

Contents

1	controller/Controller.java	4
2	gui/administrationpane/AdministrationPane.java	15
3	gui/administrationpane/NewProductGroupPane.java	17
4	gui/administrationpane/NewProductPane.java	19
5	gui/administrationpane/PricelistPane.java	28
6	gui/css/app.css	33
7	gui/helpers/Calendar.java	35
8	gui/helpers/DateIntervalPicker.java	36
9	gui/helpers/ProductAndPriceHelper.java	37
10	gui/helpers/ProductDatePicker.java	39
11	gui/helpers/TableViewHelper.java	41
12	gui/helpers/TimePicker.java	43
13	gui/helpers/ValidationHelper.java	44
14	gui/HomePane.java	46
15	gui/LeftSideBarInterface.java	48
16	gui/LeftSideBar.java	49
17	gui/MainApp.java	51
18	gui/Navigation.java	53
19	gui/overviewpane/OverviewPane.java	56
20	gui/overviewpane/PricelistOverviewPane.java	58
21	gui/overviewpane/ProductOverviewPane.java	60
22	gui/overviewpane/RentOverviewPane.java	64
23	gui/overviewpane/SaleOverviewPane.java	68
24	gui/overviewpane/TourOverviewPane.java	70
25	gui/salepane/PaymentPane.java	72

26	gui/salepane/RentInformationPane.java	78
27	gui/salepane/SaleOptionsPane.java	82
28	gui/salepane/SalePane.java	83
29	gui/salepane/SelectOngoingSale.java	90
30	gui/salepane/SelectPricelistPane.java	93
31	gui/salepane/TableViewSalesLines.java	95
32	gui/salepane/TableViewSalesLines.java	97
33	gui/StatisticPane.java	99
34	gui/statistics/ExportPane.java	101
35	gui/statistics/FilterPane.java	103
36	gui/statistics/GraphsPane.java	107
37	gui/statistics/PieChartExtended.java	110
38	gui/statistics/ReportTextPane.java	111
39	model/enums/PaymentMethod.java	113
40	model/enums/SaleSate.java	114
41	model/enums/Unit.java	115
42	model/Payment.java	116
43	model/Pricelist.java	118
44	model/product/Bundle.java	120
45	model/product/Container.java	122
46	model/product/Product.java	123
47	model/product/Tour.java	126
48	model/product/rentable/NotRentable.java	127
49	model/product/rentable/Rentable.java	128
50	model/product/rentable/RentableStrategy.java	129
51	model/ProductGroup.java	130
52	model/Rent.java	133
53	model/Sale.java	135
54	model/SalesLine.java	139
55	model/SalesLine.java	141
56	storage/Storage.java	143

57 tests/Integration/UseCase4Test.java	145
58 tests/Unit/UseCase4Test.java	147
59 tests/Unit/ContainerTest.java	149
60 tests/Unit/PaymentTest.java	150
61 tests/Unit/PricelistTest.java	152
62 tests/Unit/ProductTest.java	154
63 tests/Unit/RentAndTourTest.java	155
64 tests/Unit/SalesLineTest.java	157
65 tests/Unit/SaleTest.java	159
66 tests/Unit/UnitSuite.java	161

Chapter 1: controller/Controller.java

```
1 package controller;
2
3 import model.Pricelist;
4 import model.ProductGroup;
5 import model.Rent;
6 import model.Sale;
7 import model.enums.PaymentMethod;
8 import model.enums.SaleState;
9 import model.enums.Unit;
10 import model.product.Bundle;
11 import model.product.Container;
12 import model.product.Product;
13 import model.product.Tour;
14 import model.product.rentable.Rentable;
15 import storage.Storage;
16 import java.io.*;
17 import java.time.LocalDateTime;
18 import java.util.List;
19 import java.util.Random;
20 import java.util.stream.Collectors;
21
22 public class Controller {
23
24     private static Controller controller;
25     private static Storage storage;
26
27     /**
28      * Creates and returns a new productGroup
29      *
30      * @param name the name of the productGroup
31      * @return created productGroup
32      */
33     public ProductGroup createProductGroup(String name) {
34         ProductGroup productGroup = new ProductGroup(name);
35         storage.addProductGroup(productGroup);
36         return productGroup;
37     }
38
39     /**
40      * Creates and returns a new product
41      *
42      * @param productGroup the productGroup of the product
43      * @param name          the name of the product
44      * @param description   the description of the product
45      * @return created product
46      */
47     public Product createProductInProductGroup(ProductGroup productGroup, String name,
48         ↪ String description) {
```

```

48     Product product = productGroup.createProduct(name, description);
49     storage.addProduct(product);
50     return product;
51 }
52
53 /**
54  * Creates and returns a new container
55  *
56  * @param productGroup the productGroup of the product
57  * @param name          the name of the container
58  * @param description   the description of the container
59  * @return created container
60  */
61 public Container createContainerProductInProductGroup(ProductGroup productGroup, String
    ↪ name, String description, double size, Unit unit) {
62     Container container = productGroup.createContainerProduct(name, description, size,
    ↪ unit);
63     storage.addProduct(container);
64     return container;
65 }
66
67 /**
68  * Creates and returns a new tour
69  *
70  * @param productGroup the productGroup of the tour
71  * @param name          the name of the tour
72  * @param description   the description of the tour
73  * @return created tour
74  */
75 public Tour createTourProductInProductGroup(ProductGroup productGroup, String name,
    ↪ String description) {
76     Tour tour = productGroup.createTourProduct(name, description);
77     storage.addProduct(tour);
78     return tour;
79 }
80
81 /**
82  * Creates and returns a new bundle
83  *
84  * @param productGroup the productGroup of the bundle
85  * @param name          the name of the bundle
86  * @param description   the description of the bundle
87  * @return created bundle
88  */
89 public Bundle createBundleProductInProductGroup(ProductGroup productGroup, String name,
    ↪ String description) {
90     Bundle bundle = productGroup.createBundleProduct(name, description);
91     storage.addProduct(bundle);
92     return bundle;
93 }
94
95 /**
96  * Creates and returns a new pricelist
97  *
98  * @param name the name of the pricelist
99  * @return created pricelist
100  */
101 public Pricelist createPricelist(String name) {

```

```

102     Pricelist pricelist = new Pricelist(name);
103     storage.addPriceList(pricelist);
104     return pricelist;
105 }
106
107 /**
108  * Adds a product with price to pricelist
109  *
110  * @param pricelist the pricelist to be added to
111  * @param product   the product to be added
112  * @param price     the price of the product to be added
113  */
114 public void addProductWithPriceToPricelist(Pricelist pricelist, Product product, double
    ↪ price) {
115     pricelist.addProductWithPrice(product, price);
116 }
117
118 /**
119  * Creates and returns a new sale
120  *
121  * @return created sale
122  */
123 public Sale createSale() {
124     Sale sale = new Sale();
125     storage.addSale(sale);
126     return sale;
127 }
128
129 /**
130  * Creates and returns a new bundle
131  *
132  * @param contactName      the contact name of the rent
133  * @param contactInformation the contact information of the rent
134  * @param deliveryDateAndTime the delivery date and time of the rent
135  * @param returnDateAndTime the return date and time of the rent
136  * @return created rent
137  */
138 public Rent createRent(String contactName, String contactInformation, LocalDateTime
    ↪ deliveryDateAndTime,
139                        LocalDateTime returnDateAndTime) {
140     Rent rent = new Rent(contactName, contactInformation, deliveryDateAndTime,
    ↪ returnDateAndTime);
141     storage.addSale(rent);
142     return rent;
143 }
144
145 //
    ↪ -----
    ↪
146
147 /**
148  * Get list of product groups from storage
149  *
150  * @return list of product groups
151  */
152 public List<ProductGroup> getProductGroupList() {
153     return storage.getProductGroupList();
154 }

```

```

155
156 /**
157  * Get list of products from storage
158  *
159  * @return list of products
160  */
161 public List<Product> getProductList() {
162     return storage.getProductList();
163 }
164
165 /**
166  * Get list of pricelists from storage
167  *
168  * @return list of pricelist
169  */
170 public List<Pricelist> getPricelists() {
171     return storage.getPricelists();
172 }
173
174 /**
175  * Get list of sales from storage
176  *
177  * @return list of sales
178  */
179 public List<Sale> getSales() {
180     return storage.getSales();
181 }
182
183 /**
184  * Get list of sales from storage that is instance of Rent,
185  * but does not contain product with instance of Tour in salesLines
186  *
187  * @return list of sales
188  */
189 public List<Sale> getRents() {
190     return getSales().stream()
191         .filter(sale -> sale instanceof Rent)
192         .filter(sale -> sale.getSalesLines().stream().noneMatch(salesLine -> !(
193             ↪ salesLine.getProduct() instanceof Tour)))
194         .collect(Collectors.toList());
195 }
196
197 /**
198  * Get list of sales from storage that is instance of Rent,
199  * and contains a salesLine with a product instance of Tour
200  *
201  * @return list of sales
202  */
203 public List<Sale> getTours() {
204     return getSales().stream()
205         .filter(sale -> sale instanceof Rent)
206         .filter(sale -> sale.getSalesLines().stream().anyMatch(salesLine ->
207             ↪ salesLine.getProduct() instanceof Tour))
208         .collect(Collectors.toList());
209 }
210 //
211 ↪
212 ↪

```

```

209
210 /**
211  * Lazy loaded singleton controller
212  *
213  * @return controller
214  */
215 public static Controller getInstance() {
216     if (controller == null)
217         controller = new Controller();
218     return controller;
219 }
220
221 //
222
223 public static final String PATH = "data/Serilizabledata.ser";
224
225 public void saveToFile(String path) {
226     try (FileOutputStream fileOutputStream = new FileOutputStream(path);
227         ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream
228             ↪ )) {
229
230         objectOutputStream.writeObject(storage);
231         System.out.println("Saved to file: " + PATH);
232
233     } catch (IOException e) {
234         e.printStackTrace();
235     }
236
237 public void readFromFile(String path) {
238     try (FileInputStream fileInputStream = new FileInputStream(path);
239         ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream)) {
240
241         Object object = objectInputStream.readObject();
242         if (object instanceof Storage)
243             storage = (Storage) object;
244         System.out.println("Read from file: " + PATH);
245
246     } catch (IOException | ClassNotFoundException e) {
247         System.out.println("Could not read file, recreating data from initStorage");
248         controller.initStorage();
249     }
250 }
251
252 //
253
254 /**
255  * For testing purposes, dont need to generate date
256  */
257 public void initStorageOnly() {
258     storage = Storage.getInstance();
259 }
260
261 public void initStorage() {

