String.valueOf(size) : "";
        }

        public String getUnit() {
            return unit != null ? unit.toString() : "";
        }

        public String getBundleList() {
            return bundleList;
        }
    }
}
```

Chapter 22: gui/overviewpane/RentOverviewPane.java

```java
package gui.overviewpane;

import controller.Controller;
import javafx.beans.property.SimpleStringProperty;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.GridPane;
import model.Rent;
import model.Sale;
import model.SalesLine;
import model.product.Tour;

import java.time.LocalDateTime;
import java.util.List;

public class RentOverviewPane extends GridPane {

    private static RentOverviewPane rentOverviewPane;
    private TableView<Object> rentOverviewTable;
    private TableView<Object> rentSalesLineTable;

    public RentOverviewPane() {
        getStyleClass().add("gridpane");

        rentOverviewTable = getRentOverviewTable();
        update();
        rentOverviewTable.getSelectionModel().selectedItemProperty().addListener((obs,
           ↪ oldSelection, newSelection) -> {
            if (newSelection != null) {
                updateSalesLineTable();
            }
        });
        this.add(rentOverviewTable, 0, 0, 1, 2);

        Label lblProductsInRent = new Label("Produkter i udlejning:");
        this.add(lblProductsInRent, 1, 0);
        rentSalesLineTable = getRentSaleLinesTable();
        this.add(rentSalesLineTable, 1, 1);

        Button btnUpdate = new Button("OpdatÃl’r");
        btnUpdate.setOnAction(e -> this.update());
        this.add(btnUpdate, 0, 4);

    }

    private TableView<Object> getRentOverviewTable() {
        TableView<Object> tableView = new TableView<>();
        tableView.setEditable(false);
        tableView.setPrefHeight(700);
        tableView.setPrefWidth(800);
        tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);

        TableColumn<Object, String> column1 = new TableColumn<>("Tid");
        column1.setCellValueFactory(new PropertyValueFactory<>("timestamp"));

        TableColumn<Object, String> column2 = new TableColumn<>("ID");
        column2.setCellValueFactory(new PropertyValueFactory<>("id"));

        TableColumn<Object, String> column3 = new TableColumn<>("Status");
        column3.setCellValueFactory(new PropertyValueFactory<>("saleState"));

        TableColumn<Object, String> column4 = new TableColumn<>("Kontakt navn");
        column4.setCellValueFactory(new PropertyValueFactory<>("contactName"));

        TableColumn<Object, String> column5 = new TableColumn<>("Kontakt information");
        column5.setCellValueFactory(new PropertyValueFactory<>("contactInformation"));

        TableColumn<Object, String> column6 = new TableColumn<>("Leveringstidspunkt");
        column6.setCellValueFactory(new PropertyValueFactory<>("deliveryDateAndTime"));

        TableColumn<Object, String> column7 = new TableColumn<>("Returtidspunkt");
        column7.setCellValueFactory(new PropertyValueFactory<>("returnDateAndTime"));

        tableView.getColumns().add(column1);
        tableView.getColumns().add(column2);
        tableView.getColumns().add(column3);
        tableView.getColumns().add(column4);
        tableView.getColumns().add(column5);
        tableView.getColumns().add(column6);
        tableView.getColumns().add(column7);

        return tableView;
    }

    private TableView<Object> getRentSaleLinesTable() {
        TableView<Object> tableView = new TableView<>();
        tableView.setEditable(false);
        tableView.setPrefHeight(700);
        tableView.setPrefWidth(280);
        tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);

        TableColumn<Object, String> column1 = new TableColumn<>("Produkt");
        column1.setCellValueFactory(new PropertyValueFactory<>("product"));

        TableColumn<Object, String> column2 = new TableColumn<>("Produktgruppe");
        column2.setCellValueFactory(
                cellData -> {
                    SalesLine salesLine = ((SalesLine) cellData.getValue());
                    return new SimpleStringProperty(salesLine.getProduct().getProductGroup()
                        ↪ .getName());
                });

        TableColumn<Object, String> column3 = new TableColumn<>("Antal");
```

```java
        column3.setCellValueFactory(new PropertyValueFactory<>("quantity"));

        tableView.getColumns().add(column1);
        tableView.getColumns().add(column2);
        tableView.getColumns().add(column3);

        return tableView;
    }

    private void updateSalesLineTable() {
        RentExtended rentExtended = (RentExtended) rentOverviewTable.getSelectionModel().getSelectedItem();
        rentSalesLineTable.getItems().setAll(rentExtended.getSalesLines());
    }

    public void update() {
        rentOverviewTable.getItems().clear();
        for (Sale sale : Controller.getInstance().getSales()) {
            if (sale instanceof Rent) {
                if (sale.getSalesLines().stream().anyMatch(salesLine -> !(salesLine.getProduct() instanceof Tour))) {
                    RentExtended rentExtended = new RentExtended((Rent) sale);
                    rentOverviewTable.getItems().add(rentExtended);
                }
            }
        }
    }

    public static RentOverviewPane getInstance() {
        if (rentOverviewPane == null)
            rentOverviewPane = new RentOverviewPane();
        return rentOverviewPane;
    }

    // --------------------------------------------------------------------------------

    /*
     * Private class to display table Not pretty, but does the job
     */

    protected static class RentExtended {

        private Rent rent;

        public RentExtended(Rent rent) {
            this.rent = rent;
        }

        public int getId() {
            return rent.getId();
        }

        public String getSaleState() {
            return rent.getSaleState().toString();
        }

        public List<SalesLine> getSalesLines() {
            return rent.getSalesLines();
        }

        public String getTimestamp() {
            LocalDateTime time = rent.getTimestamp();
            return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.
               ↪ getMonthValue(), time.getYear(), time.getHour(), time.getMinute());
        }

        public String getContactName() {
            return rent.getContactName();
        }

        public String getContactInformation() {
            return rent.getContactInformation();
        }

        public String getDeliveryDateAndTime() {
            LocalDateTime time = rent.getDeliveryDateAndTime();
            return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.
               ↪ getMonthValue(), time.getYear(), time.getHour(), time.getMinute());
        }

        public String getReturnDateAndTime() {
            LocalDateTime time = rent.getReturnDateAndTime();
            return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.
               ↪ getMonthValue(), time.getYear(), time.getHour(), time.getMinute());
        }
    }
}
```

Chapter 23: gui/overviewpane/SaleOverviewPane.java

```java
package gui.overviewpane;

import controller.Controller;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.GridPane;
import model.Rent;
import model.Sale;

import java.time.LocalDateTime;

public class SaleOverviewPane extends GridPane {

        private static SaleOverviewPane saleOverviewPane;
        private TableView<Object> saleOverviewTable;

        public SaleOverviewPane() {
                getStyleClass().add("gridpane");
                saleOverviewTable = getSaleOverviewTable();
                update(); // init table with content
                this.add(saleOverviewTable, 0, 0);
        }

        private TableView<Object> getSaleOverviewTable() {
                TableView<Object> tableView = new TableView<>();
                tableView.setEditable(false);
                tableView.setPrefHeight(700);
                tableView.setPrefWidth(1100);
                tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);

                TableColumn<Object, String> column1 = new TableColumn<>("Tid");
                column1.setCellValueFactory(new PropertyValueFactory<>("timestamp"));

                TableColumn<Object, String> column2 = new TableColumn<>("ID");
                column2.setCellValueFactory(new PropertyValueFactory<>("id"));

                TableColumn<Object, String> column3 = new TableColumn<>("Status");
                column3.setCellValueFactory(new PropertyValueFactory<>("saleState"));

                TableColumn<Object, String> column4 = new TableColumn<>("Udlejning");
                column4.setCellValueFactory(new PropertyValueFactory<>("isRent"));

                tableView.getColumns().add(column1);
                tableView.getColumns().add(column2);
                tableView.getColumns().add(column3);
                tableView.getColumns().add(column4);

```

```java

            return tableView;
    }

    public void update() {
        saleOverviewTable.getItems().clear();
            for (Sale sale : Controller.getInstance().getSales()) {
                SaleExtended saleExtended = new SaleExtended(sale);
                saleOverviewTable.getItems().add(saleExtended);
            }
    }

    public static SaleOverviewPane getInstance() {
        if (saleOverviewPane == null)
            saleOverviewPane = new SaleOverviewPane();
        return saleOverviewPane;
    }

    // ------------------------------------------------------------------------------------------

    /*
     * Private class to display table Not pretty, but does the job
     */

    protected static class SaleExtended {

        private Sale sale;

        public SaleExtended(Sale sale) {
            this.sale = sale;
        }

        public int getId() {
            return sale.getId();
        }

        public String getSaleState() {
            return sale.getSaleState().toString();
        }

        public String getTimestamp() {
            LocalDateTime time = sale.getTimestamp();
            return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.getMonthValue(), time.getYear(),
                    time.getHour(), time.getMinute());
        }

        public String getIsRent() {
            return sale instanceof Rent ? "x" : "";
        }

    }

}
```

Chapter 24: gui/overviewpane/TourOverviewPane.java

```java
package gui.overviewpane;

import controller.Controller;
import gui.helpers.Calendar;
import javafx.beans.property.SimpleStringProperty;
import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import model.Rent;
import model.Sale;

import java.util.List;

public class TourOverviewPane extends GridPane {

    private static TourOverviewPane tourOverviewPane;
    private TableView<Sale> tourOverviewTable;
    private VBox vbCalendar;

    public TourOverviewPane() {
        getStyleClass().add("gridpane");

        tourOverviewTable = getTourOverviewTable();
        this.add(tourOverviewTable, 0, 0);

        vbCalendar = new VBox();
        vbCalendar.setAlignment(Pos.CENTER);
        this.add(vbCalendar, 1, 0);

        Button btnUpdate = new Button("OpdatÃl'r");
        btnUpdate.setOnAction(e -> this.update());
        this.add(btnUpdate, 0, 1);
        update(); // init table with content
    }

    public TableView<Sale> getTourOverviewTable() {
        TableView<Sale> tableView = new TableView<>();
        tableView.setEditable(false);
        tableView.setPrefHeight(700);
        tableView.setPrefWidth(800);
        tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        TableColumn<Sale, String> column1 = new TableColumn<>("Tid");
        column1.setCellValueFactory(new PropertyValueFactory<>("timestamp"));

        TableColumn<Sale, String> column2 = new TableColumn<>("Salgs ID");
        column2.setCellValueFactory(new PropertyValueFactory<>("id"));

        TableColumn<Sale, String> column3 = new TableColumn<>("Antal");
        column3.setCellValueFactory(
                cellData -> new SimpleStringProperty(String.valueOf(cellData.getValue().
                    ↪ getSalesLines()
                        .stream().mapToInt(salesLine -> salesLine.getQuantity())
                        .sum())));

        TableColumn<Sale, String> column4 = new TableColumn<>("Beskrivelse");
        column4.setCellValueFactory(new PropertyValueFactory<>("description"));

        TableColumn<Sale, String> column5 = new TableColumn<>("Kontaktperson");
        column5.setCellValueFactory(new PropertyValueFactory<>("contactName"));

        TableColumn<Sale, String> column6 = new TableColumn<>("Kontaktperson");
        column6.setCellValueFactory(new PropertyValueFactory<>("contactInformation"));

        TableColumn<Sale, String> column7 = new TableColumn<>("Rundvisningstidspunkt");
        column7.setCellValueFactory(
                cellData -> new SimpleStringProperty(
                        ((Rent) cellData.getValue()).getDeliveryDateAndTime().toLocalDate().
                            ↪ toString()));

        tableView.getColumns().add(column1);
        tableView.getColumns().add(column2);
        tableView.getColumns().add(column3);
        tableView.getColumns().add(column4);
        tableView.getColumns().add(column5);
        tableView.getColumns().add(column6);
        tableView.getColumns().add(column7);

        return tableView;
    }

    public void update() {
        List<Sale> tourSale = Controller.getInstance().getTours();
        // refreshes the calendar by removing and adding a new
        vbCalendar.getChildren().clear();
        vbCalendar.getChildren().add(new Calendar(tourSale));
        tourOverviewTable.getItems().clear();
        for (Sale sale : tourSale) {
            tourOverviewTable.getItems().add(sale);
        }
    }

    public static TourOverviewPane getInstance() {
        if (tourOverviewPane == null)
            tourOverviewPane = new TourOverviewPane();
        return tourOverviewPane;
    }
}
```

Chapter 25: gui/salepane/PaymentPane.java

```java
package gui.salepane;

import controller.Controller;
import gui.MainApp;
import gui.Navigation;
import gui.images.LoadImage;
import gui.overviewpane.RentOverviewPane;
import gui.overviewpane.SaleOverviewPane;
import gui.overviewpane.TourOverviewPane;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import model.Pricelist;
import model.Rent;
import model.Sale;
import model.SalesLine;
import model.enums.PaymentMethod;
import model.enums.SaleState;

import java.time.LocalDateTime;
import java.util.Map;

public class PaymentPane extends GridPane {

    private Sale sale;
    private Pricelist pricelist;

    private static final String IMAGE_FOLDER = "res/images/paymentpane";

    private TextField txfAmount;
    private ToggleGroup toggleGroup;
    private Label lblMissingPaymentAmount;
    private Button btnPayDelay;
    private Button btnPayComplete;

    private TableView<PaymentMethodAndAmount> transfersTableView;

    private Label lblFinalPriceAmount;

    public PaymentPane(Sale sale, Pricelist pricelist, String discount, String
        ↪ finalPriceAmount) {
```

```java
        this.getStyleClass().add("gridpane");

        this.sale = sale;
        this.pricelist = pricelist;

        Label lblSaleLine = new Label("Salgslinje:");
        lblSaleLine.setFont(new Font(16));
        this.add(lblSaleLine, 3, 0);

        // Table 2
        TableViewSalesLines tableViewSalesLines = new TableViewSalesLines(sale);
        this.add(tableViewSalesLines, 3, 1, 2, 4);

        Label lblRentPrice = new Label("Udlejnings pris:");
        this.add(lblRentPrice, 3, 5);
        lblRentPrice.setMinWidth(200);

        Label lblRentPriceAmount = new Label("0.0");
        double totalDeposit = 0d;
        for (SalesLine salesLine : sale.getSalesLines()) {
            totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
                ↪ getTotalLineDeposit() : 0;
        }
        lblRentPriceAmount.setText(String.valueOf(totalDeposit));
        setHalignment(lblRentPriceAmount, HPos.RIGHT);
        this.add(lblRentPriceAmount, 4, 5);

        Label lblTotalPrice = new Label("Samlet pris:");
        this.add(lblTotalPrice, 3, 6);
        lblTotalPrice.setMinWidth(200);

        Label lblTotalPriceAmount = new Label();
        setHalignment(lblTotalPriceAmount, HPos.RIGHT);
        lblTotalPriceAmount.setText(String.valueOf(sale.totalSalePrice()));
        this.add(lblTotalPriceAmount, 4, 6);

        Label lblDiscount = new Label("Rabat:");
        this.add(lblDiscount, 3, 7);

        Label lblDiscountValue = new Label(discount);
        this.add(lblDiscountValue, 4, 7);
        lblDiscountValue.setMinWidth(100);
        lblDiscountValue.setMaxWidth(100);
        lblDiscountValue.setAlignment(Pos.CENTER_RIGHT);
        setHalignment(lblDiscountValue, HPos.RIGHT);

        Label txfFinalPrice = new Label("Slut pris:");
        this.add(txfFinalPrice, 3, 8);

        lblFinalPriceAmount = new Label();
        lblFinalPriceAmount.setText(finalPriceAmount);
        this.add(lblFinalPriceAmount, 4, 8);
        setHalignment(lblFinalPriceAmount, HPos.RIGHT);

        // Placeholder for layout
        Button btnPlaceholder = new Button("Placeholder");
        this.add(btnPlaceholder, 3, 9, 2, 1);
        setHalignment(btnPlaceholder, HPos.CENTER);
        btnPlaceholder.setDisable(true);
        btnPlaceholder.setVisible(false);

        Label lblPayment = new Label("Betaling:");
        lblPayment.setFont(new Font(16));
        lblPayment.setPrefWidth(800);
        this.add(lblPayment, 0, 0, 3, 1);

        VBox vBox = new VBox();
        this.add(vBox, 0, 1, 1, 4);
        vBox.setAlignment(Pos.CENTER_LEFT);
        vBox.setSpacing(5);

        Label lblPaymentMethod = new Label("Vælg betalingsmetode:");
        vBox.getChildren().add(lblPaymentMethod);

        Map<String, Image> imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);
        toggleGroup = new ToggleGroup();

        for (PaymentMethod paymentMethod : PaymentMethod.values()) {
            ToggleButton toggleButton = new ToggleButton(paymentMethod.toString());
            toggleButton.setUserData(paymentMethod);
            toggleButton.setToggleGroup(toggleGroup);
            toggleButton.setContentDisplay(ContentDisplay.LEFT);
            toggleGroup.selectToggle(toggleButton); // Select last one added
            vBox.getChildren().add(toggleButton);

            String paymentMethodName = paymentMethod.toString();
            if (imageMap != null && imageMap.containsKey(paymentMethodName)) {
                ImageView imageView = new ImageView(imageMap.get(paymentMethodName));
                imageView.setFitHeight(20);
                imageView.setFitWidth(20);
                toggleButton.setGraphic(imageView);
            }
        }

        Label lblAmount = new Label("Beløb:");
        vBox.getChildren().add(lblAmount);

        String suggestedPayment = String
                .valueOf(Double.parseDouble(lblFinalPriceAmount.getText()) - sale.getPayment
                    ↪ ().calcTotal());
        txfAmount = new TextField(suggestedPayment);
        txfAmount.setMaxWidth(120);
        vBox.getChildren().add(txfAmount);

        Button btnAddTransfer = new Button("Tilføj");
        vBox.getChildren().add(btnAddTransfer);
        btnAddTransfer.setOnAction(event -> addTransfer());

        // Table 1
        transfersTableView = new TableView<>();
        this.add(transfersTableView, 1, 1, 2, 4);
        transfersTableView.setPrefWidth(660);
        transfersTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        transfersTableView.setEditable(false);
        createTransfersTableView();

        Label lblMissingPayment = new Label("Manglende beløb til betaling:");
        this.add(lblMissingPayment, 1, 5);

        lblMissingPaymentAmount = new Label();
        if (sale instanceof Rent && !sale.getSaleState().equals(SaleState.DELAYED)) {
            totalDeposit = 0d;
            for (SalesLine salesLine : sale.getSalesLines()) {
                totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
                    ↪ getTotalLineDeposit() : 0;
            }
            lblMissingPaymentAmount.setText(String.valueOf(totalDeposit));
            txfAmount.setText(String.valueOf(totalDeposit));
        } else {
            lblMissingPaymentAmount.setText(suggestedPayment);
        }
        this.add(lblMissingPaymentAmount, 2, 5);
        setHalignment(lblMissingPaymentAmount, HPos.RIGHT);

        HBox hBox = new HBox();
        this.add(hBox, 0, 7, 3, 1);
        hBox.setSpacing(100);
        hBox.setAlignment(Pos.CENTER);

        Button btnDelayPayment = new Button("Udskyd betaling");
        hBox.getChildren().add(btnDelayPayment);
        btnDelayPayment.setOnAction(event -> finishSale(SaleState.DELAYED));

        btnPayDelay = new Button("Betal (del)");
        hBox.getChildren().add(btnPayDelay);
        btnPayDelay.setDisable(true);
        btnPayDelay.setOnAction(event -> {
            finishSale(SaleState.DELAYED);
            SelectOngoingSale.getInstance().refreshOngoingSales();
        });

        btnPayComplete = new Button("Betal (afslut)");
        hBox.getChildren().add(btnPayComplete);
        btnPayComplete.setDisable(true);
        btnPayComplete.setOnAction(event -> {
            finishSale(SaleState.COMPLETED);

            SelectOngoingSale.getInstance().refreshOngoingSales();
        });
    }

    public void addTransfer() {
        PaymentMethod paymentMethod = (PaymentMethod) toggleGroup.getSelectedToggle().
            ↪ getUserData();
        double amount = Double.parseDouble(txfAmount.getText());

        sale.addTransfer(paymentMethod, amount);

        PaymentMethodAndAmount paymentMethodAndAmount = new PaymentMethodAndAmount(
            ↪ paymentMethod, amount);

        transfersTableView.getItems().add(paymentMethodAndAmount);

        double missingPaymentAmount = Double.parseDouble(lblMissingPaymentAmount.getText())
            ↪ - amount;
        lblMissingPaymentAmount.setText(String.valueOf(missingPaymentAmount));

        if (missingPaymentAmount <= 0.0)
            btnPayDelay.setDisable(false);
        if (sale.getPayment().calcTotal() >= Double.parseDouble(lblFinalPriceAmount.getText
            ↪ ()))
            btnPayComplete.setDisable(false);

    }

    public void createTransfersTableView() {
        TableColumn<PaymentMethodAndAmount, String> column1 = new TableColumn<>("
            ↪ Betalingsmetode");
        column1.setCellValueFactory(new PropertyValueFactory<>("paymentMethod"));
        column1.setStyle("-fx-alignment: CENTER-LEFT");

        TableColumn<PaymentMethodAndAmount, Double> column2 = new TableColumn<>("Beløb");
        column2.setCellValueFactory(new PropertyValueFactory<>("amount"));
        column2.setStyle("-fx-alignment: CENTER-RIGHT");

        transfersTableView.getColumns().add(column1);
        transfersTableView.getColumns().add(column2);

        for (PaymentMethod paymentMethod : sale.getPayment().getTransfers().keySet()) {
            transfersTableView.getItems().add(
                    new PaymentMethodAndAmount(paymentMethod, sale.getPayment().getTransfers
                        ↪ ().get(paymentMethod)));
        }
    }

    private void finishSale(SaleState saleState) {
        if (Controller.getInstance().getSales().contains(sale)) {
            sale.setAgreedPrice(Double.parseDouble(lblFinalPriceAmount.getText()));
            sale.setSaleState(saleState);
            return; // Ikke opret salg hvis allerede eksisterer
        }
        Sale savedSale;
        if (sale instanceof Rent) {
            String contactName = ((Rent) sale).getContactName();
            String contactInformation = ((Rent) sale).getContactInformation();
            LocalDateTime deliveryDateAndTime = ((Rent) sale).getDeliveryDateAndTime();
            LocalDateTime returnDateAndTime = ((Rent) sale).getReturnDateAndTime();
            savedSale = Controller.getInstance().createRent(contactName, contactInformation,
                ↪  deliveryDateAndTime,
                    returnDateAndTime);
            RentOverviewPane.getInstance().update();
            TourOverviewPane.getInstance().update();
        } else {
            savedSale = Controller.getInstance().createSale();
            SaleOverviewPane.getInstance().update();
        }
        savedSale.setAgreedPrice(Double.parseDouble(lblFinalPriceAmount.getText()));
        savedSale.setSaleState(saleState);
        for (SalesLine salesLine : sale.getSalesLines()) {
            savedSale.updateSalesLine(salesLine.getProduct(), salesLine.getQuantity(),
                    salesLine.getTotalLinePrice() / salesLine.getQuantity());
```

```java
        }
        for (PaymentMethod paymentMethod : sale.getPayment().getTransfers().keySet()) {
            savedSale.getPayment().addTransfer(paymentMethod, sale.getPayment().getTransfers().get(paymentMethod));
        }

        sale = null;

        Navigation.removeLastBackPane();
        MainApp.changePane(new SalePane(null, pricelist), false);
        // 'false'
    }

    // ------------------------------------------------------------------------------------

    /*
     * Class to enable gui (Cannot be private)
     */
    protected static class PaymentMethodAndAmount {

        private PaymentMethod paymentMethod;
        private double amount;

        public PaymentMethodAndAmount(PaymentMethod paymentMethod, double amount) {
            this.paymentMethod = paymentMethod;
            this.amount = amount;
        }

        public PaymentMethod getPaymentMethod() {
            return paymentMethod;
        }

        public double getAmount() {
            return amount;
        }
    }

}
```

Chapter 26: gui/salepane/RentInformationPane.java

```java
package gui.salepane;

import controller.Controller;
import gui.MainApp;
import gui.helpers.ProductDatePicker;
import gui.helpers.ValidationHelper;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Font;
import model.Pricelist;
import model.Rent;
import model.Sale;
import model.SalesLine;

import java.time.LocalDateTime;

public class RentInformationPane extends GridPane {

    private Sale sale;
    private Pricelist pricelist;
    private String discount;
    private boolean tour;

    private Label lblFinalPriceAmount;

    private TextField txfContactName;
    private TextArea txaContactInformation;
    private DatePicker dpDeliveryDateAndTime;
    private DatePicker dpReturnDateAndTime;
    private ProductDatePicker productDatePicker;

    public RentInformationPane(Sale sale, Pricelist pricelist, String discount, String
        ↪ finalPriceAmount, boolean tour) {
        this.getStyleClass().add("gridpane");

        this.sale = sale;
        this.pricelist = pricelist;
        this.discount = discount;
        this.tour = tour;

        Label lblRentInformation = new Label("Kontakt information:");
        lblRentInformation.setFont(new Font(16));
        lblRentInformation.setPrefWidth(800);
        this.add(lblRentInformation, 0, 0, 3, 1);

        //
        Label lblContactName = new Label("Kontakt navn:");
        this.add(lblContactName, 0, 1);
        txfContactName = new TextField();
        this.add(txfContactName, 1, 1);

        Label lblContactInformation = new Label("Kontakt information:");
        setValignment(lblContactInformation, VPos.TOP);
        this.add(lblContactInformation, 0, 2);
        txaContactInformation = new TextArea();
        txaContactInformation.setPrefColumnCount(3);
        this.add(txaContactInformation, 1, 2);

        if (!tour) {
            Label lblDeliveryDateAndTime = new Label("Afleverings tidspunkt:");
            this.add(lblDeliveryDateAndTime, 0, 3);
            dpDeliveryDateAndTime = new DatePicker();
            this.add(dpDeliveryDateAndTime, 1, 3);

            Label lblReturnDateAndTime = new Label("Returnerings tidspunk:");
            this.add(lblReturnDateAndTime, 0, 4);
            dpReturnDateAndTime = new DatePicker();
            this.add(dpReturnDateAndTime, 1, 4);
        } else {
            Label lblReturnDateAndTime = new Label("Rundvisnings tidspunkt:");
            this.add(lblReturnDateAndTime, 0, 3);
            productDatePicker = new ProductDatePicker(Controller.getInstance().getRents());
            this.add(productDatePicker, 1, 3);
        }

        Button btnDelayPayment = new Button("Betaling");
        this.add(btnDelayPayment, 0, 7, 3, 1);
        setHalignment(btnDelayPayment, HPos.CENTER);
        btnDelayPayment.setOnAction(event -> goToPayment());

        Label lblSaleLine = new Label("Salgslinje:");
        lblSaleLine.setFont(new Font(16));
        this.add(lblSaleLine, 3, 0);

        // Table 2
        TableViewSalesLines salesLineTableView = new TableViewSalesLines(sale);
        this.add(salesLineTableView, 3, 1, 2, 4);

        Label lblRentPrice = new Label("Udlejnings pris:");
        this.add(lblRentPrice, 3, 5);
        lblRentPrice.setMinWidth(200);

        Label lblRentPriceAmount = new Label("0.0");
        double totalDeposit = 0d;
        for (SalesLine salesLine : sale.getSalesLines()) {
            totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
                ↪ getTotalLineDeposit() : 0;
        }
        lblRentPriceAmount.setText(String.valueOf(totalDeposit));
        setHalignment(lblRentPriceAmount, HPos.RIGHT);
        this.add(lblRentPriceAmount, 4, 5);

        Label lblTotalPrice = new Label("Samlet pris:");
        this.add(lblTotalPrice, 3, 6);
        lblTotalPrice.setMinWidth(200);

        Label lblTotalPriceAmount = new Label();
        setHalignment(lblTotalPriceAmount, HPos.RIGHT);
        lblTotalPriceAmount.setText(String.valueOf(sale.totalSalePrice()));
        this.add(lblTotalPriceAmount, 4, 6);

        Label lblDiscount = new Label("Rabat:");
        this.add(lblDiscount, 3, 7);

        Label lblDiscountValue = new Label(discount);
        this.add(lblDiscountValue, 4, 7);
        lblDiscountValue.setMinWidth(100);
        lblDiscountValue.setMaxWidth(100);
        lblDiscountValue.setAlignment(Pos.CENTER_RIGHT);
        setHalignment(lblDiscountValue, HPos.RIGHT);

        Label txfFinalPrice = new Label("Slut pris:");
        this.add(txfFinalPrice, 3, 8);

        lblFinalPriceAmount = new Label();
        lblFinalPriceAmount.setText(finalPriceAmount);
        this.add(lblFinalPriceAmount, 4, 8);
        setHalignment(lblFinalPriceAmount, HPos.RIGHT);

        // Placeholder for layout
        Button btnPlaceholder = new Button("Placeholder");
        this.add(btnPlaceholder, 3, 9, 2, 1);
        setHalignment(btnPlaceholder, HPos.CENTER);
        btnPlaceholder.setDisable(true);
        btnPlaceholder.setVisible(false);
    }

    private void goToPayment() {
        boolean hasError = false;
        String contactName = txfContactName.getText().trim();
        if (!ValidationHelper.isStringBetween0to30characters(contactName)) {
            txfContactName.getStyleClass().add("error");
            hasError = true;
        }
        String contactInformation = txaContactInformation.getText().trim();
        if (!contactInformation.matches("((.|\\n)*)") || contactInformation.isEmpty()) {
            txaContactInformation.getStyleClass().add("error");
            hasError = true;
        }

        if (!hasError) {
            Rent rent;
            if (!tour) {
                LocalDateTime deliveryDateAndTime = dpDeliveryDateAndTime.getValue().atStartOfDay();
                LocalDateTime returnDateAndTime = dpReturnDateAndTime.getValue().atStartOfDay();
                rent = new Rent(contactName, contactInformation, deliveryDateAndTime, returnDateAndTime);
            } else {
                LocalDateTime tourDateAndTime = productDatePicker.getValue().atStartOfDay();
                rent = new Rent(contactName, contactInformation, tourDateAndTime,
                   ↪ tourDateAndTime);
            }
            for (SalesLine salesLine : sale.getSalesLines()) {
                rent.updateSalesLine(salesLine.getProduct(), salesLine.getQuantity(),
                   ↪ salesLine.getPrice());
            }
            MainApp.changePane(new PaymentPane(rent, pricelist, discount,
               ↪ lblFinalPriceAmount.getText()), true);
        }
    }