```



```

262     storage = Storage.getInstance();
263
264     Pricelist pricelist1 = createPricelist("Fredagsbar");
265     Pricelist pricelist2 = createPricelist("Butik");
266     Pricelist pricelist3 = createPricelist("Rundvisninger");
267
268     // Klippekort
269     ProductGroup productGroup1 = createProductGroup("Klippekort");
270     Product product1 = createProductInProductGroup(productGroup1, "Klippekort, 4 klip",
271         ↪ "");
272
273     addProductWithPriceToPricelist(pricelist1, product1, 130);
274     addProductWithPriceToPricelist(pricelist2, product1, 130);
275
276     // Flaske
277     ProductGroup productGroup2 = createProductGroup("Flaske");
278     Product product2 = createContainerProductInProductGroup(productGroup2, "Klosterbryg"
279         ↪ , "", 60, Unit.cl);
280     Product product3 = createContainerProductInProductGroup(productGroup2, "Sweet
281         ↪ Georgia Brown", "", 60, Unit.cl);
282     Product product4 = createContainerProductInProductGroup(productGroup2, "Extra
283         ↪ Pilsner", "", 60, Unit.cl);
284     Product product5 = createContainerProductInProductGroup(productGroup2, "Celebration"
285         ↪ , "", 60, Unit.cl);
286     Product product6 = createContainerProductInProductGroup(productGroup2, "Blondie", ""
287         ↪ , 60, Unit.cl);
288     Product product7 = createContainerProductInProductGroup(productGroup2, "Forårsbryg",
289         ↪ "", 60, Unit.cl);
290     Product product8 = createContainerProductInProductGroup(productGroup2, "India Pale
291         ↪ Ale", "", 60, Unit.cl);
292     Product product9 = createContainerProductInProductGroup(productGroup2, "Julebryg", "
293         ↪ ", 60, Unit.cl);
294     Product product10 = createContainerProductInProductGroup(productGroup2, "Juletønden"
295         ↪ , "", 60, Unit.cl);
296     Product product11 = createContainerProductInProductGroup(productGroup2, "Old Strong
297         ↪ Ale", "", 60, Unit.cl);
298     Product product12 = createContainerProductInProductGroup(productGroup2, "Fregatten
299         ↪ Jylland", "", 60, Unit.cl);
300     Product product13 = createContainerProductInProductGroup(productGroup2, "Imperial
301         ↪ Stout", "", 60, Unit.cl);
302     Product product14 = createContainerProductInProductGroup(productGroup2, "Tribute", "
303         ↪ ", 60, Unit.cl);
304     Product product15 = createContainerProductInProductGroup(productGroup2, "Black
305         ↪ Monster", "", 60, Unit.cl);
306
307     for (int i = 1; i < 15; i++) {
308         Product product = storage.getProductList().get(i);
309         addProductWithPriceToPricelist(pricelist1, product, 70);
310         addProductWithPriceToPricelist(pricelist2, product, 36);
311     }
312     addProductWithPriceToPricelist(pricelist1, product15, 100);
313     addProductWithPriceToPricelist(pricelist2, product15, 60);
314
315     // Fadøl, 40 cl
316     ProductGroup productGroup3 = createProductGroup("Fadøl, 40 cl");
317     Product product16 = createContainerProductInProductGroup(productGroup3, "Klosterbryg
318         ↪ ", "", 40, Unit.cl);

```

```

303 Product product17 = createContainerProductInProductGroup (productGroup3, "Jazz
    ↪ Classic", "", 40, Unit.cl);
304 Product product18 = createContainerProductInProductGroup (productGroup3, "Extra
    ↪ Pilsner", "", 40, Unit.cl);
305 Product product19 = createContainerProductInProductGroup (productGroup3, "Celebration
    ↪ ", "", 40, Unit.cl);
306 Product product20 = createContainerProductInProductGroup (productGroup3, "Blondie", "
    ↪ ", 40, Unit.cl);
307 Product product21 = createContainerProductInProductGroup (productGroup3, "Forårsbryg"
    ↪ , "", 40, Unit.cl);
308 Product product22 = createContainerProductInProductGroup (productGroup3, "India Pale
    ↪ Ale", "", 40, Unit.cl);
309 Product product23 = createContainerProductInProductGroup (productGroup3, "Julebryg",
    ↪ "", 40, Unit.cl);
310 Product product24 = createContainerProductInProductGroup (productGroup3, "Imperial
    ↪ Stout", "", 40, Unit.cl);
311 Product product25 = createContainerProductInProductGroup (productGroup3, "Special", "
    ↪ ", 40, Unit.cl);
312
313 for (int i = 15; i < 25; i++) {
314     Product product = storage.getProductList().get(i);
315     addProductWithPriceToPricelist (pricelist2, product, 38);
316 }
317
318 // Spiritus
319 ProductGroup productGroup4 = createProductGroup("Spiritus");
320 Product product26 = createContainerProductInProductGroup (productGroup4, "Spirit of
    ↪ Aarhus", "", 70, Unit.cl);
321 Product product27 = createContainerProductInProductGroup (productGroup4, "SOA med
    ↪ pind", "", 70, Unit.cl);
322 Product product28 = createContainerProductInProductGroup (productGroup4, "Whisky", ""
    ↪ , 70, Unit.cl);
323 Product product29 = createContainerProductInProductGroup (productGroup4, "Liquor of
    ↪ Aarhus", "", 70, Unit.cl);
324
325 addProductWithPriceToPricelist (pricelist1, product26, 300);
326 addProductWithPriceToPricelist (pricelist2, product26, 300);
327 addProductWithPriceToPricelist (pricelist1, product27, 350);
328 addProductWithPriceToPricelist (pricelist2, product27, 350);
329 addProductWithPriceToPricelist (pricelist1, product28, 500);
330 addProductWithPriceToPricelist (pricelist2, product28, 500);
331 addProductWithPriceToPricelist (pricelist1, product29, 175);
332 addProductWithPriceToPricelist (pricelist2, product29, 175);
333
334 // Fustage
335 ProductGroup productGroup5 = createProductGroup("Fustage");
336 Product product30 = createContainerProductInProductGroup (productGroup5, "Klosterbryg
    ↪ ", "", 20, Unit.l);
337 product30.setRentableStrategy(new Rentable(200));
338 Product product31 = createContainerProductInProductGroup (productGroup5, "Jazz
    ↪ Classic", "", 25, Unit.l);
339 product31.setRentableStrategy(new Rentable(200));
340 Product product32 = createContainerProductInProductGroup (productGroup5, "Extra
    ↪ Pilsner", "", 25, Unit.l);
341 product32.setRentableStrategy(new Rentable(200));
342 Product product33 = createContainerProductInProductGroup (productGroup5, "Celebration
    ↪ ", "", 20, Unit.l);
343 product33.setRentableStrategy(new Rentable(200));

```

```

344 Product product34 = createContainerProductInProductGroup(productGroup5, "Blondie", "
    ↪ ", 25, Unit.l);
345 product34.setRentableStrategy(new Rentable(200));
346 Product product35 = createContainerProductInProductGroup(productGroup5, "Forårsbryg"
    ↪ ", "", 20, Unit.l);
347 product35.setRentableStrategy(new Rentable(200));
348 Product product36 = createContainerProductInProductGroup(productGroup5, "India Pale
    ↪ Ale", "", 20, Unit.l);
349 product36.setRentableStrategy(new Rentable(200));
350 Product product37 = createContainerProductInProductGroup(productGroup5, "Julebryg", "
    ↪ ", 20, Unit.l);
351 product37.setRentableStrategy(new Rentable(200));
352 Product product38 = createContainerProductInProductGroup(productGroup5, "Imperial
    ↪ Stout", "", 20, Unit.l);
353 product38.setRentableStrategy(new Rentable(200));
354
355 addProductWithPriceToPricelist(pricelist2, product30, 775);
356 addProductWithPriceToPricelist(pricelist2, product31, 625);
357 addProductWithPriceToPricelist(pricelist2, product32, 575);
358 addProductWithPriceToPricelist(pricelist2, product33, 775);
359 addProductWithPriceToPricelist(pricelist2, product34, 700);
360 addProductWithPriceToPricelist(pricelist2, product35, 775);
361 addProductWithPriceToPricelist(pricelist2, product36, 775);
362 addProductWithPriceToPricelist(pricelist2, product37, 775);
363 addProductWithPriceToPricelist(pricelist2, product38, 775);
364
365 // Kulsyre
366 ProductGroup productGroup6 = createProductGroup("Kulsyre");
367 Product product39 = createContainerProductInProductGroup(productGroup6, "Kulsyre", "
    ↪ ", 6, Unit.kg);
368 product39.setRentableStrategy(new Rentable(1000));
369 Product product40 = createContainerProductInProductGroup(productGroup6, "Kulsyre", "
    ↪ ", 4, Unit.kg);
370 product40.setRentableStrategy(new Rentable(1000));
371 Product product41 = createContainerProductInProductGroup(productGroup6, "Kulsyre", "
    ↪ ", 10, Unit.kg);
372 product41.setRentableStrategy(new Rentable(1000));
373
374 addProductWithPriceToPricelist(pricelist1, product39, 400);
375 addProductWithPriceToPricelist(pricelist2, product39, 400);
376
377 // Malt
378 ProductGroup productGroup7 = createProductGroup("Malt");
379 Product product42 = createContainerProductInProductGroup(productGroup7, "Malt søk", "
    ↪ ", 25, Unit.kg);
380
381 addProductWithPriceToPricelist(pricelist2, product42, 300);
382
383 // Beklædning
384 ProductGroup productGroup8 = createProductGroup("Beklædning");
385 Product product43 = createProductInProductGroup(productGroup8, "T-shirt", "");
386 Product product44 = createProductInProductGroup(productGroup8, "Polo", "");
387 Product product45 = createProductInProductGroup(productGroup8, "Cap", "");
388
389 addProductWithPriceToPricelist(pricelist1, product43, 70);
390 addProductWithPriceToPricelist(pricelist2, product43, 70);
391 addProductWithPriceToPricelist(pricelist1, product44, 100);
392 addProductWithPriceToPricelist(pricelist2, product44, 100);

```

```

393 addProductWithPriceToPricelist(pricelist1, product45, 30);
394 addProductWithPriceToPricelist(pricelist2, product45, 30);
395
396 // Anlæg
397 ProductGroup productGroup9 = createProductGroup("Anlæg");
398 Product product46 = createProductInProductGroup(productGroup9, "1-hane", "");
399 product46.setRentableStrategy(new Rentable(250));
400 Product product47 = createProductInProductGroup(productGroup9, "2-haner", "");
401 product47.setRentableStrategy(new Rentable(400));
402 Product product48 = createProductInProductGroup(productGroup9, "Bar med flere haner"
    ↪ , "");
403 product48.setRentableStrategy(new Rentable(500));
404 Product product49 = createProductInProductGroup(productGroup9, "Bar med flere haner"
    ↪ , "");
405 product49.setRentableStrategy(new Rentable(500));
406 Product product50 = createProductInProductGroup(productGroup9, "Krus", "");
407 product50.setRentableStrategy(new Rentable(60));
408
409 addProductWithPriceToPricelist(pricelist2, product46, 250);
410 addProductWithPriceToPricelist(pricelist2, product47, 400);
411 addProductWithPriceToPricelist(pricelist2, product48, 500);
412 addProductWithPriceToPricelist(pricelist2, product49, 500);
413 addProductWithPriceToPricelist(pricelist2, product50, 60);
414
415 // Glas
416 ProductGroup productGroup10 = createProductGroup("Glas");
417 Product product51 = createProductInProductGroup(productGroup10, "Uanset størrelse",
    ↪ "");
418
419 addProductWithPriceToPricelist(pricelist2, product51, 15);
420
421 // Sampakning
422 ProductGroup productGroup11 = createProductGroup("Sampakning");
423 Bundle product52 = createBundleProductInProductGroup(productGroup11, "Gaveæske 2 øl,
    ↪ 2 glas", "");
424 for (int i = 0; i < 2; i++) {
425     product52.addProduct(product2);
426     product52.addProduct(product51);
427 }
428 Bundle product53 = createBundleProductInProductGroup(productGroup11, "Gaveæske 4 øl"
    ↪ , "");
429 for (int i = 0; i < 4; i++) {
430     product53.addProduct(product2);
431 }
432 Bundle product54 = createBundleProductInProductGroup(productGroup11, "Trækasse 6 øl"
    ↪ , "");
433 for (int i = 0; i < 6; i++) {
434     product54.addProduct(product2);
435 }
436 Bundle product55 = createBundleProductInProductGroup(productGroup11, "Gavekurv 6 øl,
    ↪ 2 glas", "");
437 for (int i = 0; i < 6; i++) {
438     product55.addProduct(product2);
439 }
440 product55.addProduct(product51);
441 product55.addProduct(product51);
442 Bundle product56 = createBundleProductInProductGroup(productGroup11, "Trækasse 6 øl,
    ↪ 6 glas", "");