}
```

Chapter 27: gui/salepane/SaleOptionsPane.java

```java
package gui.salepane;

import gui.LeftSideBar;
import gui.LeftSideBarInterface;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;

import java.util.LinkedHashMap;
import java.util.Map;

public class SaleOptionsPane extends BorderPane implements LeftSideBarInterface {

    private static SaleOptionsPane saleOptionsPane;

    private LeftSideBar leftSideBar;

    public SaleOptionsPane() {
        this.setPadding(new Insets(20));

        //Creating panes
        Map<String, Pane> navigationMap = new LinkedHashMap<>();
        navigationMap.put("Salg/udlejning", SelectPricelistPane.getInstance());
        navigationMap.put("Uafsluttede salg", SelectOngoingSale.getInstance());

        leftSideBar = new LeftSideBar(navigationMap, this);
        setLeft(leftSideBar);
        changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
    }

    @Override
    public void changeSelected(String nameOfButton) {
        SelectOngoingSale.getInstance().refreshOngoingSales();
        Label title = new Label(nameOfButton);
        title.getStyleClass().add("title");
        setAlignment(title, Pos.CENTER);
        setTop(title);
        setCenter(leftSideBar.getPane(nameOfButton));
    }

    public static SaleOptionsPane getInstance() {
        if (saleOptionsPane == null)
            saleOptionsPane = new SaleOptionsPane();
        return saleOptionsPane;
    }
}
```

Chapter 28: gui/salepane/SalePane.java

```java
package gui.salepane;

import gui.MainApp;
import gui.helpers.ProductAndPriceHelper;
import gui.helpers.ValidationHelper;
import gui.images.LoadImage;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.util.converter.DoubleStringConverter;
import model.Pricelist;
import model.Sale;
import model.SalesLine;
import model.enums.SaleState;
import model.product.Product;
import model.product.Tour;
import model.product.rentable.Rentable;

import java.text.DecimalFormat;
import java.util.Map;

public class SalePane extends GridPane {

    private static final String IMAGE_FOLDER = "res/images/salespane";
    private Map<String, Image> imageMap;

    private Sale sale;
    private Pricelist pricelist;

    private TableView<ProductAndPriceHelper> productTableView;
    private TableView<SalesLine> salesLineTableView;

    private Label lblRentPriceAmount;
    private Label lblTotalPriceAmount;
    private TextField txfDiscount;
    private Label lblFinalPriceAmount;

    private Button btnCompleteSale;

    public SalePane(Sale sale, Pricelist pricelist) {
        this.getStyleClass().add("gridpane");

        this.sale = sale;
        this.pricelist = pricelist;

        // Load images
        imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);

        Label lblPricelist = new Label("Prisliste:");
        lblPricelist.setFont(new Font(16));
        this.add(lblPricelist, 0, 0);

        // Table 1
        productTableView = new TableView<>();
        this.add(productTableView, 0, 1, 1, 6);
        productTableView.setPrefHeight(600);
        productTableView.setPrefWidth(800);
        productTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        productTableView.setEditable(false);
        createProductAndPriceHelperTable();

        Label lblSaleLine = new Label("Salgslinje:");
        lblSaleLine.setFont(new Font(16));
        this.add(lblSaleLine, 1, 0);

        // Table 2
        salesLineTableView = new TableView<>();
        this.add(salesLineTableView, 1, 1, 2, 1);
        salesLineTableView.setPrefHeight(450);
        salesLineTableView.setPrefWidth(400);
        salesLineTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        salesLineTableView.setEditable(true);
        salesLineTableView.setPlaceholder(new Label("Ingen produkter"));
        createSalesLineTable();

        Label lblRentPrice = new Label("Udlejnings pris:");
        this.add(lblRentPrice, 1, 2);
        lblRentPrice.setMinWidth(200);

        lblRentPriceAmount = new Label("0.0");
        setHalignment(lblRentPriceAmount, HPos.RIGHT);
        this.add(lblRentPriceAmount, 2, 2);

        Label lblTotalPrice = new Label("Samlet pris:");
        this.add(lblTotalPrice, 1, 3);
        lblTotalPrice.setMinWidth(200);

        lblTotalPriceAmount = new Label("0.0");
        setHalignment(lblTotalPriceAmount, HPos.RIGHT);
        this.add(lblTotalPriceAmount, 2, 3);

        Label lblDiscount = new Label("Rabat: (kan være %)");
        this.add(lblDiscount, 1, 4);

        txfDiscount = new TextField("0.0");
        this.add(txfDiscount, 2, 4);
        txfDiscount.setMinWidth(100);
        txfDiscount.setMaxWidth(100);
        txfDiscount.setAlignment(Pos.CENTER_RIGHT);
        setHalignment(txfDiscount, HPos.RIGHT);

        txfDiscount.textProperty().addListener((observable -> {
            double finalPrice;
            if (!txfDiscount.getText().isEmpty() && ValidationHelper.isPercentNumber(
                ↪ txfDiscount.getText())) {
                String substring = txfDiscount.getText().substring(0, txfDiscount.getText().
                    ↪ length() - 1);
                double procent = (100 - Double.parseDouble(substring)) / 100;
                finalPrice = Double.parseDouble(lblTotalPriceAmount.getText()) * procent;
            } else if (!txfDiscount.getText().isEmpty() && ValidationHelper.isNumber(
                ↪ txfDiscount.getText())) {
                finalPrice = Double.parseDouble(lblTotalPriceAmount.getText())
                        - Double.parseDouble(txfDiscount.getText());
            } else
                finalPrice = Double.parseDouble(lblTotalPriceAmount.getText());
            lblFinalPriceAmount.setText(String.valueOf(new DecimalFormat("#0.00").format(
                ↪ finalPrice)));
        }));

        Label txfFinalPrice = new Label("Slut pris:");
        this.add(txfFinalPrice, 1, 5);

        lblFinalPriceAmount = new Label("0.0");
        this.add(lblFinalPriceAmount, 2, 5);
        setHalignment(lblFinalPriceAmount, HPos.RIGHT);

        btnCompleteSale = new Button("Gå til betaling");
        this.add(btnCompleteSale, 1, 6, 2, 1);
        btnCompleteSale.setOnAction(event -> goToPayment());
        setHalignment(btnCompleteSale, HPos.CENTER);
        btnCompleteSale.setDisable(true);

        if (sale != null) {
            updateSalesLineGui();
            lblFinalPriceAmount.setText(String.valueOf(sale.getAgreedPrice()));
        }
    }

    private void createProductAndPriceHelperTable() {
        TableColumn<ProductAndPriceHelper, String> column1 = new TableColumn<>("Produkt");
        column1.setCellValueFactory(new PropertyValueFactory<>("product"));
        column1.setStyle("-fx-alignment: CENTER-LEFT");

        TableColumn<ProductAndPriceHelper, String> column2 = new TableColumn<>("
            ↪ Produktgruppe");
        column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));
        column2.setStyle("-fx-alignment: CENTER");

        TableColumn<ProductAndPriceHelper, String> column3 = new TableColumn<>("");
        column3.setCellFactory(param -> new TableCell<ProductAndPriceHelper, String>() {
            final Button btnMinus = new Button("-");
            final TextField txfAmount = new TextField("1");
            final Button btnPlus = new Button("+");
```

```java

            @Override
            public void updateItem(String item, boolean empty) {
                setGraphic(null);
                if (!empty) {
                    btnMinus.setOnAction(event -> {
                        ProductAndPriceHelper ProductAndPriceHelper = getTableView().getItems().get(getIndex());
                        if (ValidationHelper.isNumber(txfAmount.getText()))
                            updateSalesLine(ProductAndPriceHelper.getProduct(), -Integer.parseInt(txfAmount.getText()), ProductAndPriceHelper.getPrice());
                    });
                    if (imageMap != null && imageMap.containsKey("minus")) {
                        ImageView imageView = new ImageView(imageMap.get("minus"));
                        imageView.setFitWidth(13);
                        imageView.setFitHeight(13);
                        btnPlus.setGraphic(imageView);
                        btnPlus.setText(""); // Remove text and use button instead
                    }

                    txfAmount.setMinWidth(35);
                    txfAmount.setMaxWidth(35);
                    txfAmount.setAlignment(Pos.CENTER);

                    btnPlus.setOnAction(event -> {
                        ProductAndPriceHelper ProductAndPriceHelper = getTableView().getItems().get(getIndex());
                        if (ValidationHelper.isNumber(txfAmount.getText()))
                            updateSalesLine(ProductAndPriceHelper.getProduct(), Integer.parseInt(txfAmount.getText()), ProductAndPriceHelper.getPrice());
                    });
                    if (imageMap != null && imageMap.containsKey("plus")) {
                        ImageView imageView = new ImageView(imageMap.get("plus"));
                        imageView.setFitWidth(13);
                        imageView.setFitHeight(13);
                        btnPlus.setGraphic(imageView);
                        btnPlus.setText(""); // Remove text and use button instead
                    }

                    HBox hBox = new HBox();
                    hBox.setSpacing(5);
                    hBox.setAlignment(Pos.CENTER);
                    hBox.getChildren().addAll(btnMinus, txfAmount, btnPlus);
                    setGraphic(hBox);
                }
            }
        });
        column3.setMinWidth(120);
        column3.setMaxWidth(120);

        TableColumn<ProductAndPriceHelper, Double> column4 = new TableColumn<>("Pris");

        TableColumn<ProductAndPriceHelper, String> column5 = new TableColumn<>("Udlejning");
        column5.setCellValueFactory(new PropertyValueFactory<>("deposit"));
        column5.setStyle("-fx-alignment: CENTER-RIGHT");
        column5.setMinWidth(120);
        column5.setMaxWidth(120);

        TableColumn<ProductAndPriceHelper, Double> column6 = new TableColumn<>("Salg");
        column6.setCellValueFactory(new PropertyValueFactory<>("price"));
        column6.setStyle("-fx-alignment: CENTER-RIGHT");
        column6.setMinWidth(120);
        column6.setMaxWidth(120);

        productTableView.getColumns().add(column1);
        productTableView.getColumns().add(column2);
        productTableView.getColumns().add(column3);
        productTableView.getColumns().add(column4);

        column4.getColumns().add(column5);
        column4.getColumns().add(column6);

        ObservableList<ProductAndPriceHelper> productAndPriceHelperObservableList;
        productAndPriceHelperObservableList = FXCollections
                .observableList(ProductAndPriceHelper.parseToProductAndPrice(pricelist.
                    ↪ getProductsWithPrice()));
        productTableView.setItems(productAndPriceHelperObservableList);
    }

    public void createSalesLineTable() {
        TableColumn<SalesLine, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("product"));

        TableColumn<SalesLine, Integer> column2 = new TableColumn<>("Antal");
        column2.setCellValueFactory(new PropertyValueFactory<>("quantity"));
        column2.setStyle("-fx-alignment: CENTER");
        column2.setMinWidth(60);
        column2.setMaxWidth(60);

        TableColumn<SalesLine, Double> column3 = new TableColumn<>("Samlet Pris");

        TableColumn<SalesLine, String> column4 = new TableColumn<>("Udlejning");
        column4.setCellValueFactory(cellData -> cellData.getValue().getTotalLineDeposit() >=
            ↪  0
                ? new SimpleStringProperty(String.valueOf(cellData.getValue().
                    ↪ getTotalLineDeposit()))
                : new SimpleStringProperty(""));
        column4.setStyle("-fx-alignment: CENTER-RIGHT");

        TableColumn<SalesLine, Double> column5 = new TableColumn<>("Salg");
        column5.setCellValueFactory(new PropertyValueFactory<>("totalLinePrice"));
        column5.setStyle("-fx-alignment: CENTER-RIGHT");

        TableColumn<SalesLine, String> column6 = new TableColumn<>("");
        column6.setCellFactory(param -> new TableCell<SalesLine, String>() {
            final Button btnRemove = new Button("X");

            @Override
            public void updateItem(String item, boolean empty) {
                setGraphic(null);
                if (!empty) {
                    btnRemove.setOnAction(event -> {
                        SalesLine salesLine = getTableView().getItems().get(getIndex());
```

```java
                        updateSalesLine(salesLine.getProduct(), -salesLine.getQuantity(),
                            ↪ salesLine.getPrice());
                    });
                    if (imageMap != null && imageMap.containsKey("cross")) {
                        ImageView imageView = new ImageView(imageMap.get("cross"));
                        imageView.setFitWidth(13);
                        imageView.setFitHeight(13);
                        btnRemove.setGraphic(imageView);
                        btnRemove.setText(""); // Remove text and use button instead
                    }
                    setAlignment(Pos.CENTER);
                    setGraphic(btnRemove);
                }
            }
        });
        column6.setMinWidth(40);
        column6.setMaxWidth(40);

        Label lblNyStykPris = new Label("Ny Styk Pris");
        lblNyStykPris.setWrapText(true);
        lblNyStykPris.setTextAlignment(TextAlignment.CENTER);
        TableColumn<SalesLine, Double> column7 = new TableColumn<>();
        column7.setCellValueFactory(new PropertyValueFactory<>("price"));
        column7.setCellFactory(TextFieldTableCell.forTableColumn(new DoubleStringConverter()
            ↪ ));
        column7.setOnEditCommit(event -> {
            event.getRowValue().setNewPrice(event.getNewValue());
            updateSalesLineGui();
        });
        column7.setEditable(true);
        column7.setMinWidth(60);
        column7.setMaxWidth(60);
        column7.setStyle("-fx-alignment: CENTER-RIGHT");
        column7.setGraphic(lblNyStykPris);

        salesLineTableView.getColumns().add(column1);
        salesLineTableView.getColumns().add(column2);
        salesLineTableView.getColumns().add(column3);
        column3.getColumns().add(column4);
        column3.getColumns().add(column5);
        salesLineTableView.getColumns().add(column6);
        salesLineTableView.getColumns().add(column7);
    }

    private void updateSalesLine(Product product, int quantity, double price) {
        if (sale == null)
            sale = new Sale();

        sale.updateSalesLine(product, quantity, price);
        updateSalesLineGui();
    }

    private void updateSalesLineGui() {
        salesLineTableView.getItems().setAll(sale.getSalesLines());
        double totalDeposit = 0d;
        for (SalesLine salesLine : sale.getSalesLines()) {
            totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
                ↪ getTotalLineDeposit() : 0;
        }
        lblRentPriceAmount.setText(String.valueOf(totalDeposit));
        lblTotalPriceAmount.setText(String.valueOf(sale.totalSalePrice()));
        double finalPrice = !txfDiscount.getText().isEmpty() && ValidationHelper.isNumber(
            ↪ txfDiscount.getText())
                ? Double.parseDouble(lblTotalPriceAmount.getText()) - Double.parseDouble(
                    ↪ txfDiscount.getText())
                : Double.parseDouble(lblTotalPriceAmount.getText());
        lblFinalPriceAmount.setText(Double.toString(finalPrice));

        if (sale.getSalesLines().isEmpty())
            btnCompleteSale.setDisable(true);
        else
            btnCompleteSale.setDisable(false);

        if (containsRent())
            btnCompleteSale.setText("Videre");
        else
            btnCompleteSale.setText("Til betaling");
    }

    private void goToPayment() {
        // Og hvis ikke en tour der er igangværende
        if (containsRent() && !sale.getSaleState().equals(SaleState.DELAYED))
            MainApp.changePane(new RentInformationPane(sale, pricelist, txfDiscount.getText
                ↪ (), lblFinalPriceAmount.getText(), isTourOnly()), true);
        else
            MainApp.changePane(new PaymentPane(sale, pricelist, txfDiscount.getText(),
                ↪ lblFinalPriceAmount.getText()), true);
    }

    private boolean containsRent() {
        for (SalesLine salesLine : sale.getSalesLines()) {
            if (salesLine.getProduct().getRentableStrategy() instanceof Rentable)
                return true;
        }
        return false;
    }

    private boolean isTourOnly() {
        for (SalesLine salesLine : sale.getSalesLines()) {
            if (salesLine.getProduct() instanceof Tour && sale.getSalesLines().size() == 1)
                return true;
        }
        return false;
    }
}
```

Chapter 29: gui/salepane/SelectOngoingSale.java

```java
package gui.salepane;

import controller.Controller;
import gui.MainApp;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.beans.property.SimpleStringProperty;
import javafx.geometry.Pos;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import model.Pricelist;
import model.Rent;
import model.Sale;
import model.enums.SaleState;

import java.util.LinkedList;
import java.time.LocalDateTime;

public class SelectOngoingSale extends BorderPane {

    private static SelectOngoingSale selectOngoingSale;

    private ObservableList<Sale> obsOngoingSales;
    private ChoiceBox<Pricelist> pricelistChoiceBox;

    private TableView<Sale> ongoingSaleTableView;
    private HBox hbContent;

    private SelectOngoingSale() {
        obsOngoingSales = FXCollections.observableList(new LinkedList<>());

        hbContent = new HBox();
        hbContent.setSpacing(20);
        hbContent.setAlignment(Pos.CENTER);
        this.setCenter(hbContent);
        ongoingSaleTableView = getOngoingSaleTableView();
        hbContent.getChildren().add(ongoingSaleTableView);

        pricelistChoiceBox = new ChoiceBox<>();
        pricelistChoiceBox.getItems().addAll(Controller.getInstance().getPricelists());
        pricelistChoiceBox.getSelectionModel().selectFirst();
        hbContent.getChildren().add(pricelistChoiceBox);
        refreshOngoingSales();
    }

    public void refreshOngoingSales() {
        obsOngoingSales.clear(); // because a sale can be completed (not just new sales
           ↪ appearing)
        for (Sale sale : Controller.getInstance().getSales()) {
            if (!sale.getSaleState().equals(SaleState.COMPLETED))
                obsOngoingSales.add(sale);
        }
        hbContent.getChildren().clear();
        hbContent.getChildren().addAll(ongoingSaleTableView, pricelistChoiceBox);
    }

    private TableView<Sale> getOngoingSaleTableView() {
        TableView<Sale> ongoingSaleTableView = new TableView<>();
        ongoingSaleTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        ongoingSaleTableView.setEditable(false);
        ongoingSaleTableView.setPrefHeight(500);
        ongoingSaleTableView.setPrefWidth(500);

        ongoingSaleTableView.getSelectionModel().selectedItemProperty().addListener((
           ↪ observable, oldValue, newValue) -> goToSale(ongoingSaleTableView.
           ↪ getSelectionModel().getSelectedItem()));

        TableColumn<Sale, String> column1 = new TableColumn<>("Tidspunkt");
        column1.setCellValueFactory(cellData -> {
            LocalDateTime time = cellData.getValue().getTimestamp();
            return new SimpleStringProperty(String.format("%s-%s-%s - %s:%s", time.
               ↪ getDayOfMonth(),
                    time.getMonthValue(), time.getYear(), time.getHour(), time.getMinute()))
                       ↪ ;
        });

        TableColumn<Sale, String> column2 = new TableColumn<>("ID");
        column2.setCellValueFactory(new PropertyValueFactory<>("id"));

        TableColumn<Sale, String> column3 = new TableColumn<>("Kontakt navn");
        column3.setCellValueFactory(cellData -> (cellData.getValue() instanceof Rent
                ? new SimpleStringProperty(((Rent) cellData.getValue()).getContactName())
                : new SimpleStringProperty("")));

        TableColumn<Sale, String> column4 = new TableColumn<>("Kontakt information");
        column4.setCellValueFactory(cellData -> (cellData.getValue() instanceof Rent
                ? new SimpleStringProperty(((Rent) cellData.getValue()).
                   ↪ getContactInformation())
                : new SimpleStringProperty("")));

        ongoingSaleTableView.getColumns().add(column1);
        ongoingSaleTableView.getColumns().add(column2);
        ongoingSaleTableView.getColumns().add(column3);
        ongoingSaleTableView.getColumns().add(column4);


        ongoingSaleTableView.setItems(obsOngoingSales);
        return ongoingSaleTableView;
    }


    public void goToSale(Sale sale) {
        MainApp.changePane(new SalePane(sale, pricelistChoiceBox.getSelectionModel().
            ↪ getSelectedItem()), true);
    }

    public static SelectOngoingSale getInstance() {
        if (selectOngoingSale == null)
            selectOngoingSale = new SelectOngoingSale();
        return selectOngoingSale;
    }

}
```

Chapter 30: gui/salepane/SelectPricelistPane.java

```java
package gui.salepane;

import controller.Controller;
import gui.MainApp;
import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Pagination;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import model.Pricelist;

import java.util.LinkedList;
import java.util.List;

public class SelectPricelistPane extends BorderPane {

    private static SelectPricelistPane selectPricelistPane;

    private static List<Button> buttonList;

    private static final int BUTTONS_PER_PAGE = 5;
    private static Pagination pagination;

    private SelectPricelistPane() {
        VBox vBox = new VBox();
        vBox.setAlignment(Pos.CENTER);
        vBox.setSpacing(10);
        this.setCenter(vBox);

        Label lblSelect = new Label("Vælg prisliste:");
        lblSelect.setFont(new Font(24));
        vBox.getChildren().add(lblSelect);

        pagination = new Pagination();

        buttonList = new LinkedList<>();

        fillList();

        pagination.setCurrentPageIndex(0);
        pagination.setMaxPageIndicatorCount(3);

        pagination.setPageFactory((pageIndex) -> {

            HBox hBox = new HBox();
            hBox.setAlignment(Pos.CENTER);
            hBox.setSpacing(50);

            for (int i = 0; i < BUTTONS_PER_PAGE; i++) {
                int index = i + (pageIndex * BUTTONS_PER_PAGE);
                if (index > buttonList.size() - 1) break;
                Button button = buttonList.get(index);
                button.setWrapText(true);
                button.setTextAlignment(TextAlignment.CENTER);

                VBox vBox1 = new VBox();
                vBox1.setAlignment(Pos.CENTER);
                vBox1.setSpacing(10);

                vBox1.getChildren().add(button);
                vBox1.getChildren().add(new Label(button.getText()));

                hBox.getChildren().add(vBox1);
            }
            int i = ((buttonList.size() - 1) / BUTTONS_PER_PAGE) + 1;
            pagination.setPageCount(i);

            return hBox;
        });

        vBox.getChildren().add(pagination);
    }

    private void fillList() {
        for (Pricelist pricelist : Controller.getInstance().getPricelists()) {
            Button button = new Button(pricelist.getName());
            button.setPrefHeight(75);
            button.setPrefWidth(75);
            button.setOnAction(event -> selectedPricelist(pricelist));
            buttonList.add(button);
        }
    }

    public void addPricelist(Pricelist pricelist) {
        Button button = new Button(pricelist.getName());
        button.setPrefHeight(75);
        button.setPrefWidth(75);
        button.setOnAction(event -> selectedPricelist(pricelist));
        buttonList.add(button);
    }

    private static void selectedPricelist(Pricelist pricelist) {
        MainApp.changePane(new SalePane(null, pricelist), true);
    }

    public static SelectPricelistPane getInstance() {
        if (selectPricelistPane == null)
            selectPricelistPane = new SelectPricelistPane();
        return selectPricelistPane;
    }
}
```

Chapter 31: gui/salepane/TableViewSalesLines.java

```java
package gui.salepane;

import gui.images.LoadImage;
import javafx.beans.property.SimpleStringProperty;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import model.Sale;
import model.SalesLine;

public class TableViewSalesLines extends TableView<SalesLine> {

    public TableViewSalesLines(Sale sale) {
        setPrefHeight(450);
        setPrefWidth(400);
        setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        setEditable(false);
        setPlaceholder(new Label("Ingen produkter"));

        TableColumn<SalesLine, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("product"));
        column1.setStyle("-fx-alignment: CENTER-LEFT");

        TableColumn<SalesLine, Integer> column2 = new TableColumn<>("Antal");
        column2.setCellValueFactory(new PropertyValueFactory<>("quantity"));
        column2.setStyle("-fx-alignment: CENTER");

        TableColumn<SalesLine, Double> column3 = new TableColumn<>("Samlet Pris");

        TableColumn<SalesLine, String> column4 = new TableColumn<>("Udlejning");
        column4.setCellValueFactory(cellData -> cellData.getValue().getTotalLineDeposit() >=
            ↪ 0
                ? new SimpleStringProperty(String.valueOf(cellData.getValue().
                    ↪ getTotalLineDeposit()))
                : new SimpleStringProperty(""));
        column4.setStyle("-fx-alignment: CENTER-RIGHT");

        TableColumn<SalesLine, Double> column5 = new TableColumn<>("Salg");
        column5.setCellValueFactory(new PropertyValueFactory<>("totalLinePrice"));
        column5.setStyle("-fx-alignment: CENTER-RIGHT");

        TableColumn<SalesLine, String> column6 = new TableColumn<>("");
        column6.setCellFactory(param -> new TableCell<SalesLine, String>() {
            final Button btnRemove = new Button("X");

            @Override
            public void updateItem(String item, boolean empty) {
                setGraphic(null);
                if (!empty) {
                    Image image = LoadImage.loadImage("res/images/salespane/cross.png");
                    if (image != null) {
                        ImageView imageView = new ImageView(image);
                        imageView.setFitWidth(13);
                        imageView.setFitHeight(13);
                        btnRemove.setGraphic(imageView);
                        btnRemove.setText(""); // Remove text and use button instead
                    }
                    btnRemove.setDisable(true);
                    setAlignment(Pos.CENTER);
                    setGraphic(btnRemove);
                }
            }
        });
        column6.setMinWidth(40);
        column6.setMaxWidth(40);

        getColumns().add(column1);
        getColumns().add(column2);
        getColumns().add(column3);
        column3.getColumns().add(column4);
        column3.getColumns().add(column5);
        getColumns().add(column6);

        getItems().addAll(sale.getSalesLines());
    }

}
```

Chapter 32: gui/salepane/TableViewSalesLines.java

```java
package gui.salepane;

import gui.images.LoadImage;
import javafx.beans.property.SimpleStringProperty;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import model.Sale;
import model.SalesLine;

public class TableViewSalesLines extends TableView<SalesLine> {

    public TableViewSalesLines(Sale sale) {
        setPrefHeight(450);
        setPrefWidth(400);
        setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        setEditable(false);
        setPlaceholder(new Label("Ingen produkter"));

        TableColumn<SalesLine, String> column1 = new TableColumn<>("Navn");
        column1.setCellValueFactory(new PropertyValueFactory<>("product"));
        column1.setStyle("-fx-alignment: CENTER-LEFT");

        TableColumn<SalesLine, Integer> column2 = new TableColumn<>("Antal");
        column2.setCellValueFactory(new PropertyValueFactory<>("quantity"));
        column2.setStyle("-fx-alignment: CENTER");

        TableColumn<SalesLine, Double> column3 = new TableColumn<>("Samlet Pris");

        TableColumn<SalesLine, String> column4 = new TableColumn<>("Udlejning");
        column4.setCellValueFactory(cellData -> cellData.getValue().getTotalLineDeposit() >=
            ↪  0
                ? new SimpleStringProperty(String.valueOf(cellData.getValue().
                    ↪ getTotalLineDeposit()))
                : new SimpleStringProperty(""));
        column4.setStyle("-fx-alignment: CENTER-RIGHT");

        TableColumn<SalesLine, Double> column5 = new TableColumn<>("Salg");
        column5.setCellValueFactory(new PropertyValueFactory<>("totalLinePrice"));
        column5.setStyle("-fx-alignment: CENTER-RIGHT");

        TableColumn<SalesLine, String> column6 = new TableColumn<>("");
        column6.setCellFactory(param -> new TableCell<SalesLine, String>() {
            final Button btnRemove = new Button("X");

            @Override
            public void updateItem(String item, boolean empty) {
                setGraphic(null);
                if (!empty) {
                    Image image = LoadImage.loadImage("res/images/salespane/cross.png");
                    if (image != null) {
                        ImageView imageView = new ImageView(image);
                        imageView.setFitWidth(13);
                        imageView.setFitHeight(13);
                        btnRemove.setGraphic(imageView);
                        btnRemove.setText(""); // Remove text and use button instead
                    }
                    btnRemove.setDisable(true);
                    setAlignment(Pos.CENTER);
                    setGraphic(btnRemove);
                }
            }
        });
        column6.setMinWidth(40);
        column6.setMaxWidth(40);

        getColumns().add(column1);
        getColumns().add(column2);
        getColumns().add(column3);
        column3.getColumns().add(column4);
        column3.getColumns().add(column5);
        getColumns().add(column6);

        getItems().addAll(sale.getSalesLines());
    }

}
```

Chapter 33: gui/StatisticPane.java

```java
package gui;

import gui.statistics.ExportPane;
import gui.statistics.FilterPane;
import gui.statistics.GraphsPane;
import gui.statistics.ReportTextPane;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;

import java.util.LinkedHashMap;
import java.util.Map;

public class StatisticPane extends BorderPane implements LeftSideBarInterface {

    private static StatisticPane statisticPane = getInstance();

    private Label lblTitle;
    private LeftSideBar leftSideBar;

    private StatisticPane() {
        this.setPadding(new Insets(20));

        // Initialize
        lblTitle = new Label();
        lblTitle.setPrefWidth(700);
        lblTitle.getStyleClass().add("subtitle");
        lblTitle.setPrefWidth(700);
        this.setTop(lblTitle);

        Map<String, Pane> navigationMap = new LinkedHashMap<>();
        navigationMap.put("Filtrer", FilterPane.getInstance());
        navigationMap.put("Grafer", GraphsPane.getInstance());
        navigationMap.put("Tekstrapport", ReportTextPane.getInstance());
        navigationMap.put("EksportÃl'r rapport", ExportPane.getInstance());
        leftSideBar = new LeftSideBar(navigationMap, this);
        setLeft(leftSideBar);

        changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
    }

    public void setTitle(String title) {
        lblTitle.setText(title);
    }

    @Override
    public void changeSelected(String nameOfButton) {
        Label title = new Label(nameOfButton);
        title.getStyleClass().add("title");
        setAlignment(title, Pos.CENTER);
        setTop(title);
        setCenter(leftSideBar.getPane(nameOfButton));
    }

    public static StatisticPane getInstance() {
        if (statisticPane == null)
            statisticPane = new StatisticPane();
        return statisticPane;
    }