```

```

443     for (int i = 0; i < 6; i++) {
444         product56.addProduct(product2);
445         product56.addProduct(product51);
446     }
447     Bundle product57 = createBundleProductInProductGroup(productGroup11, "Trækasse 12 øl
    ↪ ", "");
448     for (int i = 0; i < 12; i++) {
449         product57.addProduct(product2);
450     }
451     Bundle product58 = createBundleProductInProductGroup(productGroup11, "papkasse 12 øl
    ↪ ", "");
452     for (int i = 0; i < 12; i++) {
453         product58.addProduct(product2);
454     }
455
456     // Rundvisning
457     ProductGroup productGroup12 = createProductGroup("Rundvisning");
458     Product product59 = createTourProductInProductGroup(productGroup12, "Rundvisning pr
    ↪ person", "");
459     product59.setRentableStrategy(new Rentable());
460
461     pricelist3.addProductWithPrice(product59, 100);
462
463     // -----
464
465     // Snacks
466     ProductGroup pgSnacks = createProductGroup("Snacks");
467     Product chips = createProductInProductGroup(pgSnacks, "Chips", "");
468     Product peanuts = createProductInProductGroup(pgSnacks, "Peanuts", "");
469     Product chokolade = createProductInProductGroup(pgSnacks, "Chokolade", "");
470     Product kapsler = createProductInProductGroup(pgSnacks, "Kapsler", "");
471     Bundle bValentinskurv = createBundleProductInProductGroup(pgSnacks, "Karstens
    ↪ Valentinskurv", "");
472     bValentinskurv.addProduct(chips);
473     bValentinskurv.addProduct(peanuts);
474     Bundle bHjemmehygge = createBundleProductInProductGroup(pgSnacks, "Hjemme hygge", ""
    ↪ );
475     bHjemmehygge.addProduct(chokolade);
476     bHjemmehygge.addProduct(chips);
477
478     addProductWithPriceToPricelist(pricelist1, chips, 10);
479     addProductWithPriceToPricelist(pricelist1, peanuts, 10);
480     addProductWithPriceToPricelist(pricelist1, chokolade, 15);
481     addProductWithPriceToPricelist(pricelist1, kapsler, 10);
482     addProductWithPriceToPricelist(pricelist1, bValentinskurv, 20);
483     addProductWithPriceToPricelist(pricelist1, bHjemmehygge, 20);
484
485     Random random = new Random();
486     int pricelistCount = pricelist1.getProductsWithPrice().size();
487
488     for (int i = 0; i < 10000; i++) {
489         int rngProductIndex = random.nextInt(pricelistCount);
490         // Unoptimized
491         Product randomProduct = (Product) pricelist1.getProductsWithPrice().keySet().
    ↪ toArray()[rngProductIndex];
492         double productPrice = pricelist1.getPriceOfProduct(randomProduct);
493         Sale sale = createSale();
494         sale.updateSalesLine(randomProduct, 1, productPrice);

```

```
495         LocalDateTime localDateTime = LocalDateTime.of(2020, random.nextInt(12) + 1,
496             ↪ random.nextInt(29) + 1, 12, 0);
497         sale.setTimestamp(localDateTime);
498         PaymentMethod paymentMethod = PaymentMethod.values()[random.nextInt(
499             ↪ PaymentMethod.values().length)];
500         sale.addTransfer(paymentMethod, productPrice);
501         if (i % 5000 == 0)
502             sale.setSaleState(SaleState.DELAYED);
503         else
504             sale.setSaleState(SaleState.COMPLETED);
505     }
506     // Create dummy tour
507     Sale tourSale = createRent("Alexander", "Ikke tilgængelig", LocalDateTime.now(),
508         ↪ LocalDateTime.now());
509     tourSale.updateSalesLine(product59, 42, 100);
510 }
```

Chapter 2: gui/administrationpane/AdministrationPane.java

```
1 package gui.administrationpane;
2
3 import gui.LeftSideBar;
4 import gui.LeftSideBarInterface;
5 import javafx.geometry.Insets;
6 import javafx.geometry.Pos;
7 import javafx.scene.control.Label;
8 import javafx.scene.layout.BorderPane;
9 import javafx.scene.layout.Pane;
10
11 import java.util.LinkedHashMap;
12 import java.util.Map;
13
14 public class AdministrationPane extends BorderPane implements LeftSideBarInterface {
15
16     private static AdministrationPane administrationPane;
17
18     private LeftSideBar leftSideBar;
19
20     private AdministrationPane() {
21         this.setPadding(new Insets(20));
22
23         Map<String, Pane> navigationMap = new LinkedHashMap<>();
24         navigationMap.put("Produkt gruppe", NewProductGroupPane.getInstance());
25         navigationMap.put("Produkt", NewProductPane.getInstance());
26         navigationMap.put("Prisliester", PricelistPane.getInstance());
27
28         leftSideBar = new LeftSideBar(navigationMap, this);
29         setLeft(leftSideBar);
30
31         // Creating panes
32
33         changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
34     }
35
36     @Override
37     public void changeSelected(String nameOfButton) {
38         Label title = new Label(nameOfButton);
39         title.getStyleClass().add("title");
40         setAlignment(title, Pos.CENTER);
41         setTop(title);
42         setCenter(leftSideBar.getPane(nameOfButton));
43     }
44
45
46     public static AdministrationPane getInstance() {
47         if (administrationPane == null)
48             administrationPane = new AdministrationPane();
49     }
50 }
```

```
49 |         return administrationPane;  
50 |     }  
51 | }
```


Chapter 3: gui/administrationpane/NewProductGroupPane.java

```
1 package gui.administrationpane;
2
3 import controller.Controller;
4 import gui.helpers.TableViewHelper;
5 import gui.helpers.ValidationHelper;
6 import gui.overviewpane.ProductOverviewPane;
7 import javafx.scene.control.*;
8 import javafx.scene.layout.GridPane;
9 import javafx.scene.text.Font;
10 import model.ProductGroup;
11
12 public class NewProductGroupPane extends GridPane {
13
14     private static NewProductGroupPane newProductGroupPane;
15
16     private TextField txfName;
17
18     private NewProductGroupPane() {
19         this.getStyleClass().add("gridpane");
20
21         // Title
22         Label lblTitle = new Label("Ny produktgruppe");
23         lblTitle.setFont(new Font(24));
24         lblTitle.setPrefWidth(200);
25         this.add(lblTitle, 0, 0, 2, 1);
26
27         this.add(new Separator(), 0, 1, 2, 1);
28
29         // Fields
30         Label lblName = new Label("Navn:");
31         this.add(lblName, 0, 2);
32         txfName = new TextField();
33         txfName.setPromptText("Navn");
34         this.add(txfName, 1, 2);
35
36         Button btnCreateProduct = new Button("Opret Produktgruppe");
37         this.add(btnCreateProduct, 1, 3);
38         btnCreateProduct.setOnAction(event -> createProductGroup());
39
40         // Table
41         TableView<ProductGroup> productGroupTableView = TableViewHelper.
42             ↪ createProductGroupTable();
43         this.add(productGroupTableView, 2, 0, 1, 5);
44         productGroupTableView.setPrefHeight(700);
45         productGroupTableView.setPrefWidth(300);
46         productGroupTableView.setColumnResizePolicy(TableView.
47             ↪ CONSTRAINED_RESIZE_POLICY);
48         productGroupTableView.setEditable(false);
```

```
47     }
48
49     private void createProductGroup() {
50         boolean hasError = false;
51         String name = txfName.getText().trim();
52         if (!ValidationHelper.isStringBetween0to30characters(name)) {
53             txfName.getStyleClass().add("error");
54             hasError = true;
55         }
56         if (!hasError) {
57             ProductGroup productGroup = Controller.getInstance().
58                 ↪ createProductGroup(name);
59             TableViewHelper.addProductGroupToObservableList(productGroup);
60             ProductOverviewPane.getInstance().update();
61
62             txfName.clear();
63             ValidationHelper.removeErrorClassStyle(this);
64         }
65
66     }
67
68     public static void update() {
69         newProductGroupPane = new NewProductGroupPane();
70     }
71
72     public static NewProductGroupPane getInstance() {
73         if (newProductGroupPane == null)
74             newProductGroupPane = new NewProductGroupPane();
75         return newProductGroupPane;
76     }
77 }
```

Chapter 4: gui/administrationpane/NewProductPane.java

```
1 package gui.administrationpane;
2
3 import controller.Controller;
4 import gui.helpers.TableViewHelper;
5 import gui.helpers.ValidationHelper;
6 import gui.overviewpane.ProductOverviewPane;
7 import javafx.geometry.Pos;
8 import javafx.scene.control.*;
9 import javafx.scene.control.cell.PropertyValueFactory;
10 import javafx.scene.layout.GridPane;
11 import javafx.scene.layout.HBox;
12 import javafx.scene.text.Font;
13 import model.ProductGroup;
14 import model.enums.Unit;
15 import model.product.Bundle;
16 import model.product.Container;
17 import model.product.Product;
18 import model.product.Tour;
19 import model.product.rentable.Rentable;
20
21 import java.util.HashMap;
22 import java.util.Map;
23
24 public class NewProductPane extends GridPane {
25
26     private static NewProductPane productPane;
27
28     private TextField txfName;
29     private TextArea txfDescription;
30     private ChoiceBox<ProductGroup> productGroupChoiceBox;
31     private CheckBox rentableCheckBox;
32     private TextField txfSize;
33     private ChoiceBox<Unit> UnitChoiceBox;
34
35     String[] productMenuNames = { "Standard", "Standard (med størrelse)", "Sampakning",
36         ↪ "Rundvisning" };
37     private Map<String, ToggleButton> toggleButtonMap;
38
39     private TextField txfDeposit;
40
41     private TableView<Product> productTableView;
42
43     private TableView<Product> bundleProductsTableView;
44
45     private GridPane currentProductGridpane;
46
47     private NewProductPane() {
48         this.getStyleClass().add("gridpane");
```

```

48
49 // Title
50 Label lblTitle = new Label("Nyt produkt");
51 lblTitle.setFont(new Font(24));
52 lblTitle.setPrefWidth(300);
53 this.add(lblTitle, 0, 0, 2, 1);
54
55 this.add(new Separator(), 0, 1, 2, 1);
56
57 currentProductGridpane = new GridPane();
58 currentProductGridpane.getStyleClass().add(String.valueOf(this.getStyleClass
    ↪ ));
59 this.add(currentProductGridpane, 0, 3);
60
61 HBox hBox = new HBox();
62 hBox.setAlignment(Pos.CENTER);
63 hBox.setSpacing(5);
64 this.add(hBox, 0, 2, 2, 1);
65
66 toggleButtonMap = new HashMap<>();
67
68 ToggleGroup toggleGroup = new ToggleGroup();
69
70 for (String productMenuName : productMenuNames) {
71     ToggleButton toggleButton = new ToggleButton(productMenuName);
72     if (productMenuName.equals(productMenuNames[0])) { // Set "Standard"
    ↪ as selected
73         toggleButton.setSelected(true);
74         createStandardProductMenu();
75     }
76     toggleButtonMap.put(productMenuName, toggleButton);
77     hBox.getChildren().add(toggleButton);
78     toggleButton.setToggleGroup(toggleGroup);
79 }
80
81 toggleButtonMap.get(productMenuNames[0]).setOnAction(event ->
    ↪ createStandardProductMenu());
82 toggleButtonMap.get(productMenuNames[1]).setOnAction(event ->
    ↪ createContainerProductMenu());
83 toggleButtonMap.get(productMenuNames[2]).setOnAction(event ->
    ↪ createBundleProductMenu());
84 toggleButtonMap.get(productMenuNames[3]).setOnAction(event ->
    ↪ createTourProductMenu());
85
86 Label lblProductList = new Label("Produktliste:");
87 this.add(lblProductList, 2, 0);
88
89 // Table
90 bundleProductsTableView = null;
91 productTableView = TableViewHelper.createProductWithDescriptionTable();
92 this.add(productTableView, 2, 1, 1, 4);
93 productTableView.setPrefHeight(500);
94 productTableView.setPrefWidth(375);
95 productTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
96 productTableView.setEditable(false);
97 }
98
99 private void clearFields() {