}
```

Chapter 34: gui/statistics/ExportPane.java

```java
package gui.statistics;

import javafx.geometry.Pos;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.stage.DirectoryChooser;
import model.Sale;
import model.SalesLine;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class ExportPane extends GridPane {
    private static ExportPane exportPane;

    private ExportPane() {
        this.getStyleClass().add("gridpane");
        this.setAlignment(Pos.CENTER);

        // Buttons
        Button btnSaveTxt = new Button("Gem .txt");
        Button btnSaveCSV = new Button("Gem .csv");
        btnSaveTxt.getStyleClass().add("important");
        btnSaveCSV.getStyleClass().add("important");
        HBox hbButtons = new HBox(btnSaveTxt, btnSaveCSV);
        hbButtons.setSpacing(100);

        // Listeners
        btnSaveCSV.setOnAction(e -> exportCsv());
        btnSaveTxt.setOnAction(e -> exportTxt());

        // GUI
        this.add(hbButtons, 0, 0);
    }

    ////// Export
    private void exportTxt() {
        File directory = new DirectoryChooser().showDialog(null);
        if (directory != null) {
            String filename = "salgslinjer for perioden " + FilterPane.getInstance().
                ↪ getDateTimeFrom().toLocalDate() + "-" + FilterPane.getInstance().
                ↪ getDateTimeTo().toLocalDate()
                    + ".txt";
            File fileOutput = new File(directory + File.separator + filename);
            writeFile(fileOutput, ReportTextPane.getInstance().getText());

        }
    }

    private void exportCsv() {
        File directory = new DirectoryChooser().showDialog(null);
        if (directory != null) {
            String filename = "salgslinjer for perioden " + FilterPane.getInstance().
                ↪ getDateTimeFrom().toLocalDate() + "-" + FilterPane.getInstance().
                ↪ getDateTimeTo().toLocalDate()
                    + ".csv";
            File fileOutput = new File(directory + File.separator + filename);
            StringBuilder sbOutputContent = new StringBuilder();
            sbOutputContent.append("dato,salgID,produkt,antal,pris i kr.");
            for (Sale sale : FilterPane.getInstance().getSalesFromFilter()) {
                for (SalesLine salesLine : sale.getSalesLines()) {
                    sbOutputContent.append("\n");
                    sbOutputContent.append(String.format("\"%s\",%s,\"%s\",%s,%s", sale.
                        ↪ getTimestamp(), sale.getId(),
                            salesLine.getProduct().getName(), salesLine.getQuantity(),
                                ↪ salesLine.getTotalLinePrice()));
                }
            }
            writeFile(fileOutput, sbOutputContent.toString());
        }
    }

    private void writeFile(File file, String content) {
        try (FileWriter fileWriter = new FileWriter(file)) {
            fileWriter.write(content);
        } catch (IOException e) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setContentText("Fejl: kan ikke gemme filen.");
            alert.showAndWait();
        }
    }


    public static ExportPane getInstance() {
        if (exportPane == null)
            exportPane = new ExportPane();
        return exportPane;
    }
}
```

Chapter 35: gui/statistics/FilterPane.java

```java
package gui.statistics;

import controller.Controller;
import gui.StatisticPane;
import gui.helpers.DateIntervalPicker;
import javafx.collections.ListChangeListener;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.SelectionMode;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import model.Pricelist;
import model.Sale;
import model.enums.SaleState;
import model.product.Product;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

public class FilterPane extends GridPane {
    private static FilterPane filterPane;

    private DateIntervalPicker datePicker;
    private ListView<Pricelist> lwPricelists;
    private ListView<Product> lwProducts;
    private PieChartExtended pcProductsUnwrapBundles;
    private Label lblProductQuantityOverview;

    private FilterPane() {
        this.getStyleClass().add("gridpane");

        datePicker = new DateIntervalPicker();
        // listviews
        lwPricelists = new ListView<>();
        lwPricelists.getItems().setAll(Controller.getInstance().getPricelists());
        VBox vbPricelist = new VBox(new Label("Prislister"), lwPricelists);
        vbPricelist.setMaxHeight(200);
        lwProducts = new ListView<>();
        lwProducts.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
        VBox vbProducts = new VBox(new Label("Produkter (fra valgte prislister)"),
            ↪ lwProducts);
        vbProducts.setMaxHeight(200);
        Button btnApply = new Button("Anvend filter");
        Button btnSelectAll = new Button("Vælg alt");
        VBox vbFilterButtons = new VBox(btnApply, btnSelectAll);
//          btnApply.getStyleClass().add("important");
        pcProductsUnwrapBundles = new PieChartExtended("Alle salg for perioden");
        lblProductQuantityOverview = new Label();


        // Listeners
        lwPricelists.getSelectionModel().getSelectedItems()
                .addListener((ListChangeListener<Pricelist>) c -> updateProductList());
        btnSelectAll.setOnAction(e -> {
            lwProducts.getSelectionModel().selectAll();
            updateReport();
        });
        btnApply.setOnAction(e -> updateReport());

        this.add(datePicker, 0, 1);
        this.add(vbPricelist, 0, 2);
        this.add(vbProducts, 1, 2);
        this.add(vbFilterButtons, 2, 2);
        this.add(pcProductsUnwrapBundles, 0, 3);
        this.add(lblProductQuantityOverview,1,3);
    }

    /**
     * Updates the list of products to reflect the Pricelist selection
     * contains instead of having it inside lwProducts
     */
    private void updateProductList() {
        lwProducts.getItems().clear();
        lwPricelists.getSelectionModel().getSelectedItems()
                .forEach(pricelist -> pricelist.getProductsWithPrice()
                        .forEach((product, price) -> {
                            if (!lwProducts.getItems().contains(product))
                                lwProducts.getItems().add(product);
                        }));
        lwProducts.getSelectionModel().selectAll();
    }

    public List<Product> getSelectedProducts() {
        return lwProducts.getSelectionModel().getSelectedItems();
    }

    public LocalDateTime getDateTimeFrom() {
        return datePicker.getDateFrom().atStartOfDay();
    }

    public LocalDateTime getDateTimeTo() {
        return datePicker.getDateTo().atStartOfDay();
    }

    /**
     * When any changes to products or date interval is made, this method is called
     * and updates all relevant content.
     * <p>
     * -   method is private because it can be updated by change of filter through other
     ↪ panes
     */
    public void updateReport() {
        StatisticPane.getInstance().setTitle(
                String.format("Salgsoplysning fra: [%s - %s]", datePicker.getDateFrom(), datePicker.getDateTo()));
        Set<Sale> salesInDateRange = getSalesFromFilter();

        Map<Product, Integer> productsWithQuantity = GraphsPane.getInstance().getProductsWithQuantity(
                salesInDateRange.stream().flatMap(sale -> sale.getSalesLines().stream()).collect(Collectors.toSet()));
        Map<Product, Integer> bundleProductsWithQuantity = GraphsPane.getInstance().unwrapBundleFromProductsWithQuantity(productsWithQuantity);
        Map<Product, Integer> unwrapedBundleWithProductWithQuantity = GraphsPane.getInstance().mergeAndRemoveBundleFromProductWithQuantityMaps(
                bundleProductsWithQuantity, productsWithQuantity);

        GraphsPane.getInstance().setPcProducts(productsWithQuantity);
        GraphsPane.getInstance().setPcBundles(bundleProductsWithQuantity);
        GraphsPane.getInstance().setPcProductsUnwrapBundles(unwrapedBundleWithProductWithQuantity);
        pcProductsUnwrapBundles.setData(GraphsPane.getInstance().getPieSlicesFromProductsWithQuantity(unwrapedBundleWithProductWithQuantity));
        GraphsPane.getInstance().setPcPaymentMethods(salesInDateRange);
        updateLblProductQuantityOverview(unwrapedBundleWithProductWithQuantity);

        ReportTextPane.getInstance().setText(salesInDateRange, unwrapedBundleWithProductWithQuantity);
    }

    private void updateLblProductQuantityOverview(Map<Product, Integer> productQuantityMap) {
        StringBuilder sbOverview = new StringBuilder();
        for (Product product : productQuantityMap.keySet()){
            sbOverview.append(product.getName() + ": " + productQuantityMap.get(product) + "\n");
        }
        lblProductQuantityOverview.setText(sbOverview.toString());
    }

    public Set<Sale> getSalesFromFilter() {
        List<Product> productList = FilterPane.getInstance().getSelectedProducts();
        LocalDateTime dateTimeFrom = FilterPane.getInstance().getDateTimeFrom();
        LocalDateTime dateTimeTo = FilterPane.getInstance().getDateTimeTo();

        return Controller.getInstance().getSales().stream()
                .filter(sale -> (sale.getTimestamp().isAfter(dateTimeFrom) && sale.getTimestamp().isBefore(dateTimeTo)))
                .filter(sale -> sale.getSaleState().equals(SaleState.COMPLETED))
                .filter(sale -> sale.getSalesLines().stream()
                        .anyMatch(salesLine -> productList.contains(salesLine.getProduct())))
                .collect(Collectors.toSet());
    }

    public static FilterPane getInstance() {
        if (filterPane == null)
            filterPane = new FilterPane();
        return filterPane;
    }
```

```
}
```

Chapter 36: gui/statistics/GraphsPane.java

```java
package gui.statistics;

import javafx.scene.chart.PieChart;
import javafx.scene.layout.GridPane;
import model.Sale;
import model.SalesLine;
import model.enums.PaymentMethod;
import model.product.Bundle;
import model.product.Product;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

public class GraphsPane extends GridPane {
    private static GraphsPane graphsPane;


    private PieChartExtended pcProducts; // products where bundles show as slice
    private PieChartExtended pcBundles; // products sold by bundle in selection
    private PieChartExtended pcProductsUnwrapBundles; // products where bundles are "
        ↪ unwraped"
    private PieChartExtended pcPaymentMethods;

    private GraphsPane() {
        this.getStyleClass().add("gridpane");

        // piecharts
        pcProducts = new PieChartExtended("Produkter (bundt vises som sammenpakning)");
        pcBundles = new PieChartExtended("Produkter solgt gennem bundter");
        pcProductsUnwrapBundles = new PieChartExtended("Produkter (bundters produkter inkl.)
            ↪ ");
        pcPaymentMethods = new PieChartExtended("Betalingsmetoder");

        this.add(pcProducts,0,0);
        this.add(pcBundles,1,0);
        this.add(pcProductsUnwrapBundles,0,1);
        this.add(pcPaymentMethods,1,1);
    }

    public Map<Product, Integer> unwrapBundleFromProductsWithQuantity(Map<Product, Integer>
        ↪ productsWithQuantity) {
        Map<Product, Integer> unwrapedProductsWithQuantity = new HashMap<>();
        productsWithQuantity.keySet().stream().filter(product -> product instanceof Bundle)
                .forEach(bundle -> ((Bundle) bundle).getProductsWithQuantity()
                        .forEach((product, quantity) -> unwrapedProductsWithQuantity.put(
                            ↪ product,
                                unwrapedProductsWithQuantity.getOrDefault(product, 0)
                                        + quantity * productsWithQuantity.get(bundle))));

        return unwrapedProductsWithQuantity;
    }

    public Map<Product, Integer> getProductsWithQuantity(Set<SalesLine> salesLines) {
        HashMap<Product, Integer> products = new HashMap<>();
        for (SalesLine salesLine : salesLines) {
            products.put(salesLine.getProduct(), products.getOrDefault(salesLine.getProduct
                ↪ (), 0) + 1);
        }
        return products;
    }

    public Map<Product, Integer> mergeAndRemoveBundleFromProductWithQuantityMaps(
            Map<Product, Integer> productWithQuantityAndBundle, Map<Product, Integer>
                ↪ productWithQuantity) {
        // Remove all instances of bundle
        productWithQuantityAndBundle.entrySet().removeIf(productQuantity -> productQuantity.
            ↪ getKey() instanceof Bundle);

        for (Product product : productWithQuantityAndBundle.keySet()) {
            productWithQuantity.put(product,
                    productWithQuantity.getOrDefault(product, 0) +
                        ↪ productWithQuantityAndBundle.get(product));
        }

        return productWithQuantity;
    }

    public List<PieChart.Data> getPieSlicesFromProductsWithQuantity(Map<Product, Integer>
        ↪ productQuantity) {
        return productQuantity.keySet().stream()
                .map(product -> new PieChart.Data(product.getName(), productQuantity.get(
                    ↪ product)))
                .collect(Collectors.toList());
    }

    public void setPcProducts(Map<Product, Integer> productsWithQuantity) {
        pcProducts.setData(getPieSlicesFromProductsWithQuantity(productsWithQuantity));
    }

    public void setPcBundles(Map<Product, Integer> productsWithQuantity) {
        pcBundles.setData(getPieSlicesFromProductsWithQuantity(productsWithQuantity));
    }

    public void setPcProductsUnwrapBundles(Map<Product, Integer> productsWithQuantity) {
        pcProductsUnwrapBundles.setData(getPieSlicesFromProductsWithQuantity(
            ↪ productsWithQuantity));
    }

    public void setPcPaymentMethods(Set<Sale> sales){

        pcPaymentMethods.setData(getPieSlicesFromPaymentMethod(getPaymentAmountMap(sales)));
    }

    public Map<PaymentMethod, Double> getPaymentAmountMap(Set<Sale> sales) {
        HashMap<PaymentMethod, Double> paymentAmountMap = new HashMap<>();
        for (Sale sale : sales) {
            Map<PaymentMethod, Double> transfers = sale.getPayment().getTransfers();
            for (PaymentMethod paymentMethod : transfers.keySet()) {
                paymentAmountMap.put(paymentMethod,paymentAmountMap.getOrDefault(
                    ↪ paymentMethod,0d)+transfers.get(paymentMethod));
            }
        }
        return paymentAmountMap;
    }

    public List<PieChart.Data> getPieSlicesFromPaymentMethod(Map<PaymentMethod, Double>
        ↪ paymentMethodAmountMap) {
        return paymentMethodAmountMap.keySet().stream()
                .map(paymentMethod -> new PieChart.Data(paymentMethod.name(),
                    ↪ paymentMethodAmountMap.get(paymentMethod)))
                .collect(Collectors.toList());
    }

    public static GraphsPane getInstance() {
        if (graphsPane == null)
            graphsPane = new GraphsPane();
        return graphsPane;
    }
}
```

Chapter 37: gui/statistics/PieChartExtended.java

```java
package gui.statistics;

import javafx.scene.chart.PieChart;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;

import java.util.List;

public class PieChartExtended extends PieChart {

    public PieChartExtended(String title) {
        setTitle(title);
    }

    public void setData(List<PieChart.Data> dataPoints) {
        this.getData().setAll(dataPoints);

        final Label caption = new Label("");
        caption.setTextFill(Color.DARKORANGE);
        caption.setStyle("-fx-font: 24 arial;");
        this.getChildren().add(caption);
        for (final PieChart.Data data : this.getData()) {
            data.getNode().addEventHandler(MouseEvent.MOUSE_ENTERED, e -> {
                // Position not known before Node has been initialized
                caption.setTranslateX(e.getSceneX() - getLayoutX());
                caption.setTranslateY(e.getSceneY());
                caption.setText(String.valueOf(data.getPieValue()));
            });

            data.getNode().addEventHandler(MouseEvent.MOUSE_EXITED, e -> caption.setText("")
               ↪ );
        }
    }
}
```

Chapter 38: gui/statistics/ReportTextPane.java