```

```

100         txfName.clear();
101         txfDescription.clear();
102         txfDeposit.clear();
103         if (txfSize != null)
104             txfSize.clear();
105         if (bundleProductsTableView != null) {
106             bundleProductsTableView.getItems().clear();
107         }
108         ValidationHelper.removeErrorClassStyle(currentProductGridpane);
109     }
110
111     private void update(Product product) {
112         clearFields();
113         TableViewHelper.addProductToObservableList(product);
114         PricelistPane.getInstance().addNewProduct(product);
115         ProductOverviewPane.getInstance().update();
116     }
117
118     private String getProductName() {
119         return txfName.getText().trim();
120     }
121
122     private String getProductDescription() {
123         return txfDescription.getText().trim();
124     }
125
126     private ProductGroup getProductGroup() {
127         return productGroupChoiceBox.getSelectionModel().getSelectedItem();
128     }
129
130     private Double getDeposit() {
131         return Double.parseDouble(txfDeposit.getText());
132     }
133
134     private Unit getUnit() {
135         return UnitChoiceBox.getValue();
136     }
137
138     private Double getSize() {
139         return Double.parseDouble(txfSize.getText());
140     }
141
142     private boolean isRentable() {
143         return rentableCheckBox.isSelected();
144     }
145
146     private void createProduct() {
147         if (!hasError()) {
148             Product product = Controller.getInstance().
149                 ↪ createProductInProductGroup(getProductGroup(), getProductName
150                 ↪ (),
151                                     getProductDescription());
152
153             if (isRentable())
154                 product.setRentableStrategy(new Rentable(getDeposit()));
155
156             update(product); // update the pane(s)
157         }
158     }

```

```

156     }
157
158     private void createContainerProduct() {
159         if (!hasError()) {
160             Container container = Controller.getInstance().
161                 ↪ createContainerProductInProductGroup(getProductGroup(),
162                     getProductName(), getProductDescription(), getSize()
163                     ↪ , getUnit());
164
165             if (isRentable())
166                 container.setRentableStrategy(new Rentable(getDeposit()));
167
168             update(container);
169         }
170     }
171
172     private void createBundleProduct() {
173         if (!hasError()) {
174             Bundle bundle = Controller.getInstance().
175                 ↪ createBundleProductInProductGroup(getProductGroup(),
176                     getProductName(), getProductDescription());
177             for (Product product : bundleProductsTableView.getItems()) {
178                 bundle.addProduct(product);
179             }
180
181             if (isRentable())
182                 bundle.setRentableStrategy(new Rentable(getDeposit()));
183
184             update(bundle);
185         }
186     }
187
188     private void createTourProduct() {
189         if (!hasError()) {
190             Tour tour = Controller.getInstance().createTourProductInProductGroup
191                 ↪ (getProductGroup(), getProductName(),
192                     getProductDescription());
193             tour.setRentableStrategy(new Rentable(0d));
194
195             update(tour); // update the pane(s)
196         }
197     }
198
199     private boolean hasError() {
200         boolean hasError = false;
201         String name = txtName.getText().trim();
202         if (!ValidationHelper.isStringBetween0to30characters(name)) {
203             txtName.getStyleClass().add("error");
204             hasError = true;
205         }
206
207         ProductGroup productGroup = productGroupChoiceBox.getSelectionModel().
208             ↪ getSelectedItem();
209         if (productGroup == null) {
210             productGroupChoiceBox.getStyleClass().add("error");
211             hasError = true;
212         }

```

```

209     }
210
211     if (toggleButtonMap.get (productMenuNames[1]).isSelected()) {
212         if (!ValidationHelper.isNumber (txfSize.getText().trim()) || txfSize.
213             ↪ getText().isEmpty()) {
214             txfSize.getStyleClass().add("error");
215             hasError = true;
216         }
217     }
218
219     if (rentableCheckBox.isSelected())
220         if (!ValidationHelper.isNumberAndPositive (txfDeposit.getText().trim
221             ↪ ())) {
222             txfDeposit.getStyleClass().add("error");
223             hasError = true;
224         }
225     return hasError;
226 }
227
228 private void createStandardProductMenu() {
229     currentProductGridpane.getChildren().clear();
230
231     // Fields
232     Label lblName = new Label("Navn:");
233     lblName.setPrefWidth(125);
234     currentProductGridpane.add(lblName, 0, 0);
235     txfName = new TextField();
236     txfName.setPromptText("Navn");
237     currentProductGridpane.add(txfName, 1, 0);
238
239     Label lblDescription = new Label("Beskrivelse:");
240     currentProductGridpane.add(lblDescription, 0, 1);
241     txfDescription = new TextArea();
242     txfDescription.setPromptText("Beskrivelse");
243     txfDescription.setPrefRowCount(2);
244     txfDescription.setPrefHeight(60);
245     txfDescription.setPrefWidth(txfName.getWidth());
246     txfDescription.setWrapText(true);
247     currentProductGridpane.add(txfDescription, 1, 1);
248
249     Label lblProductGroup = new Label("Produktgruppe:");
250     currentProductGridpane.add(lblProductGroup, 0, 2);
251     productGroupChoiceBox = new ChoiceBox<>();
252     productGroupChoiceBox.setItems(TableViewController.getProductGroupObservableList
253         ↪ ());
254     currentProductGridpane.add(productGroupChoiceBox, 1, 2);
255
256     Label lblRentable = new Label("Kan udlejes:");
257     currentProductGridpane.add(lblRentable, 0, 3);
258
259     HBox hBox = new HBox();
260     hBox.setSpacing(5);
261     hBox.setAlignment(Pos.CENTER_LEFT);
262     currentProductGridpane.add(hBox, 1, 3);
263
264     rentableCheckBox = new CheckBox();
265     hBox.getChildren().add(rentableCheckBox);

```

```

264         txfDeposit = new TextField();
265         txfDeposit.setPromptText("Pant/depositum");
266         txfDeposit.setDisable(true);
267         hbox.getChildren().add(txfDeposit);
268
269         rentableCheckBox.selectedProperty().addListener((observable, oldValue,
270             ↪ newValue) -> {
271             if (newValue)
272                 txfDeposit.setDisable(false);
273             else {
274                 txfDeposit.setText("");
275                 txfDeposit.setDisable(true);
276             }
277         });
278
279         Button btnCreateProduct = new Button("Opret produkt");
280         currentProductGridpane.add(btnCreateProduct, 1, 4);
281         btnCreateProduct.setOnAction(event -> createProduct());
282     }
283
284     private void createContainerProductMenu() {
285         currentProductGridpane.getChildren().clear();
286
287         // Fields
288         Label lblName = new Label("Navn:");
289         lblName.setPrefWidth(125);
290         currentProductGridpane.add(lblName, 0, 0);
291         txfName = new TextField();
292         txfName.setPromptText("Navn");
293         currentProductGridpane.add(txfName, 1, 0);
294
295         Label lblDescription = new Label("Beskrivelse:");
296         currentProductGridpane.add(lblDescription, 0, 1);
297         txfDescription = new TextArea();
298         txfDescription.setPromptText("Beskrivelse");
299         txfDescription.setPrefRowCount(2);
300         txfDescription.setPrefHeight(60);
301         txfDescription.setPrefWidth(txfName.getWidth());
302         txfDescription.setWrapText(true);
303         currentProductGridpane.add(txfDescription, 1, 1);
304
305         Label lblProductGroup = new Label("Produktgruppe:");
306         currentProductGridpane.add(lblProductGroup, 0, 2);
307         productGroupChoiceBox = new ChoiceBox<>();
308         productGroupChoiceBox.setItems(TableViewController.getProductGroupObservableList
309             ↪ ());
310         currentProductGridpane.add(productGroupChoiceBox, 1, 2);
311
312         Label lblRentable = new Label("Kan udlejes:");
313         currentProductGridpane.add(lblRentable, 0, 3);
314
315         HBox hbox = new HBox();
316         hbox.setSpacing(5);
317         hbox.setAlignment(Pos.CENTER_LEFT);
318         currentProductGridpane.add(hbox, 1, 3);
319
320         rentableCheckBox = new CheckBox();
321         hbox.getChildren().add(rentableCheckBox);

```



```

320         txfDeposit = new TextField();
321         txfDeposit.setPromptText("Pant/depositum");
322         txfDeposit.setDisable(true);
323         hBox.getChildren().add(txfDeposit);
324
325         rentableCheckBox.selectedProperty().addListener((observable, oldValue,
326             ↪ newValue) -> {
327             if (newValue)
328                 txfDeposit.setDisable(false);
329             else {
330                 txfDeposit.setText("");
331                 txfDeposit.setDisable(true);
332             }
333         });
334
335         Label lblSize = new Label("Størrelse:");
336         currentProductGridpane.add(lblSize, 0, 4);
337
338         HBox hBoxSize = new HBox();
339         hBoxSize.setSpacing(5);
340         hBoxSize.setAlignment(Pos.TOP_LEFT);
341         currentProductGridpane.add(hBoxSize, 1, 4);
342
343         txfSize = new TextField();
344         txfSize.setMaxWidth(50);
345         hBoxSize.getChildren().add(txfSize);
346         UnitChoiceBox = new ChoiceBox<>();
347         for (Unit unit : Unit.values()) {
348             UnitChoiceBox.getItems().add(unit);
349         }
350         UnitChoiceBox.getSelectionModel().selectFirst();
351         hBoxSize.getChildren().add(UnitChoiceBox);
352
353         Button btnCreateProduct = new Button("Opret produkt");
354         currentProductGridpane.add(btnCreateProduct, 1, 5);
355         btnCreateProduct.setOnAction(event -> createContainerProduct());
356     }
357
358     private void createBundleProductMenu() {
359         currentProductGridpane.getChildren().clear();
360
361         // Fields
362         Label lblName = new Label("Bundt navn:");
363         lblName.setPrefWidth(125);
364         currentProductGridpane.add(lblName, 0, 0);
365         txfName = new TextField();
366         txfName.setPromptText("Navn");
367         currentProductGridpane.add(txfName, 1, 0);
368
369         Label lblProductGroup = new Label("Produktgruppe:");
370         currentProductGridpane.add(lblProductGroup, 0, 2);
371         productGroupChoiceBox = new ChoiceBox<>();
372         productGroupChoiceBox.setItems(TableViewController.getProductGroupObservableList
373             ↪ ());
374         currentProductGridpane.add(productGroupChoiceBox, 1, 2);
375
376         /////

```

```
376 Label lblRentable = new Label("Kan udlejes:");
377 currentProductGridpane.add(lblRentable, 0, 3);
378
379 HBox hBox = new HBox();
380 hBox.setSpacing(5);
381 hBox.setAlignment(Pos.CENTER_LEFT);
382 currentProductGridpane.add(hBox, 1, 3);
383 rentableCheckBox = new CheckBox();
384 hBox.getChildren().add(rentableCheckBox);
385
386 txfDeposit = new TextField();
387 txfDeposit.setPromptText("Pant/depositum");
388 txfDeposit.setDisable(true);
389 hBox.getChildren().add(txfDeposit);
390
391 rentableCheckBox.selectedProperty().addListener((observable, oldValue,
392     ↪ newValue) -> {
393     if (newValue)
394         txfDeposit.setDisable(false);
395     else {
396         txfDeposit.setText("");
397         txfDeposit.setDisable(true);
398     }
399 });
400
401 ///
402
403 Label lblProductsInBundle = new Label("Produkter i sampakning:");
404 currentProductGridpane.add(lblProductsInBundle, 0, 4);
405
406 HBox hbProductsInBundle = new HBox();
407 hbProductsInBundle.setSpacing(5);
408 hbProductsInBundle.setAlignment(Pos.CENTER);
409 currentProductGridpane.add(hbProductsInBundle, 0, 5, 2, 2);
410
411 bundleProductsTableView = new TableView<>();
412 bundleProductsTableView.setPrefHeight(300);
413 bundleProductsTableView.setPrefWidth(250);
414 bundleProductsTableView.setColumnResizePolicy(TableView.
415     ↪ CONSTRAINED_RESIZE_POLICY);
416 bundleProductsTableView.setEditable(false);
417 hbProductsInBundle.getChildren().add(bundleProductsTableView);
418
419 Button btnAddToBundle = new Button("Tilføj");
420 btnAddToBundle.setOnAction(event -> addToBundle());
421 hbProductsInBundle.getChildren().add(btnAddToBundle);
422
423 TableColumn<Product, String> column1 = new TableColumn<>("Navn");
424 column1.setCellValueFactory(new PropertyValueFactory<>("name"));
425
426 TableColumn<Product, String> column2 = new TableColumn<>("Produktgruppe");
427 column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));
428
429 bundleProductsTableView.getColumns().add(column1);
430 bundleProductsTableView.getColumns().add(column2);
431
432 Button btnCreateBundle = new Button("Opret sampakning");
```

```

432         btnCreateBundle.setOnAction(event -> createBundleProduct());
433         currentProductGridpane.add(btnCreateBundle, 1, 8);
434     }
435
436     private void addToBundle() {
437         Product product = productTableView.getSelectionModel().getSelectedItem();
438         if (product != null && !(product instanceof Bundle)) {
439             bundleProductsTableView.getItems().add(product);
440         }
441     }
442
443     private void createTourProductMenu() {
444         currentProductGridpane.getChildren().clear();
445
446         // Fields
447         Label lblName = new Label("Navn:");
448         lblName.setPrefWidth(125);
449         currentProductGridpane.add(lblName, 0, 0);
450         TextField txfName = new TextField();
451         txfName.setPromptText("Navn");
452         currentProductGridpane.add(txfName, 1, 0);
453
454         Label lblDescription = new Label("Beskrivelse:");
455         currentProductGridpane.add(lblDescription, 0, 1);
456         TextArea txfDescription = new TextArea();
457         txfDescription.setPromptText("Beskrivelse");
458         txfDescription.setPrefRowCount(2);
459         txfDescription.setPrefHeight(60);
460         txfDescription.setPrefWidth(txfName.getWidth());
461         txfDescription.setWrapText(true);
462         currentProductGridpane.add(txfDescription, 1, 1);
463
464         Label lblProductGroup = new Label("Produktgruppe:");
465         currentProductGridpane.add(lblProductGroup, 0, 2);
466         productGroupChoiceBox = new ChoiceBox<>();
467         productGroupChoiceBox.setItems(TableViewControllerHelper.getProductGroupObservableList
            ↪ ());
468         currentProductGridpane.add(productGroupChoiceBox, 1, 2);
469
470         Button btnCreateProduct = new Button("Opret rundvisning");
471         currentProductGridpane.add(btnCreateProduct, 1, 4);
472         btnCreateProduct.setOnAction(event -> createTourProduct());
473     }
474
475     public static NewProductPane getInstance() {
476         if (productPane == null)
477             productPane = new NewProductPane();
478         return productPane;
479     }
480 }

```

Chapter 5: gui/administrationpane/PricelistPane.java

```
1 package gui.administrationpane;
2
3 import controller.Controller;
4 import gui.helpers.ProductAndPriceHelper;
5 import gui.helpers.TableViewHelper;
6 import gui.helpers.ValidationHelper;
7 import gui.overviewpane.PricelistOverviewPane;
8 import gui.salepane.SelectPricelistPane;
9 import javafx.collections.FXCollections;
10 import javafx.collections.ObservableList;
11 import javafx.geometry.Pos;
12 import javafx.scene.control.*;
13 import javafx.scene.control.cell.PropertyValueFactory;
14 import javafx.scene.control.cell.TextFieldTableCell;
15 import javafx.scene.layout.GridPane;
16 import javafx.scene.layout.VBox;
17 import javafx.scene.text.Font;
18 import javafx.util.converter.DoubleStringConverter;
19 import model.Pricelist;
20 import model.product.Product;
21
22 import java.util.LinkedList;
23 import java.util.List;
24
25 public class PricelistPane extends GridPane {
26
27     private static PricelistPane pricelistPane;
28
29     private TextField txfName;
30
31     private Pricelist selectedPricelist;
32
33     private TableView<ProductAndPriceHelper> productInPricelistTableView;
34     private static ObservableList<ProductAndPriceHelper> leftList = FXCollections.
35         ↪ observableList(new LinkedList<>());
36     private TableView<ProductAndPriceHelper> productNotInPricelistTableView;
37     private static ObservableList<ProductAndPriceHelper> rightList = FXCollections.
38         ↪ observableList(new LinkedList<>());
39
40     private PricelistPane() {
41         this.getStyleClass().add("gridpane");
42
43         // Title
44         Label lblTitle = new Label("Ny prisliste");
45         lblTitle.setFont(new Font(24));
46         lblTitle.setPrefWidth(200);
47         this.add(lblTitle, 0, 0, 2, 1);
```

```

47     this.add(new Separator(), 0, 1, 2, 1);
48
49     Label lblName = new Label("Navn:");
50     this.add(lblName, 0, 2);
51     TextField txfName = new TextField();
52     txfName.setPromptText("Navn");
53     this.add(txfName, 1, 2);
54
55     Button btnCreatePricelist = new Button("Opret prisliste");
56     this.add(btnCreatePricelist, 1, 3);
57     btnCreatePricelist.setOnAction(event -> createPricelist());
58
59     Label lblChoosePricelist = new Label("Vælg prisliste:");
60     this.add(lblChoosePricelist, 0, 4);
61
62     TableView<Pricelist> pricelistTableView = TableViewHelper.createPricelistTable();
63     this.add(pricelistTableView, 0, 5, 2, 1);
64     pricelistTableView.setPrefHeight(500);
65     pricelistTableView.setPrefWidth(200);
66     pricelistTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
67     pricelistTableView.setEditable(false);
68     pricelistTableView.getSelectionModel().selectedItemProperty().addListener((obs,
        ↪ oldSelection, newSelection) -> {
69         if (newSelection != null) {
70             selectedPricelist = newSelection;
71             updateLists();
72         }
73     });
74
75     Label lblLeftTable = new Label("Prislistens indhold: (ret i pris, enter for at gemme
        ↪ )");
76     this.add(lblLeftTable, 2, 0, 2, 1);
77
78     // Left table
79     productInPricelistTableView = new TableView<>();
80     this.add(productInPricelistTableView, 2, 1, 2, 5);
81     productInPricelistTableView.setPrefHeight(700);
82     productInPricelistTableView.setPrefWidth(400);
83     productInPricelistTableView.setColumnResizePolicy(TableView.
        ↪ CONSTRAINED_RESIZE_POLICY);
84     createProductInPricelistTableView();
85
86     VBox vBox = new VBox();
87     vBox.setAlignment(Pos.CENTER);
88     vBox.setSpacing(20);
89     this.add(vBox, 4, 0, 1, 6);
90
91     Button btnLeft = new Button("<-");
92     vBox.getChildren().add(btnLeft);
93     btnLeft.setOnAction(event -> moveLeft());
94
95     Button btnRight = new Button("->");
96     vBox.getChildren().add(btnRight);
97     btnRight.setOnAction(event -> moveRight());
98
99     Label lblRightTable = new Label("Produkter til tilføjelse:");
100    this.add(lblRightTable, 5, 0, 2, 1);
101

```

```

102 // Right table
103 productNotInPricelistTableView = new TableView<>();
104 this.add(productNotInPricelistTableView, 5, 1, 2, 5);
105 productNotInPricelistTableView.setPrefHeight(700);
106 productNotInPricelistTableView.setPrefWidth(400);
107 productNotInPricelistTableView.setColumnResizePolicy(TableView.
    ↳ CONSTRAINED_RESIZE_POLICY);
108 createProductNotInPricelistTableView();
109 }
110
111 private void createPricelist() {
112     String name = txfName.getText().trim();
113     if (!ValidationHelper.isStringBetween0to30characters(name)) {
114         txfName.getStyleClass().add("error");
115         return;
116     }
117     Pricelist pricelist = Controller.getInstance().createPricelist(name);
118     TableViewHelper.addPricelistToObservableList(pricelist);
119     SelectPricelistPane.getInstance().addPricelist(pricelist);
120     txfName.clear();
121     ValidationHelper.removeErrorClassStyle(this);
122 }
123
124 private void createProductInPricelistTableView() {
125     productInPricelistTableView.setEditable(true);
126
127     TableColumn<ProductAndPriceHelper, String> column1 = new TableColumn<>("Navn");
128     column1.setCellValueFactory(new PropertyValueFactory<>("product"));
129
130     TableColumn<ProductAndPriceHelper, String> column2 = new TableColumn<>("
    ↳ Produktgruppe");
131     column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));
132
133     TableColumn<ProductAndPriceHelper, Double> column3 = new TableColumn<>("Pris");
134     column3.setCellValueFactory(new PropertyValueFactory<>("price"));
135     column3.setCellFactory(TextFieldTableCell.forTableColumn(new DoubleStringConverter()
    ↳ ));
136     column3.setOnEditCommit(event -> {
137         ProductAndPriceHelper productAndPriceHelper = event.getRowValue();
138         ProductAndPriceHelper.setPrice(event.getNewValue());
139         selectedPricelist.updateProductPrice(ProductAndPriceHelper.getProduct(),
    ↳ ProductAndPriceHelper.getPrice());
140         PricelistOverviewPane.getInstance().update();
141     });
142     column3.setEditable(true);
143     column3.setMinWidth(60);
144     column3.setMaxWidth(60);
145
146     productInPricelistTableView.getColumns().add(column1);
147     productInPricelistTableView.getColumns().add(column2);
148     productInPricelistTableView.getColumns().add(column3);
149
150     productInPricelistTableView.setItems(leftList);
151 }
152
153 private void createProductNotInPricelistTableView() {
154     productNotInPricelistTableView.setEditable(false);
155 }