```java
package gui.statistics;

import javafx.geometry.Pos;
import javafx.scene.control.TextArea;
import javafx.scene.layout.GridPane;
import model.Sale;
import model.SalesLine;
import model.enums.PaymentMethod;
import model.product.Product;

import java.util.Map;
import java.util.Set;

public class ReportTextPane extends GridPane {
    private static ReportTextPane reportTextPane;

    private TextArea txaHistory;

    private ReportTextPane() {
        this.getStyleClass().add("gridpane");
        this.setAlignment(Pos.CENTER);

        txaHistory = new TextArea();
        txaHistory.setMinWidth(500);
        txaHistory.setMinHeight(600);
        this.add(txaHistory,0,0);
    }

    private String writeSalesReport(Set<Sale> sales, Map<Product, Integer>
        ↪ productsWithQuantity) {
        Map<PaymentMethod, Double> paymentAmountMap = GraphsPane.getInstance().
            ↪ getPaymentAmountMap(sales);
        StringBuilder sbLog = new StringBuilder();
        sbLog.append(String.format("Sales for perioden: %s-%s%n", FilterPane.getInstance().
            ↪ getDateTimeFrom().toLocalDate(), FilterPane.getInstance().getDateTimeTo().
            ↪ toLocalDate()));
        sbLog.append(String.format("* antal salg: %s%n", sales.size()));

        sbLog.append("\n------- Produkt oversigt: \n");
        for (PaymentMethod paymentMethod : paymentAmountMap.keySet()){
            sbLog.append("- " + paymentMethod.name().toLowerCase() + ": " + paymentAmountMap
                ↪ .get(paymentMethod) +"kr\n");
        }

        sbLog.append("\n------- Salgsmetoder: \n");
        sbLog.append(String.format("%3s) %4s: %30s%n", "#", "produkt", "antal"));
        for (int i = 0; i < productsWithQuantity.size(); i++) {
            Product product = (Product) productsWithQuantity.keySet().toArray()[i];
            sbLog.append(String.format("%3s) %4s: %30s%n", i + 1, product, productsWithQuantity.get(product)));
        }

        StringBuilder sbSales = new StringBuilder();
        sbSales.append(String.format("Alle salg: %n"));
        for (Sale sale : sales) {
            sbSales.append(String.format("-------------%n"));
            sbSales.append(String.format("Salg: d. %s %s handlet for %skr%n", sale.getTimestamp().toLocalDate(),
                    sale.getTimestamp().toLocalTime(), sale.totalSalePrice()));
            for (SalesLine salesLine : sale.getSalesLines()) {
                sbSales.append(String.format("  - %s x %s : %skr%n", salesLine.getProduct().getName(),
                        salesLine.getQuantity(), salesLine.getTotalLinePrice()));
            }
            sbSales.append(String.format("%n"));
        }
        return sbLog.toString() + "\n" + sbSales.toString();
    }

    public void setText(Set<Sale> salesInDateRange, Map<Product, Integer> unwrapedBundleWithProductWithQuantity) {
        txaHistory.setText(writeSalesReport(salesInDateRange, unwrapedBundleWithProductWithQuantity));
    }

    public String getText(){
       return txaHistory.getText();
    }

    public static ReportTextPane getInstance() {
        if (reportTextPane == null)
            reportTextPane = new ReportTextPane();
        return reportTextPane;
    }
}
```

Chapter 39: model/enums/PaymentMethod.java

```java
package model.enums;

public enum PaymentMethod {

    MOBILEPAY("MobilePay"),
    VOUCHER("Klippekort"),
    PAYMENTCARD("Betalingskort"),
    CASH("Kontant");

    PaymentMethod(String text) {
        this.text = text;
    }

    private final String text;

    @Override
    public String toString() {
        return text;
    }
}
```

Chapter 40: model/enums/SaleSate.java

```java
package model.enums;

public enum SaleState {

    INITIATED,
    DELAYED,
    COMPLETED

}
```

Chapter 41: model/enums/Unit.java

```java
package model.enums;

public enum Unit {

        kg,
        l,
        cl

}
```

Chapter 42: model/Payment.java

```java
package model;

import model.enums.PaymentMethod;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

/**
 * Creates a payment which tracks the state (finished, initiated etc)
 * and the amount currently assigned to a sale.
 * A payment consist of multiple payments (transfers) which continuously can be added.
 * A transfer has an amount and a source (paymentcard, cash etc).
 *
 * @author: Mathias, Kasper, Alexander
 */
public class Payment implements Serializable {

        private Map<PaymentMethod, Double> transfers = new HashMap<>();

        /**
         * Payment is empty until a transfer is added.
         *              thus empty constructor
         */
        public Payment() {
        }

        /**
         * Adds a transfer to the payment of a sale.
         * If adding an already added paymentMethod, then the previous and new amount will
            ↪ be added.
         * Therefore, one amount is stored per paymentMethod.
         *
         * @param paymentMethod, method of the payment.
         * @param amount,        the amount of money spend in the transfer.
         */
        public void addTransfer(PaymentMethod paymentMethod, double amount) {
                transfers.put(paymentMethod, transfers.getOrDefault(paymentMethod, 0.0) +
                    ↪ amount);
        }

        /**
         * calculate the total amount received for the payment.
         *
         * @return the total amount for the payment.
         */
        public double calcTotal() {
                double total = 0;
                for (double amount : transfers.values()) {
                        total += amount;
                }
                return total;
        }

        /**
         * Get the transfers as a map. So that each key represent a
         * paymentMethod and corresponding value is amount transferred through that method.
         *
         * @return the list of paymentMethods paired with the associated amount
         */
        public Map<PaymentMethod, Double> getTransfers() {
                return transfers;
        }
}
```

Chapter 43: model/Pricelist.java

```java
package model;

import model.product.Product;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

public class Pricelist implements Serializable {

    private String name;
    private Map<Product, Double> productsWithPrice = new HashMap<>();

    public Pricelist(String name) {
        this.name = name;
    }

    /**
     * Adds a product with a price to this pricelist, if they aren't connected
     * Has a 0..* relationship with Product
     *
     * @param product, the choosen product
     * @param price,   the price of the product on this pricelist
     */
    public void addProductWithPrice(Product product, double price) {
        if (!productsWithPrice.containsKey(product)) {
            productsWithPrice.put(product, price);
            product.addPricelist(this, price);
        }
    }

    /**
     * Remove a product with price from this pricelist.
     * Has a 0..* relationship with Product
     *
     * @param product, the product you want to remove
     */
    public void removeProductWithPrice(Product product) {
        if (productsWithPrice.containsKey(product)) {
            productsWithPrice.remove(product);
            product.removePricelist(this);
        }
    }

    /**
     * update a price on a product in this pricelist
     *
     * @param product, the product you want to update
     * @param price,   the updated price
     * @throws IllegalArgumentException if product is not in pricelist
     */
    public void updateProductPrice(Product product, double price) {
        if (productsWithPrice.containsKey(product)) {
            productsWithPrice.put(product, price);
        }
    }

    /**
     * gets the price of a chosen product in this pricelist
     *
     * @param product, the product you want the price from
     * @return the price, from this pricelist, of the chosen product
     */
    public double getPriceOfProduct(Product product) {
        try {
            return productsWithPrice.get(product);
        } catch (Exception e) {
            throw new IllegalArgumentException("Produkt findes ikke i listen");
        }
    }

    /**
     * Returns the name of the pricelist
     *
     * @return name of the pricelist
     */
    public String getName() {
        return name;
    }

    /**
     * Get the product and its paired price.
     *
     * @return map of product as key and price as value
     */
    public Map<Product, Double> getProductsWithPrice() {
        return productsWithPrice;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

Chapter 44: model/product/Bundle.java

```java
package model.product;

import model.ProductGroup;

import java.util.HashMap;
import java.util.Map;

public class Bundle extends Product {

    // Link to the Product class (--> 0..*)
    private Map<Product, Integer> productsWithQuantity = new HashMap<>();

    /**
     * Creates a bundle which is almost like a product.
     * However it can hold a list of other products (and a quantity mapped to each).
     * Can't hold another bundle!
     *
     * @param name of the bundle
     * @param productGroup that the bundle belongs to
     * @param description of the bunlde
     */
    public Bundle(String name, ProductGroup productGroup, String description) {
        super(name, productGroup, description);
    }

    /**
     * The content of the bundle key is product and value is quantity
     * @return product as key associated with quantity as value
     */
    public HashMap<Product, Integer> getProductsWithQuantity() {
        return new HashMap<>(productsWithQuantity);
    }

    /**
     * adds a product to this bundle
     *
     * @param product, the product you want added to the bundle
     * @pre product is not of Bundle type
     */
    public void addProduct(Product product) {
        productsWithQuantity.put(product, productsWithQuantity.getOrDefault(product, 0) + 1)
            ↪ ;
    }

    /**
     * removes a product from this bundle
     *
     * @param product, the product you want to remove from the bundle
     */
    public void removeProduct(Product product) {
        if (productsWithQuantity.get(product) > 0)
            productsWithQuantity.put(product, productsWithQuantity.get(product) - 1);
    }

    @Override
    public String toString() {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(getName() + ": ");
        for (Product product : productsWithQuantity.keySet()) {
            stringBuilder.append(product.getName() + "(" + productsWithQuantity.get(product)
              ↪ + "), ");
        }
        // strips last 2 characters to avoid trailing ", "
        return stringBuilder.toString().substring(0, stringBuilder.toString().length() - 2);
    }
}
```

Chapter 45: model/product/Container.java

```java
package model.product;

import model.ProductGroup;
import model.enums.Unit;

public class Container extends Product {

    private double size;
    private Unit unit;

    /**
     * Creates a product that is a container with all parameters
     * used to set attributes.
     * A container is different from a product because it can have a size for a given unit.
     *      eg: 1 litre, 2 kg
     * @param name of the container
     * @param productGroup that the container belongs to
     * @param description of the container
     * @param size of the unit
     * @param unit of the size
     */
    public Container(String name, ProductGroup productGroup, String description, double size
        ↪ , Unit unit) {
        super(name, productGroup, description);
        this.size = size;
        this.unit = unit;
    }

    /**
     * Size is set with a given Unit in mind.
     * Remember to not compare apples to oranges!
     *
     * @return the size of the unit
     */
    public double getSize() {
        return size;
    }

    /**
     * @return the unit for the size
     */
    public Unit getUnit() {
        return unit;
    }
}
```

Chapter 46: model/product/Product.java

```java
package model.product;

import controller.Controller;
import model.Pricelist;
import model.ProductGroup;
import model.product.rentable.NotRentable;
import model.product.rentable.RentableStrategy;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;

public class Product implements Serializable {

        private final int id;
        private String name;
        private String description;
        private ProductGroup productGroup;
        private Set<Pricelist> pricelists = new HashSet<>();
        private RentableStrategy rentableStrategy;

        /**
         * Creates a product without a description.
         *
         * @pre productGroup != null
         *
         * @param name         of the product
         * @param productGroup where the product belongs to
         */
        public Product(String name, ProductGroup productGroup) {
                this.name = name;
                this.productGroup = productGroup;
                this.description = "";
                id = Controller.getInstance().getProductList().size() > 0 ? Controller.
                    ↪ getInstance().getProductList().get(Controller.getInstance().
                    ↪ getProductList().size() - 1).id + 1 : 1;
                this.rentableStrategy = new NotRentable();
        }

        /**
         * Overloads the default constructor.
         * The only difference is the description
         * @param name (see other construct doc)
         * @param productGroup (see other construct doc)
         * @param description of the product
         */
        public Product(String name, ProductGroup productGroup, String description) {
                this(name, productGroup);
                this.description = description;
        }

        /**
         * Adds this product to a chosen pricelist Has a bidirbidirectionalectional 0..*
            ↪ relationship
         * with Pricelist
         *
         * @param pricelist, the pricelist you want the product added to
         * @param price,     the price of the product on the chosen pricelist
         */
        public void addPricelist(Pricelist pricelist, double price) {
                if (!pricelists.contains(pricelist)) {
                        pricelist.addProductWithPrice(this, price);
                        pricelists.add(pricelist);
                }
        }

        /**
         * removes this product from a chosen pricelist Has a bidirectional 0..*
         * relationship with Pricelist
         *
         * @param pricelist, the pricelist the product are removed from
         */
        public void removePricelist(Pricelist pricelist) {
                if (pricelists.contains(pricelist)) {
                        pricelist.removeProductWithPrice(this);
                        pricelists.remove(pricelist);
                }
        }

        /**
         * all of the pricelists the product is a part of
         * @return all of the pricelists product is part of
         */
        public Set<Pricelist> getPricelists() {
                return pricelists;
        }

        /**
         * Replace the old rentableStrategy with a new one
         * @param rentableStrategy to replace existing
         */
        public void setRentableStrategy(RentableStrategy rentableStrategy) {
                this.rentableStrategy = rentableStrategy;
        }

        /**
         * name of the product
         * @return name of the product
         */
        public String getName() {
                return name;
        }

        /**
         *
         * @return the description of the product, can be empty string
```

```java
	 */
	public String getDescription() {
		return description;
	}

	/**
	 * the group of which this product belongs to
	 * @return the group which is responsible for the product
	 */
	public ProductGroup getProductGroup() {
		return productGroup;
	}

	/**
	 * The method from the rentable strategy pattern
	 * (see report)
	 * @return the deposit of the product
	 */
	public double getDeposit() {
		return rentableStrategy.getDeposit();
	}

	/**
	 * The rentable strategy which counts for this product (see report)
	 * @return rentable strategy this product uses
	 */
	public RentableStrategy getRentableStrategy() {
		return rentableStrategy;
	}

	/**
	 * the products id
	 * @return id of the product
	 */
	public int getId() {
		return id;
	}

	@Override
	public String toString() {
		return name;
	}
}
```

Chapter 47: model/product/Tour.java

```java
package model.product;

import model.ProductGroup;

public class Tour extends Product {

        /**
         * All parameters are similar to Product.
         * This class is only for identification (see the report)
         * @param name name of the tour
         * @param productGroup that the tour belongs to
         * @param description of the tour
         */
        public Tour(String name, ProductGroup productGroup, String description) {
                super(name, productGroup, description);
        }
}
```

Chapter 48: model/product/rentable/NotRentable.java

```java
package model.product.rentable;

import java.io.Serializable;

/**
 * Described in the report
 */
public class NotRentable implements RentableStrategy, Serializable {

        @Override
        public double getDeposit() {
                return -1;
        }
}
```

Chapter 49: model/product/rentable/Rentable.java

```java
package model.product.rentable;

import java.io.Serializable;

/**
 * Described in the report
 */
public class Rentable implements RentableStrategy, Serializable {
    private double deposit;

    /**
     * Must have a deposit amount, however nothing prohibits it from being 0 (free)
     * @param deposit amount to be paid for a rent
     */
    public Rentable(double deposit) {
        this.deposit = deposit;
    }

    @Override
    public double getDeposit() {
        return deposit;
    }

}
```