```

```

156     TableColumn<ProductAndPriceHelper, String> column1 = new TableColumn<>("Navn");
157     column1.setCellValueFactory(new PropertyValueFactory<>("product"));
158
159     TableColumn<ProductAndPriceHelper, String> column2 = new TableColumn<>("
160         ↪ Produktgruppe");
161     column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));
162
163     productNotInPricelistTableView.getColumns().add(column1);
164     productNotInPricelistTableView.getColumns().add(column2);
165
166     productNotInPricelistTableView.setItems(rightList);
167 }
168
169 public void updateLists() {
170     if (selectedPricelist == null)
171         return;
172
173     leftList.clear();
174     leftList.setAll(ProductAndPriceHelper.parseToProductAndPrice(selectedPricelist.
175         ↪ getProductsWithPrice()));
176
177     List<Product> productList = Controller.getInstance().getProductList();
178     productList.removeAll(ProductAndPriceHelper.getProductList(leftList));
179     rightList.setAll(ProductAndPriceHelper.parseToProductAndPrice(productList));
180 }
181
182 public void addNewProduct(Product product) {
183     rightList.add(new ProductAndPriceHelper(product, 0));
184 }
185
186 private void moveLeft() {
187     ProductAndPriceHelper productAndPriceHelper = productNotInPricelistTableView.
188         ↪ getSelectionModel()
189         .getSelectedItem();
190     if (productNotInPricelistTableView.getSelectionModel().getSelectedItem() != null) {
191         rightList.remove(productAndPriceHelper);
192         leftList.add(productAndPriceHelper);
193     }
194     selectedPricelist.addProductWithPrice(ProductAndPriceHelper.getProduct(),
195         ↪ ProductAndPriceHelper.getPrice());
196     PricelistOverviewPane.getInstance().update();
197 }
198
199 private void moveRight() {
200     ProductAndPriceHelper productAndPriceHelper = productInPricelistTableView.
201         ↪ getSelectionModel().getSelectedItem();
202     if (productInPricelistTableView.getSelectionModel().getSelectedItem() != null) {
203         leftList.remove(productAndPriceHelper);
204         rightList.add(productAndPriceHelper);
205     }
206     selectedPricelist.removeProductWithPrice(ProductAndPriceHelper.getProduct());
207 }
208
209 public static PricelistPane getInstance() {
210     if (pricelistPane == null)
211         pricelistPane = new PricelistPane();
212     return pricelistPane;
213 }

```

209 | }

Chapter 6: gui/css/app.css

```
1  /* ===== General Style */
2  .label.title {
3      -fx-font-size: 32px;
4  }
5
6  .label.subtitle {
7      -fx-font-size: 15px;
8  }
9
10 /* ... Error Handling*/
11 .text-field.error, .date-picker.error, .choice-box.error, .text-area.error {
12     -fx-background-color: #EF6B71;
13 }
14
15 .gridpane {
16     -fx-padding: 20;
17     -fx-hgap: 20;
18     -fx-vgap: 10;
19 }
20
21 /* ===== Specific elements */
22 .button.important {
23     -fx-background-color: green;
24     -fx-min-width: 150;
25     -fx-min-height: 150;
26 }
27 .button.important:hover {
28     -fx-background-color: lightgreen;
29 }
30
31 .label.clock {
32     -fx-font-size: 14px;
33     -fx-font-weight: bold;
34     -fx-font-family: Menlo;
35 }
36
37 .dateinterval-picker {
38     -fx-alignment: center;
39 }
40 /* Mono font for better readability */
41 .salesLine {
42     -fx-font-family: "Courier New";
43 }
44
45 /* ... Tour */
46 .tour .month-year-pane {
47     -fx-background-color: aqua;
48     -fx-padding: 5 5 5 5;
```

```
49 | }
50 |
51 | .tour .day-name-cell {
52 |     -fx-background-color: black;
53 |     -fx-text-fill: aqua;
54 | }
55 |
56 | .tour-cell.weekend {
57 |     -fx-background-color: #70a666;
58 | }
59 |
60 | .tour-cell.weekday {
61 |     -fx-background-color: #ffda55;
62 | }
63 |
64 | .tour-cell.occupied {
65 |     -fx-background-color: #913a3a;
66 | }
```

Chapter 7: gui/helpers/Calendar.java

```
1 package gui.helpers;
2
3 import com.sun.javafx.scene.control.skin.DatePickerSkin;
4 import javafx.scene.Node;
5 import javafx.scene.layout.HBox;
6 import model.Sale;
7
8 import java.util.List;
9
10 public class Calendar extends HBox {
11
12     public Calendar(List<Sale> sales) {
13         ProductDatePicker productDatePicker = new ProductDatePicker(sales);
14
15         DatePickerSkin datePickerSkin = new DatePickerSkin(productDatePicker);
16         Node content = datePickerSkin.getPopupContent();
17
18         this.getChildren().add(content);
19     }
20 }
```

Chapter 8: gui/helpers/DateIntervalPicker.java

```
1 package gui.helpers;
2
3 import javafx.scene.control.DatePicker;
4 import javafx.scene.control.Label;
5 import javafx.scene.layout.HBox;
6
7 import java.time.LocalDate;
8 import java.util.Arrays;
9
10 public class DateIntervalPicker extends HBox {
11     private DatePicker datePickerFrom;
12     private DatePicker datePickerTo;
13
14     public DateIntervalPicker() {
15         this.getStyleClass().add("dateinterval-picker");
16         Label lblFrom = new Label("Fra: ");
17         Label lblTo = new Label("Til: ");
18         datePickerFrom = new DatePicker();
19         datePickerFrom.setValue(LocalDate.now().minusDays(30));
20         datePickerTo = new DatePicker();
21         datePickerTo.setValue(LocalDate.now());
22
23         datePickerFrom.valueProperty().addListener(observable ->{
24             if (datePickerFrom.getValue().isAfter(datePickerTo.getValue()))
25                 datePickerFrom.setValue(datePickerTo.getValue().minusDays(30));
26         });
27
28         datePickerTo.valueProperty().addListener(observable -> {
29             if (datePickerTo.getValue().isBefore(datePickerFrom.getValue()))
30                 datePickerTo.setValue(datePickerFrom.getValue().plusDays(30));
31         });
32
33         this.getChildren().addAll(Arrays.asList(lblFrom, datePickerFrom, lblTo, datePickerTo
34             ↪ ));
35     }
36
37     public LocalDate getDateFrom() {
38         return datePickerFrom.getValue();
39     }
40
41     public LocalDate getDateTo() {
42         return datePickerTo.getValue();
43     }
44 }
```

Chapter 9: gui/helpers/ProductAndPriceHelper.java

```
1 package gui.helpers;
2
3 import model.ProductGroup;
4 import model.product.Product;
5
6 import java.util.LinkedList;
7 import java.util.List;
8 import java.util.Map;
9
10
11 /**
12  * Helper class for GUI - Makes TableViews possible and easier
13  */
14 public class ProductAndPriceHelper {
15     private Product product;
16     private double price;
17
18     public ProductAndPriceHelper(Product product, double price) {
19         this.product = product;
20         this.price = price;
21     }
22
23     public String getName() {
24         return product.getName();
25     }
26
27     public Product getProduct() {
28         return product;
29     }
30
31     public ProductGroup getProductGroup() {
32         return product.getProductGroup();
33     }
34
35     public double getPrice() {
36         return price;
37     }
38
39     public void setPrice(double price) {
40         this.price = price;
41     }
42
43     public String getDeposit() {
44         double deposit = product.getDeposit();
45         return deposit > -1 ? String.valueOf(deposit) : ""; // Else return empty string
46     }
47 }
```

```
48 public static List<ProductAndPriceHelper> parseToProductAndPrice (Map<Product, Double>
    ↪ pricelist) {
49     List<ProductAndPriceHelper> productAndPriceList = new LinkedList<>();
50     for (Product product : pricelist.keySet()) {
51         productAndPriceList.add(new ProductAndPriceHelper(product, pricelist.get(product
    ↪ )));
52     }
53     return productAndPriceList;
54 }
55
56 public static List<ProductAndPriceHelper> parseToProductAndPrice (List<Product>
    ↪ productList) {
57     List<ProductAndPriceHelper> productAndPriceList = new LinkedList<>();
58     for (Product product : productList) {
59         productAndPriceList.add(new ProductAndPriceHelper(product, 0));
60     }
61     return productAndPriceList;
62 }
63
64 public static List<Product> getProductList (List<ProductAndPriceHelper> productAndPrices)
    ↪ {
65     List<Product> products = new LinkedList<>();
66     for (ProductAndPriceHelper productAndPrice : productAndPrices) {
67         products.add(productAndPrice.getProduct());
68     }
69     return products;
70 }
71
72 public static void removeProductGroupFromList (List<ProductAndPriceHelper> list,
    ↪ ProductGroup productGroup) {
73     list.removeIf(productAndPriceHelper -> productAndPriceHelper.getProductGroup().
    ↪ equals(productGroup));
74 }
75 }
```

Chapter 10: gui/helpers/ProductDatePicker.java

```
1 package gui.helpers;
2
3 import javafx.scene.control.DateCell;
4 import javafx.scene.control.DatePicker;
5 import javafx.scene.control.Tooltip;
6 import javafx.util.Callback;
7 import javafx.util.StringConverter;
8 import model.Rent;
9 import model.Sale;
10
11 import java.time.DayOfWeek;
12 import java.time.LocalDate;
13 import java.time.format.DateTimeFormatter;
14 import java.time.temporal.ChronoField;
15 import java.util.HashMap;
16 import java.util.LinkedList;
17 import java.util.List;
18 import java.util.Map;
19 import java.util.stream.Collectors;
20
21 public class ProductDatePicker extends DatePicker {
22     Map<LocalDate, List<Rent>> dateSalesMap;
23
24     public ProductDatePicker(List<Sale> sales) {
25         this.showWeekNumbersProperty().setValue(true);
26         dateSalesMap = new HashMap<>();
27
28         for (Sale sale : sales) {
29             LocalDate tourDate = ((Rent) sale).getDeliveryDateAndTime().toLocalDate();
30             if (!dateSalesMap.containsKey(tourDate)) {
31                 dateSalesMap.put(tourDate, new LinkedList<>());
32             }
33             dateSalesMap.get(tourDate).add((Rent) sale);
34         }
35         this.setDayCellFactory(new Callback<DatePicker, DateCell>() {
36             @Override
37             public DateCell call(DatePicker param) {
38                 return new DateCell() {
39                     @Override
40                     public void updateItem(LocalDate item, boolean empty) {
41                         super.updateItem(item, empty);
42                         getStyleClass().add("tour-cell");
43                         if (empty || item == null) {
44                             setText(null);
45                             setGraphic(null);
46                         } else {
47                             // If date is before today: disable
48                             if (item.isBefore(LocalDate.now()))
```

```

49         setDisable(true);
50
51         DayOfWeek day = DayOfWeek.of(item.get(ChronoField.DAY_OF_WEEK));
52         if (dateSalesMap.containsKey(item)) { // if day has Tour
53             getStyleClass().add("occupied");
54             setText(getText());
55             setTooltip(new Tooltip(dateSalesMap.get(item).stream()
56                 .map(rent -> "-" + rent.getContactName() + " (" +
57                     ↪ rent.getContactInformation() + ")")
58                 .collect(Collectors.joining("\n"))));
59         } else if (day.equals(DayOfWeek.SATURDAY) || day.equals(
60             ↪ DayOfWeek.SUNDAY)) { // if weekend
61             getStyleClass().add("weekend");
62         } else { // else must be weekday
63             getStyleClass().add("weekday");
64         }
65     }
66 }
67 });
68
69 /*
70  * This is purely to forcefully format date to European standard.
71  */
72 this.setConverter(new StringConverter<LocalDate>() {
73     private DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("dd/MM
74         ↪ /yyyy");
75
76     @Override
77     public String toString(LocalDate localDate) {
78         return (localDate == null) ? "" : dateTimeFormatter.format(localDate);
79     }
80
81     @Override
82     public LocalDate fromString(String string) {
83         return (string == null || string.trim().isEmpty()) ? null : LocalDate.parse(
84             ↪ string, dateTimeFormatter);
85     }
86 });
87 }
88 }

```


Chapter 11: gui/helpers/TableViewHelper.java

```
1 package gui.helpers;
2
3 import controller.Controller;
4 import javafx.collections.FXCollections;
5 import javafx.collections.ObservableList;
6 import javafx.scene.control.TableColumn;
7 import javafx.scene.control.TableView;
8 import javafx.scene.control.cell.PropertyValueFactory;
9 import model.Pricelist;
10 import model.ProductGroup;
11 import model.product.Product;
12
13 public class TableViewHelper {
14
15     private static Controller controller = Controller.getInstance();
16
17     private static ObservableList<ProductGroup> productGroupObservableList = FXCollections.
18         ↪ observableList(controller.getProductGroupList());
19     private static ObservableList<Product> productObservableList = FXCollections.
20         ↪ observableList(controller.getProductList());
21     private static ObservableList<Pricelist> pricelistObservableList = FXCollections.
22         ↪ observableList(controller.getPricelists());
23
24     //
25     ↪ -----
26     ↪
27
28     public static TableView<ProductGroup> createProductGroupTable() {
29         TableView<ProductGroup> productGroupTableView = new TableView<>();
30         TableColumn<ProductGroup, String> column1 = new TableColumn<>("Navn");
31         column1.setCellValueFactory(new PropertyValueFactory<>("name"));
32         productGroupTableView.getColumns().add(column1);
33         productGroupTableView.setItems(productGroupObservableList);
34         return productGroupTableView;
35     }
36
37     public static TableView<Product> createProductWithDescriptionTable() {
38         TableView<Product> productTableView = new TableView<>();
39         TableColumn<Product, String> column1 = new TableColumn<>("Navn");
40         column1.setCellValueFactory(new PropertyValueFactory<>("name"));
41         TableColumn<Product, String> column2 = new TableColumn<>("Produktgruppe");
42         column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));
43         TableColumn<Product, String> column3 = new TableColumn<>("Beskrivelse");
44         column3.setCellValueFactory(new PropertyValueFactory<>("description"));
45         productTableView.getColumns().add(column1);
46         productTableView.getColumns().add(column2);
47         productTableView.getColumns().add(column3);
48         productTableView.setItems(productObservableList);
49     }
50 }
```

```
44     return productTableView;
45 }
46
47 public static TableView<Pricelist> createPricelistTable() {
48     TableView<Pricelist> pricelistTableView = new TableView<>();
49     TableColumn<Pricelist, String> column1 = new TableColumn<>("Navn");
50     column1.setCellValueFactory(new PropertyValueFactory<>("name"));
51     pricelistTableView.getColumns().add(column1);
52     pricelistTableView.setItems(pricelistObservableList);
53     return pricelistTableView;
54 }
55
56 //
57
58 public static void addProductGroupToObservableList(ProductGroup productGroup) {
59     productGroupObservableList.add(productGroup);
60 }
61
62 public static void addProductToObservableList(Product product) {
63     productObservableList.add(product);
64 }
65
66 public static void addPricelistToObservableList(Pricelist pricelist) {
67     pricelistObservableList.add(pricelist);
68 }
69
70 //
71
72 public static ObservableList<ProductGroup> getProductGroupObservableList() {
73     return productGroupObservableList;
74 }
75
76 }
```

Chapter 12: gui/helpers/TimePicker.java

```
1 package gui.helpers;
2
3 import javafx.scene.control.Spinner;
4 import javafx.scene.control.SpinnerValueFactory;
5 import javafx.scene.layout.HBox;
6
7 import java.time.LocalTime;
8 import java.util.Arrays;
9
10 public class TimePicker extends HBox {
11     private Spinner<Integer> sHours;
12     private Spinner<Integer> sMinutes;
13
14     public TimePicker() {
15         sHours = new Spinner<Integer>();
16         sHours.setValueFactory(new SpinnerValueFactory.IntegerSpinnerValueFactory(0,24, 12))
17             ↪ ;
18
19         sMinutes = new Spinner<Integer>();
20         sMinutes.setValueFactory(new SpinnerValueFactory.IntegerSpinnerValueFactory(0,60,0)
21             ↪ {
22             @Override
23             public void decrement(int steps) {
24                 if(getValue() == 0) {
25                     sHours.decrement();
26                     setValue(45);
27                 } else {
28                     setValue(getValue()-15);
29                 }
30             }
31             @Override
32             public void increment(int steps) {
33                 if (getValue() == 60) {
34                     sHours.increment();
35                     setValue(0);
36                 } else {
37                     setValue(getValue()+15);
38                 }
39             }
40         });
41         this.getChildren().addAll(Arrays.asList(sHours, sMinutes));
42     }
43
44     public LocalTime getValue() {
45         return LocalTime.of(sHours.getValue(), sMinutes.getValue());
46     }
47 }
```

Chapter 13: gui/helpers/ValidationHelper.java

```
1 package gui.helpers;
2
3 import javafx.scene.layout.Pane;
4
5 /**
6  * For GUI validation
7  */
8 public class ValidationHelper {
9
10     /**
11      * Removes CSS class attributes "error" from all children of pane
12      *
13      * @param pane to loop through
14      */
15     public static void removeErrorClassStyle(Pane pane) {
16         pane.getChildren().stream().forEach(node -> {
17             // Rekursivt kald
18             if (node instanceof Pane)
19                 removeErrorClassStyle((Pane) node);
20
21             node.getStyleClass().remove("error");
22         });
23     }
24
25     /**
26      * Checks whether a given string length is: - less than or equal to 30 - bigger
27      * than 0 True if valid else false.
28      *
29      * @param string to be checked
30      * @return boolean
31      */
32     public static boolean isStringBetween0to30characters(String string) {
33         return string.matches(".+") || string.length() >= 30;
34     }
35
36     /**
37      * Tests if string is a number and doesn't overflow double/integer max
38      * @param string to be checked if number
39      * @return true if number
40      */
41     public static boolean isNumber(String string) {
42         return string.matches("\\d+") && string.length() < 9;
43     }
44
45     public static boolean isNumberAndPositive(String string) {
46         return isNumber(string) && Double.parseDouble(string) >= 0;
47     }
48 }
```

```
49  /**
50   *
51   * @param string to be checked if percentage number
52   * @return true if string represent a percentage number
53   */
54  public static boolean isPercentNumber(String string) {
55      String subsubstring = string.substring(0, string.length() - 1);
56      return string.lastIndexOf('%') == string.length() - 1 && isNumber(subsubstring)
57          && Double.parseDouble(subsubstring) <= 100;
58  }
59 }
```

Chapter 14: gui/HomePane.java

```
1 package gui;
2
3 import gui.administrationpane.AdministrationPane;
4 import gui.overviewpane.OverviewPane;
5 import gui.images.LoadImage;
6 import gui.salepane.SaleOptionsPane;
7 import javafx.geometry.Pos;
8 import javafx.scene.control.Button;
9 import javafx.scene.control.Label;
10 import javafx.scene.control.Tooltip;
11 import javafx.scene.image.Image;
12 import javafx.scene.image.ImageView;
13 import javafx.scene.layout.BorderPane;
14 import javafx.scene.layout.HBox;
15 import javafx.scene.layout.VBox;
16
17 import java.util.HashMap;
18 import java.util.Map;
19
20 public class HomePane extends BorderPane {
21
22     private static HomePane homePane = getInstance(); // eager loaded
23
24     private static final String IMAGE_FOLDER = "res/images/homepane";
25
26     public HomePane() {
27         Map<String, Image> imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);
28
29         HBox hBoxButtons = new HBox();
30         this.setCenter(hBoxButtons);
31         hBoxButtons.setAlignment(Pos.CENTER);
32         hBoxButtons.setSpacing(100);
33
34         String[] btnNames = {"Oversigt", "Administration", "Salg", "Statistikker"};
35         Map<String, Button> buttons = new HashMap<>();
36
37         for (String btnName : btnNames) {
38             Button button = new Button(btnName);
39             buttons.put(btnName, button);
40             button.setPrefHeight(100);
41             button.setPrefWidth(100);
42             button.setTooltip(new Tooltip(btnName));
43
44             VBox vBox = new VBox();
45             vBox.setSpacing(5);
46             vBox.setAlignment(Pos.CENTER);
47             vBox.getChildren().add(button);
48             vBox.getChildren().add(new Label(btnName));
```

```
49         hBoxButtons.getChildren().add(vBox);
50     if (imageMap != null && imageMap.containsKey(btnName)) {
51         ImageView imageView = new ImageView(imageMap.get(btnName));
52         button.setGraphic(imageView);
53         button.setText(""); // Remove text and use button instead
54     }
55 }
56
57 buttons.get(btnNames[0]).setOnAction(event -> MainApp.changePane(OverviewPane.
58     ↪ getInstance(), true));
59 buttons.get(btnNames[1]).setOnAction(event -> MainApp.changePane(AdministrationPane.
60     ↪ getInstance(), true));
61 buttons.get(btnNames[2]).setOnAction(event -> MainApp.changePane(SaleOptionsPane.
62     ↪ getInstance(), true));
63 buttons.get(btnNames[3]).setOnAction(event -> MainApp.changePane(StatisticPane.
64     ↪ getInstance(), true));
65 }
66
67 public static HomePane getInstance() {
68     if (homePane == null)
69         homePane = new HomePane();
70     return homePane;
71 }
```

Chapter 15: gui/LeftSideBarInterface.java

```
1 package gui;
2
3 public interface LeftSideBarInterface {
4     void changeSelected(String nameOfPane);
5 }
```