Chapter 50: model/product/rentable/RentableStrategy.java

```java
package model.product.rentable;

/**
 * Described in the report
 */
public interface RentableStrategy {
        double getDeposit();
}
```

Chapter 51: model/ProductGroup.java

```java
package model;

import model.enums.Unit;
import model.product.Bundle;
import model.product.Container;
import model.product.Product;
import model.product.Tour;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;

public class ProductGroup implements Serializable {
    private String name;
    private Set<Product> products = new HashSet<>();

    /**
     * Creates a product group without any products
     *
     * @param name of the product group
     */
    public ProductGroup(String name) {
        this.name = name;
    }

    /**
     * creates a product, and adds it to this productgroup
     *
     * @param name,        the name of the product
     * @param description, description of the product
     * @return the created product
     */
    public Product createProduct(String name, String description) {
        Product product = new Product(name, this, description);
        products.add(product);
        return product;
    }

    /**
     * Creates a container which is a subclass of product.
     * It's different because it can contain size and unit, which is appropriate for beer.
     * @param name of the container
     * @param description belonging to the container
     * @param size amount of the containers unit attribute
     * @param unit of the measured size eg. litre or kg
     * @return the newly created container product
     */
    public Container createContainerProduct(String name, String description, double size,
       ↪ Unit unit) {
        Container container = new Container(name, this, description, size, unit);
        products.add(container);
        return container;
    }

    /**
     * Creates a tour which is almost equivalent to product.
     * however, very desirable for identifying this particular type of product
     * - amount of people on the tour is stored in the sale
     *
     * @param name of the tour
     * @param description belonging to the tour
     * @return the newly created tour product
     */
    public Tour createTourProduct(String name, String description) {
        Tour tour = new Tour(name, this, description);
        products.add(tour);
        return tour;
    }

    /**
     * Creates a bundle product.
     * It's a little special because it can hold a list of other products.
     *
     * @param name of the bundle
     * @param description belonging to the bundle
     * @return the newly created bundle product
     */
    public Bundle createBundleProduct(String name, String description) {
        Bundle bundle = new Bundle(name, this, description);
        products.add(bundle);
        return bundle;
    }

    /**
     * deletes/removes the chosen product from the productgroup
     *
     * @param product, the product you want to delete
     */
    public void deleteProduct(Product product) {
        products.remove(product);
    }

    /**
     *  The current name of the productGroup
     *
     * @return the name of the productGroup
     */
    public String getName() {
        return name;
    }

    /**
     * The list of products that belongs to the productgroup
     * @return a set of products in the group
     */
    public Set<Product> getProducts() {
        return products;
    }

    @Override
    public String toString() {
        return name;
    }

}
```

Chapter 52: model/Rent.java

```java
package model;

import java.time.LocalDateTime;

public class Rent extends Sale {

    private String contactName;
    private String contactInformation;
    private LocalDateTime deliveryDateAndTime;
    private LocalDateTime returnDateAndTime;

    /**
     * Creates a rent sale.
     * - this is also used by the tour class
     * @param contactName of the person responsible for returning or the product(s)
     * @param contactInformation phonenumber, mail or any other means of contact information
     * @param deliveryDateAndTime the date and time which the rented product is either
       ↪ picked up or delivered
     * @param returnDateAndTime the date and time when the product must be returned
     */
    public Rent(String contactName, String contactInformation, LocalDateTime
      ↪ deliveryDateAndTime,
                LocalDateTime returnDateAndTime) {
        super();
        this.contactName = contactName;
        this.contactInformation = contactInformation;
        this.deliveryDateAndTime = deliveryDateAndTime;
        this.returnDateAndTime = returnDateAndTime;
    }


    /**
     * Gets the person attached to the rent
     * @return name of person responsible for returning the product
     */
    public String getContactName() {
        return contactName;
    }

    /**
     * Information about the person
     * @return information about the person
     */
    public String getContactInformation() {
        return contactInformation;
    }

    /**
```

```
     * When the product(s) are picked up or delivered.
     * If tour, when it's agreed to start.
     * @return date and time agreed
     */
    public LocalDateTime getDeliveryDateAndTime() {
        return deliveryDateAndTime;
    }

    /**
     * When product(s) must be returned
     * @return date and time agreed for return
     */
    public LocalDateTime getReturnDateAndTime() {
        return returnDateAndTime;
    }
}
```

Chapter 53: model/Sale.java

```java
package model;

import controller.Controller;
import model.enums.PaymentMethod;
import model.enums.SaleState;
import model.product.Product;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.LinkedList;
import java.util.List;

public class Sale implements Serializable {
    private final int id;
    private LocalDateTime timestamp;
    private double agreedPrice;
    private Payment payment;
    private List<SalesLine> salesLines = new LinkedList<>();
    private SaleState saleState;

    /**
     * Creates a sale, which is initiated with:
     *  - an id which calculated is (because of serializable) to be last id of known product
     ↪   + 1
     *  - (-1) as default agreed price, which is also discount. Uses -1 because of JavaFX
     *  - time of when the sale is initiated
     *  - the state of the sale is default initiated until set otherwise
     */
    public Sale() {
        id = Controller.getInstance().getSales().size() > 0 ? Controller.getInstance().
           ↪ getSales().get(Controller.getInstance().getSales().size() - 1).id + 1 : 1;
        agreedPrice = -1;
        timestamp = LocalDateTime.now();
        payment = new Payment();
        saleState = SaleState.INITIATED;
    }

    /**
     * Either changes a salesline if similar product and price is found.
     * Else it creates a new salesline.
     *
     * @param product that is either added or updated on salesline
     * @param quantity amount to modify with
     * @param price PER 1 quantity of the product
     * @return the created or updated salesline
     * @pre: product not null
     * @invarians: quantity see test report for further description
     */
    public SalesLine updateSalesLine(Product product, int quantity, double price) {
        // Check if saleLine exist
        boolean saleLineFound = false;
        int i = 0;
        while (i < salesLines.size() && !saleLineFound) {
            boolean productsSame = salesLines.get(i).getProduct().equals(product);
            boolean priceSame = salesLines.get(i).getPrice() == price;
            if (productsSame && priceSame)
                saleLineFound = true;
            else
                i++;
        }

        // If not found, create new salesLine
        SalesLine salesLine = !saleLineFound ? createSalesLine(product, quantity, price) :
           ↪ salesLines.get(i);

        // Add and return if new created
        if (!saleLineFound) {
            salesLines.add(salesLine);
            return salesLine;
        }

        // Update quantity
        salesLine.setQuantity(salesLine.getQuantity() + quantity);

        // If quantity == 0, remove
        if (salesLine.getQuantity() == 0)
            removeSalesLine(salesLine);

        return salesLine;
    }

    /**
     * Creates a new salesline. This method is private and is supposed to be called through
     * the public interface method updateSalesLine()
     * @param product to be added to the salesline
     * @param quantity of the product in the salesline
     * @param price price PER product
     * @return SalesLine
     */
    private SalesLine createSalesLine(Product product, int quantity, double price) {
        return new SalesLine(product, quantity, price);
    }

    /**
     * Removes salesline from sale.
     * @param salesLine to be removed
     */
    private void removeSalesLine(SalesLine salesLine) {
        salesLines.remove(salesLine);
    }

    /**
     * Calculate the total price of the sale.
     *
     * @return the total price of the sale.
     */
    public double totalSalePrice() {
        double total = 0;
        for (SalesLine salesLine : salesLines) {
            total += salesLine.getTotalLinePrice();
        }
        return total;
    }

    /**
     * Returns the payment attached to this sale.
     * A payment can hold multiple transfers, and is read this payment.
     * @return payment with transfers for this sale
     */
    public Payment getPayment() {
        return payment;
    }

    /**
     * Set's the timestamp for the sale,
     * unused in the application but used to make repeatable tests.
     * @param timestamp only use for testing!
     */
    public void setTimestamp(LocalDateTime timestamp) {
        this.timestamp = timestamp;
    }

    /**
     * Returns when the sale has been initiated or rather object has been created.
     * @return date and time of sale initiation
     */
    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    /**
     * Adds a transfer to the payment of a sale.
     *
     * @param paymentMethod, method of the payment.
     * @param amount,        the amount of money spend in the transfer.
     */
    public void addTransfer(PaymentMethod paymentMethod, double amount) {
        payment.addTransfer(paymentMethod, amount);
    }

    /**
     * Changes the state of the sale,
     * @param saleState to be changed to
     */
    public void setSaleState(SaleState saleState) {
        this.saleState = saleState;
    }

    /**
     * Gets the state of the sale
     * @return the state of the sale
     */
    public SaleState getSaleState() {
        return saleState;
    }

    /**
     * @return The id of the sale
     */
    public int getId() {
        return id;
    }

    /**
     * All the saleslines created through the sale.
     *
     * @return a list of saleslines
     */
    public List<SalesLine> getSalesLines() {
        return new LinkedList<>(salesLines);
    }

    /**
     * The agreed price for the entire sale AND thus also saleslines.
     * @return the agreed price for the entire sale
     */
    public double getAgreedPrice() {
        return agreedPrice;
    }

    /**
     * Set a agreed price (most likely a discount) for the entire sale
     * @param agreedPrice of the entire sale
     */
    public void setAgreedPrice(double agreedPrice) {
        this.agreedPrice = agreedPrice;
    }
}
```

Chapter 54: model/SalesLine.java

```java
package model;

import model.product.Product;
import java.io.Serializable;

public class SalesLine implements Serializable {

	private Product product;
	private int quantity;
	private double price;

	/**
	 * Creates a salesline with the values from the attributes
	 * @param product in the salesline
	 * @param quantity of the products in the salesline
	 * @param price per product in the salesline
	 */
	public SalesLine(Product product, int quantity, double price) {
		this.quantity = quantity;
		this.product = product;
		this.price = price;
	}

	/**
	 * the amount of products sold through this salesline
	 * @return quantity of products sold through this salesline
	 */
	public int getQuantity() {
		return quantity;
	}

	/**
	 * sets the amount of products to be sold through this salesline
	 * @param quantity of products sold in this salesline, overwrites old value
	 */
	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}

	/**
	 * the product in this salesline
	 * @return the product in the salesline
	 */
	public Product getProduct() {
		return product;
	}

	/**
```

```
	 * the price per product in this salesline
	 * @return price per product in the salesline
	 */
	public double getPrice() {
		return price;
	}

	/**
	 * Used if you want to set a new price to the product in the salesline.
	 *
	 * @param price, the new price
	 */
	public void setNewPrice(double price) {
		this.price = price;
	}

	/**
	 * Calculate the total price of the salesline
	 *
	 * @return the total price of the salesline
	 */
	public double getTotalLinePrice() {
		return price * quantity;
	}

	/**
	 * Calculate the total deposit of the salesline
	 *
	 * @return the total deposit of the salesline
	 */
	public double getTotalLineDeposit() {
		return product.getDeposit() * quantity;
	}

}
```

Chapter 55: model/SalesLine.java

```java
package model;

import model.product.Product;
import java.io.Serializable;

public class SalesLine implements Serializable {

        private Product product;
        private int quantity;
        private double price;

        /**
         * Creates a salesline with the values from the attributes
         * @param product in the salesline
         * @param quantity of the products in the salesline
         * @param price per product in the salesline
         */
        public SalesLine(Product product, int quantity, double price) {
                this.quantity = quantity;
                this.product = product;
                this.price = price;
        }

        /**
         * the amount of products sold through this salesline
         * @return quantity of products sold through this salesline
         */
        public int getQuantity() {
                return quantity;
        }

        /**
         * sets the amount of products to be sold through this salesline
         * @param quantity of products sold in this salesline, overwrites old value
         */
        public void setQuantity(int quantity) {
                this.quantity = quantity;
        }

        /**
         * the product in this salesline
         * @return the product in the salesline
         */
        public Product getProduct() {
                return product;
        }

        /**
```

```
         * the price per product in this salesline
         * @return price per product in the salesline
         */
        public double getPrice() {
                return price;
        }

        /**
         * Used if you want to set a new price to the product in the salesline.
         *
         * @param price, the new price
         */
        public void setNewPrice(double price) {
                this.price = price;
        }

        /**
         * Calculate the total price of the salesline
         *
         * @return the total price of the salesline
         */
        public double getTotalLinePrice() {
                return price * quantity;
        }

        /**
         * Calculate the total deposit of the salesline
         *
         * @return the total deposit of the salesline
         */
        public double getTotalLineDeposit() {
                return product.getDeposit() * quantity;
        }

}
```

Chapter 56: storage/Storage.java

```java
package storage;

import model.Pricelist;
import model.ProductGroup;
import model.Sale;
import model.product.Product;

import java.io.Serializable;
import java.util.LinkedList;
import java.util.List;

public class Storage implements Serializable {
	private static Storage storage;

	private List<ProductGroup> productGroups = new LinkedList<>();
	private List<Product> products = new LinkedList<>();
	private List<Pricelist> pricelists = new LinkedList<>();
	private List<Sale> sales = new LinkedList<>();

	public void addProductGroup(ProductGroup productGroup) {
		productGroups.add(productGroup);
	}

	public void addProduct(Product product) {
		products.add(product);
	}

	public void addPriceList(Pricelist pricelist) {
		pricelists.add(pricelist);
	}

	public void addSale(Sale sale) {
		sales.add(sale);
	}

	//
	↪ --------------------------------------------------------------------------------
	↪

	public List<ProductGroup> getProductGroupList() {
		return new LinkedList<>(productGroups);
	}

	public List<Product> getProductList() {
		return new LinkedList<>(products);
	}

	public List<Pricelist> getPricelists() {
```

```
                return new LinkedList<>(pricelists);
        }

        public List<Sale> getSales() {
                return new LinkedList<>(sales);
        }

        //
            ↪ --------------------------------------------------------------------------------
            ↪

        public static Storage getInstance() {
                if (storage == null) {
                        storage = new Storage();
                }
                return storage;
        }

}
```

Chapter 57: tests/Integration/UseCase4Test.java

```java
package Tests.Integration;

import controller.Controller;
import model.Pricelist;
import model.ProductGroup;
import model.Sale;
import model.SalesLine;
import model.product.Product;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class UseCase4Test {
    private Pricelist pricelist;
    private Product productChips;
    private Product productPeanuts;

    @Before
    public void setUp() {
        Controller.getInstance().initStorageOnly();
        pricelist = new Pricelist("Fredagsbar");
        ProductGroup productGroupSnacks = new ProductGroup("Snacks");
        productChips = new Product("Chips", productGroupSnacks);
        productPeanuts = new Product("Peanuts", productGroupSnacks);
        pricelist.addProductWithPrice(productChips, 15);
        pricelist.addProductWithPrice(productPeanuts, 10);
    }

    @Test
    public void IT1() {
        Sale sale = new Sale();
        sale.updateSalesLine(productChips, 1, pricelist.getPriceOfProduct(productChips));
        sale.updateSalesLine(productPeanuts, 1, pricelist.getPriceOfProduct(productPeanuts))
            ↪ ;
        assertEquals(2, sale.getSalesLines().size());
    }

    @Test
    public void IT2() {
        Sale sale = new Sale();
        SalesLine salesLine = sale.updateSalesLine(productChips, 1, pricelist.
            ↪ getPriceOfProduct(productChips));
        sale.updateSalesLine(productPeanuts, 1, pricelist.getPriceOfProduct(productPeanuts))
            ↪ ;
        sale.updateSalesLine(productChips, 2, pricelist.getPriceOfProduct(productChips));
        assertEquals(3, salesLine.getQuantity());
    }

    @Test
    public void IT3() {
        Sale sale = new Sale();
        SalesLine salesLine =  sale.updateSalesLine(productChips, 1, pricelist.getPriceOfProduct(productChips));
        sale.updateSalesLine(productPeanuts, 1, pricelist.getPriceOfProduct(productPeanuts));
        salesLine.setNewPrice(5);
        assertEquals(5, salesLine.getPrice(), 0.001);
        assertEquals(15, sale.totalSalePrice(), 0.001);
    }

}
```

Chapter 58: tests/Unit/UseCase4Test.java

```java
package Tests.Unit;

import controller.Controller;
import model.ProductGroup;
import model.product.Bundle;
import model.product.Product;
import org.junit.Before;
import org.junit.Test;

import java.util.HashMap;

import static org.junit.Assert.*;

public class BundleTest {
    private ProductGroup pgBar;
    private Bundle bundle;
    private Product p1;
    private Product p2;
    private Product p3;

    /*
        Da vi ikke overskriver metoder fra superklassen "Product" regner vi med at disse
           ↪ virker,
        såfremt Product består sine tests.
     */

    @Before
    public void setUp() throws Exception {
        Controller.getInstance().initStorageOnly();
        pgBar = new ProductGroup("Gaveæsker");
        bundle = new Bundle("Gaveæske 1", pgBar, "");
        p1 = new Product("Klosterbryg",pgBar);
        p2 = new Product("Forårsbryg",pgBar);
        p3 = new Product("Kellsberg",pgBar);
    }

    @Test
    public void addAndGetProductExpect1Quantity() {
        bundle.addProduct(p1);
        HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
           ↪ ();
        int quantity = bundleProductsAndQuantity.get(p1);
        assertEquals(1, quantity);
    }

    @Test
    public void addAndGetProductExpect1Quantity2() {
        bundle.addProduct(p1);
        bundle.addProduct(p1);
        HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
            ↪ ();
        int quantity = bundleProductsAndQuantity.get(p1);
        assertEquals(2, quantity);
    }

    @Test
    public void addMultipleProductsAndGetProductExpect1Quantity1() {
        bundle.addProduct(p1);
        bundle.addProduct(p3);
        HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
            ↪ ();
        int quantity = bundleProductsAndQuantity.get(p1);
        assertEquals(1, quantity);
    }

    @Test
    public void addProduct2TimesRemoveItOnceExpect1Quantity() {
        bundle.addProduct(p1);
        bundle.addProduct(p1);
        bundle.removeProduct(p1);
        HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
            ↪ ();
        int quantity = bundleProductsAndQuantity.get(p1);
        assertEquals(1, quantity);
    }

    @org.junit.Test
    public void getName() {
        Bundle bundleGift1 = new Bundle("Gaveæske 1", pgBar, "");
        assertEquals("Gaveæske 1", bundleGift1.getName());
    }
}
```

Chapter 59: tests/Unit/ContainerTest.java

```java
package Tests.Unit;

import controller.Controller;
import model.ProductGroup;
import model.enums.Unit;
import model.product.Container;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class ContainerTest {
    private Container container;
    private ProductGroup pgBar;

    @Before
    public void setUp() throws Exception {
        Controller.getInstance().initStorageOnly();
        pgBar = new ProductGroup("Øl");
        container = new Container("Klosterbryg", pgBar, "", 60, Unit.cl);
    }


    @Test
    public void getSize() {
        assertEquals(60, container.getSize(), 0.01);
    }

    @Test
    public void getUnit() {
        assertEquals(Unit.cl, container.getUnit());
    }
}
```

Chapter 60: tests/Unit/PaymentTest.java

```java
package Tests.Unit;

import model.Payment;
import model.enums.PaymentMethod;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class PaymentTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void payment0Transfer() {
        Payment payment = new Payment();
        assertEquals(payment.calcTotal(), 0, 0.001);
    }

    @Test
    public void payment1TransferCash() {
        // 1 payment with cash
        Payment  payment = new Payment();
        payment.addTransfer(PaymentMethod.CASH, 200);
        assertEquals(payment.calcTotal(), 200, 0.001);
    }

    @Test
    public void payment2TransferCash() {
        // 1 payment with cash
        Payment payment = new Payment();
        payment.addTransfer(PaymentMethod.CASH,200);
        payment.addTransfer(PaymentMethod.CASH,200);
        assertEquals(payment.calcTotal(), 400, 0.001);
    }

    @Test
    public void payment2TransferCashAndPaymentCard() {
        // 2 payments total, 1 with cash and 1 with card
        Payment payment = new Payment();
        payment.addTransfer(PaymentMethod.CASH,200);
        payment.addTransfer(PaymentMethod.PAYMENTCARD,200);
        assertEquals(payment.calcTotal(), 400, 0.001);
    }


    @Test
    public void paymentNegativeForReturnSale() {
        // 2 payments total, 1 with cash and 1 with card
        Payment payment = new Payment();
        payment.addTransfer(PaymentMethod.CASH,-200);
        assertEquals(payment.calcTotal(), -200, 0.001);
    }
}
```

Chapter 61: tests/Unit/PricelistTest.java

```java
package Tests.Unit;

import model.Pricelist;
import model.ProductGroup;
import model.enums.Unit;
import model.product.Container;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

public class PricelistTest {
    private ProductGroup pgBar;
    private Container klosterbryg;
    private Container kellsberg;
    private Pricelist pricelist;

    @Before
    public void setUp() throws Exception {
        pgBar = new ProductGroup("Øl");
        klosterbryg = new Container("Klosterbryg", pgBar, "", 0.60, Unit.cl);
        kellsberg = new Container("Kellsberg", pgBar, "", 60, Unit.cl);
        pricelist = new Pricelist("Bar");
    }

    @Test
    public void addAndGetProductWithPrice() {
        pricelist.addProductWithPrice(kellsberg, 70);
        assertTrue(pricelist.getProductsWithPrice().containsKey(kellsberg));
    }

    @Test
    public void removeProductWithPrice() {
        pricelist.addProductWithPrice(kellsberg, 70);
        pricelist.removeProductWithPrice(kellsberg);
        assertFalse(pricelist.getProductsWithPrice().containsKey(kellsberg));
    }

    @Test
    public void updateProductPriceAndPriceOfProduct() {
        pricelist.addProductWithPrice(kellsberg, 70);
        pricelist.updateProductPrice(kellsberg, 90);
        assertEquals(90,pricelist.getPriceOfProduct(kellsberg), 0.001);
    }

    @Test(expected = IllegalArgumentException.class)
    public void getPriceOfNotExistingProduct() {
        pricelist.getPriceOfProduct(klosterbryg);
    }

    @Test
    public void getName() {
        Pricelist pricelist = new Pricelist("Rundvisning");
        assertEquals("Rundvisning", pricelist.getName());
    }

}
```

Chapter 62: tests/Unit/ProductTest.java

```java
package Tests.Unit;

import controller.Controller;
import model.Pricelist;
import model.ProductGroup;
import model.product.Product;

import static org.junit.Assert.*;

public class ProductTest {
    private Pricelist plBar;
    private Product product;

    @org.junit.Before
    public void setUp() {
        Controller.getInstance().initStorageOnly();
        ProductGroup pgBar = new ProductGroup("Øl");
        plBar = new Pricelist("Bar pricelist");
        product = new Product("Klosterbryg", pgBar);
    }

    @org.junit.Test
    public void addAndRemovePriceList() {
        /*
        Det kan diskuteres hvor denne metode bør testes, om det er en Integrationstest.
        Vi tester den her, og betragter metoderne add og remove der opretholder
        den dobbeltrettede associering og som sideeffekt påvirker Pricelist.
        */

        // Add virker dobbeltrettet
        product.addPricelist(plBar, 70.0);
        assertEquals(70.0, plBar.getPriceOfProduct(product), 0.001);
        // Remove virker dobbeltrettet
        product.removePricelist(plBar);
        assertFalse(plBar.getProductsWithPrice().containsKey(product));
    }

    @org.junit.Test
    public void getName() {
        assertEquals("Klosterbryg", product.getName());
    }
}
```

Chapter 63: tests/Unit/RentAndTourTest.java

```java
package Tests.Unit;

import controller.Controller;
import model.ProductGroup;
import model.Rent;
import model.product.Tour;
import model.product.rentable.Rentable;
import org.junit.Before;
import org.junit.Test;

import java.time.LocalDateTime;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class RentAndTourTest {
    private LocalDateTime date;
    private String contactPersonName;
    private String contactPersonNumber;
    private Tour tour;
    private Rent rent;

    @Before
    public void setUp() {
        Controller.getInstance().initStorageOnly();
        //TODO : UPDATE UPDATE UPDATE
        ProductGroup pgBar = new ProductGroup("Rundvisninger");
        date = LocalDateTime.of(2020, 01, 01, 12, 30);
        contactPersonName = "Mikael";
        contactPersonNumber = "12345678";
        tour = new Tour("Rundvisning 100kr", pgBar, "");
        tour.setRentableStrategy(new Rentable(100));
        rent = new Rent(contactPersonName, contactPersonNumber, date, date);
    }

    @Test
    public void tourRentableTest() {
        assertTrue(tour.getRentableStrategy() instanceof Rentable);
    }

    @Test
    public void getStartDateAndTime() {
        assertEquals(date, rent.getDeliveryDateAndTime());
    }

    @Test
    public void getContactPersonName() {
        assertEquals(contactPersonName, rent.getContactName());
    }

    @Test
    public void getContactPersonNumber() {
        assertEquals(contactPersonNumber, rent.getContactInformation());
    }
}
```

Chapter 64: tests/Unit/SalesLineTest.java

```java
package Tests.Unit;

import model.ProductGroup;
import model.SalesLine;
import model.product.Bundle;
import model.product.Product;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

public class SalesLineTest {
    private Product p1;

    @Before
    public void setUp() throws Exception {
        ProductGroup pgBundles = new ProductGroup("Gavepakker");
        p1 = new Bundle("Valentinspakke til Karsten", pgBundles, "");
    }

    @Test
    public void setAndGetQuantity() {
        SalesLine salesLine = new SalesLine(p1, 1, 200);
        assertEquals(1, salesLine.getQuantity());
        salesLine.setQuantity(5);
        assertEquals(5, salesLine.getQuantity());
    }


    @Test
    public void getProduct() {
        SalesLine salesLine = new SalesLine(p1, 1, 200);
        assertEquals(p1, salesLine.getProduct());
    }

    @Test
    public void totalLinePrice1Product() {
        SalesLine salesLine = new SalesLine(p1, 1, 200);
        assertEquals(200, salesLine.getTotalLinePrice(), 0.001);
    }

    @Test
    public void totalLinePrice2Products() {
        SalesLine salesLine = new SalesLine(p1, 2, 200);
        assertEquals(400, salesLine.getTotalLinePrice(), 0.001);
    }

    @Test
    public void setNewPrice() {
        SalesLine salesLine = new SalesLine(p1, 1, 200);
        salesLine.setNewPrice(100);
        assertEquals(100, salesLine.getTotalLinePrice(), 0.001);
    }


}
```

Chapter 65: tests/Unit/SaleTest.java

```java
package Tests.Unit;

import controller.Controller;
import model.ProductGroup;
import model.Sale;
import model.SalesLine;
import model.enums.PaymentMethod;
import model.product.Product;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

public class SaleTest {
    private Product p1;
    private Product p2;

    @Before
    public void setUp() {
        Controller.getInstance().initStorageOnly();
        ProductGroup pgBundles = new ProductGroup("Snacks");
        p1 = new Product("Chips", pgBundles);
        p2 = new Product("Peanuts", pgBundles);
    }

    // Metode: updateSales(...)

    @Test
    public void TC1addToSale1SalesLine() {
        Sale sale = new Sale();
        SalesLine salesLine = sale.updateSalesLine(p1, 1, 10);
        assertTrue(sale.getSalesLines().contains(salesLine));
    }

    @Test
    public void TC2totalSalePriceAddToSale() {
        Sale sale = new Sale();
        sale.updateSalesLine(p1, 1, 10);
        assertEquals(10, sale.totalSalePrice(), 0.001);
    }

    @Test
    public void TC3addSameProductTwiceToSalesLine() {
        Sale sale = new Sale();
        sale.updateSalesLine(p1, 1, 10);
        sale.updateSalesLine(p1, 1, 10);
        assertEquals(1, sale.getSalesLines().size());
        assertEquals(20, sale.totalSalePrice(), 0.001);
    }

    @Test
    public void TC4addSameProductTwiceDifferentPriceToSalesLine() {
        Sale sale = new Sale();
        sale.updateSalesLine(p1, 1, 10);
        sale.updateSalesLine(p1, 1, 15);
        assertEquals(2, sale.getSalesLines().size());
        assertEquals(25, sale.totalSalePrice(), 0.001);
    }

    @Test
    public void TC5addTwoProductsToSalesLine() {
        Sale sale = new Sale();
        sale.updateSalesLine(p1, 1, 10);
        sale.updateSalesLine(p2, 1, 5);
        assertEquals(2, sale.getSalesLines().size());
        assertEquals(15, sale.totalSalePrice(), 0.001);
    }

    @Test
    public void TC6ifQuantityOfSalesIs0Remove() {
        Sale sale = new Sale();
        SalesLine salineLine1 = sale.updateSalesLine(p1, 1, 10);
        assertEquals(1, sale.getSalesLines().size());
        sale.updateSalesLine(p1, -1, 10);
        assertEquals(0, sale.getSalesLines().size());
        assertFalse(sale.getSalesLines().contains(salineLine1));
        assertEquals(0, sale.totalSalePrice(), 0.001);
    }

    // Metode: addTransfter(...)

    @Test
    public void TC7addTransfer() {
        Sale sale = new Sale();
        sale.addTransfer(PaymentMethod.CASH, 10);
        assertTrue(sale.getPayment().getTransfers().containsKey(PaymentMethod.CASH));
        assertEquals(10, sale.getPayment().calcTotal(), 0.0);
    }

    @Test
    public void TC8add2TransferDifferentPaymentMethods() {
        Sale sale = new Sale();
        sale.addTransfer(PaymentMethod.CASH, 10);
        sale.addTransfer(PaymentMethod.MOBILEPAY, 15);
        assertTrue(sale.getPayment().getTransfers().containsKey(PaymentMethod.CASH));
        assertTrue(sale.getPayment().getTransfers().containsKey(PaymentMethod.MOBILEPAY));
        assertEquals(25, sale.getPayment().calcTotal(), 0.0);
    }

}
```

Chapter 66: tests/Unit/UnitSuite.java

```java
package Tests.Unit;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ProductTest.class, BundleTest.class, ContainerTest.class, RentAndTourTest.
    ↪ class, PaymentTest.class, PricelistTest.class, SalesLineTest.class, SaleTest.class})
public class UnitSuite {

}
```