Chapter 16: gui/LeftSideBar.java

```
1 package gui;
2
3 import javafx.geometry.Pos;
4 import javafx.scene.control.ToggleButton;
5 import javafx.scene.control.ToggleGroup;
6 import javafx.scene.control.Tooltip;
7 import javafx.scene.layout.Pane;
8 import javafx.scene.layout.VBox;
9 import javafx.scene.text.TextAlignment;
10
11 import java.util.HashMap;
12 import java.util.Map;
13
14 public class LeftSideBar extends VBox {
15     private Map<String, Pane> nameAndPaneMap;
16
17     public LeftSideBar(Map<String, Pane> nameAndPaneMap, LeftSideBarInterface pane) {
18         this.nameAndPaneMap = nameAndPaneMap;
19         this.setAlignment(Pos.CENTER_LEFT);
20         this.setSpacing(20);
21
22         Map<String, ToggleButton> toggleButtons = new HashMap<>();
23         ToggleGroup toggleGroup = new ToggleGroup();
24
25         for (String nameOfButton : nameAndPaneMap.keySet()) {
26             ToggleButton toggleButton = new ToggleButton(nameOfButton);
27             toggleButtons.put(nameOfButton, toggleButton);
28             toggleButton.setPrefHeight(75);
29             toggleButton.setPrefWidth(75);
30             toggleButton.setWrapText(true);
31             toggleButton.setTextAlignment(TextAlignment.CENTER);
32             toggleButton.setTooltip(new Tooltip(toggleButton.getText()));
33             toggleButton.setToggleGroup(toggleGroup);
34             this.getChildren().add(toggleButton);
35         }
36
37         int i = 0;
38         for (String nameOfButton : toggleButtons.keySet()) {
39             int finalI = i;
40             toggleButtons.get(nameOfButton).setOnAction(event -> pane.changeSelected(
41                 ↪ nameOfButton));
42             i++;
43         }
44
45         public Pane getPane(String nameOfPane) {
46             return nameAndPaneMap.get(nameOfPane);
47         }
48     }
49 }
```

48 | }

Chapter 17: gui/MainApp.java

```
1 package gui;
2
3 import controller.Controller;
4 import javafx.application.Application;
5 import javafx.scene.Scene;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.Pane;
8 import javafx.stage.Stage;
9
10 public class MainApp extends Application {
11
12     private static final String TITLE = "Aarhus Bryghus - KASse System";
13     private static final int HEIGHT = 720;
14     private static final int WIDTH = HEIGHT * 16 / 9;
15
16     private static BorderPane borderPane;
17
18     public static void main(String[] args) {
19         Application.launch();
20     }
21
22     @Override
23     public void start(Stage stage) {
24         stage.setTitle(TITLE);
25
26         borderPane = new BorderPane();
27         this.initContent(borderPane);
28
29         Scene scene = new Scene(borderPane, WIDTH, HEIGHT);
30         scene.getStylesheets().add(getClass().getResource("css/app.css").toExternalForm());
31         stage.setScene(scene);
32         stage.setResizable(false); // Ikke sikkert skal sættes
33         stage.show();
34     }
35
36     @Override
37     public void stop() {
38         Controller.getInstance().saveToFile("data/Serilizabledata.ser");
39     }
40
41     public void initContent(BorderPane borderPane) {
42         Controller.getInstance().readFromFile("data/Serilizabledata.ser");
43         borderPane.setTop(Navigation.getInstance());
44         borderPane.setCenter(HomePane.getInstance()); // Set start pane
45     }
46
47     public static BorderPane getBorderPane() {
48         return borderPane;
49     }
50 }
```

```
49     }
50
51     public static void changePane(Pane pane, boolean remember) {
52         if (remember)
53             Navigation.addPaneToBackList((Pane) borderPane.getCenter());
54         Navigation.disableForwardButton();
55
56         borderPane.setCenter(pane);
57
58         Navigation.checkBackButton();
59     }
60 }
```

Chapter 18: gui/Navigation.java

```
1 package gui;
2
3 import gui.images.LoadImage;
4 import javafx.animation.Animation;
5 import javafx.animation.KeyFrame;
6 import javafx.animation.Timeline;
7 import javafx.scene.control.Button;
8 import javafx.scene.control.Label;
9 import javafx.scene.control.ToolBar;
10 import javafx.scene.image.Image;
11 import javafx.scene.image.ImageView;
12 import javafx.scene.layout.*;
13 import javafx.util.Duration;
14
15 import java.time.LocalDateTime;
16 import java.util.Arrays;
17 import java.util.LinkedList;
18 import java.util.List;
19 import java.util.Map;
20
21 /**
22  * The top bar for the application.
23  * Contains: home, back, forward and clock (etc).
24  */
25 public class Navigation extends ToolBar {
26     private static Navigation navigation = getInstance(); // eager loaded
27
28     private static final String IMAGE_FOLDER = "res/images/toolbar";
29
30     private static Button backButton;
31     private static Button forwardButton;
32     private static Label lblClock;
33
34     private static List<Pane> backList = new LinkedList<>();
35     private static List<Pane> forwardList = new LinkedList<>();
36
37     private Navigation() {
38         Map<String, Image> imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);
39         BorderPane borderPane = MainApp.getBorderPane();
40
41         Button homeButton = new Button();
42         homeButton.setOnAction(event -> {
43             if (borderPane.getCenter() != HomePane.getInstance())
44                 MainApp.changePane(HomePane.getInstance(), true);
45         });
46         if (imageMap != null && imageMap.containsKey("home")) {
47             ImageView imageView = new ImageView(imageMap.get("home"));
48             imageView.setFitWidth(20);
```

```

49         imageView.setFitHeight(20);
50         homeButton.setGraphic(imageView);
51         homeButton.setText("");
52     }
53
54     backButton = new Button("<-");
55
56     backButton.setDisable(true);
57     backButton.setOnAction(event -> {
58         forwardList.add((Pane) borderPane.getCenter());
59         forwardButton.setDisable(false);
60         int last = backList.size() - 1;
61         borderPane.setCenter(backList.get(last));
62         backList.remove(last);
63         if (backList.isEmpty())
64             backButton.setDisable(true);
65     });
66     if (imageMap != null && imageMap.containsKey("arrowLeft")) {
67         ImageView imageView = new ImageView(imageMap.get("arrowLeft"));
68         imageView.setFitWidth(20);
69         imageView.setFitHeight(20);
70         backButton.setGraphic(imageView);
71         backButton.setText("");
72     }
73
74     forwardButton = new Button("->");
75
76     forwardButton.setDisable(true);
77     forwardButton.setOnAction(event -> {
78         backList.add((Pane) borderPane.getCenter());
79         backButton.setDisable(false);
80         int last = forwardList.size() - 1;
81         borderPane.setCenter(forwardList.get(last));
82         forwardList.remove(last);
83         if (forwardList.isEmpty())
84             forwardButton.setDisable(true);
85     });
86     if (imageMap != null && imageMap.containsKey("arrowRight")) {
87         ImageView imageView = new ImageView(imageMap.get("arrowRight"));
88         imageView.setFitWidth(20);
89         imageView.setFitHeight(20);
90         forwardButton.setGraphic(imageView);
91         forwardButton.setText("");
92     }
93
94     lblClock = new Label();
95     lblClock.getStyleClass().add("clock");
96
97     Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(1), ev -> {
98         LocalDateTime time = LocalDateTime.now();
99         String timeString = String.format("%s:%s:%s", time.getHour(),
100             time.getMinute() > 9 ? time.getMinute() : "0" + time.getMinute(),
101             time.getSecond() > 9 ? time.getSecond() : "0" + time.getSecond());
102         lblClock.setText(timeString);
103     }));
104     timeline.setCycleCount(Animation.INDEFINITE);
105     timeline.play();
106

```

```
107     Region region = new Region();
108     HBox.setHgrow(region, Priority.ALWAYS);
109     this.getItems().addAll(Arrays.asList(homeButton, backButton, forwardButton, region,
110         ↪ lblClock));
111 }
112 public static Navigation getInstance() {
113     if (navigation == null)
114         navigation = new Navigation();
115     return navigation;
116 }
117
118 public static void addPaneToBackList(Pane pane) {
119     backList.add(pane);
120 }
121
122 public static void removeLastBackPane() {
123     int last = backList.size() - 1;
124     backList.remove(last);
125 }
126
127 public static void disableForwardButton() {
128     forwardButton.setDisable(true);
129     forwardList.clear();
130 }
131
132 public static void checkBackButton() {
133     if (!backList.isEmpty())
134         backButton.setDisable(false);
135 }
136 }
```

Chapter 19: gui/overviewpane/OverviewPane.java

```
1 package gui.overviewpane;
2
3 import gui.LeftSideBar;
4 import gui.LeftSideBarInterface;
5 import javafx.geometry.Insets;
6 import javafx.geometry.Pos;
7 import javafx.scene.control.Label;
8 import javafx.scene.layout.BorderPane;
9 import javafx.scene.layout.Pane;
10
11 import java.util.LinkedHashMap;
12 import java.util.Map;
13
14 public class OverviewPane extends BorderPane implements LeftSideBarInterface {
15
16     private static OverviewPane overviewPane;
17
18     private LeftSideBar leftSideBar;
19
20     public OverviewPane() {
21         this.setPadding(new Insets(20));
22
23         Label lblOverview = new Label("Oversigt");
24         lblOverview.getStyleClass().add("title");
25         this.setTop(lblOverview);
26
27         Map<String, Pane> navigationMap = new LinkedHashMap<>();
28         navigationMap.put("Produkter", ProductOverviewPane.getInstance());
29         navigationMap.put("Prislistener", PricelistOverviewPane.getInstance());
30         navigationMap.put("Salg", SaleOverviewPane.getInstance());
31         navigationMap.put("Udlejninger", RentOverviewPane.getInstance());
32         navigationMap.put("Rundvisninger", TourOverviewPane.getInstance());
33         leftSideBar = new LeftSideBar(navigationMap, this);
34         setLeft(leftSideBar);
35
36         changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
37     }
38
39     @Override
40     public void changeSelected(String nameOfButton) {
41         Label title = new Label(nameOfButton);
42         title.getStyleClass().add("title");
43         setAlignment(title, Pos.CENTER);
44         setTop(title);
45         setCenter(leftSideBar.getPane(nameOfButton));
46     }
47
48     public static OverviewPane getInstance() {
```



```
49         if (overviewPane == null)
50             overviewPane = new OverviewPane();
51         return overviewPane;
52     }
53 }
```

Chapter 20: gui/overviewpane/PricelistOverviewPane.java

```
1 package gui.overviewpane;
2
3 import controller.Controller;
4 import gui.helpers.ProductAndPriceHelper;
5 import javafx.scene.control.TreeItem;
6 import javafx.scene.control.TreeTableColumn;
7 import javafx.scene.control.TreeTableView;
8 import javafx.scene.control.cell.TreeItemPropertyValueFactory;
9 import javafx.scene.layout.GridPane;
10 import model.Pricelist;
11 import model.product.Product;
12
13 public class PricelistOverviewPane extends GridPane {
14
15     private static PricelistOverviewPane pricelistOverviewPane;
16     private TreeTableView<Object> pricelistTreeTable;
17     private final int treeTableColIndex = 0;
18     private final int treeTableRowIndex = 0;
19
20     public PricelistOverviewPane() {
21         getStyleClass().add("gridpane");
22
23         pricelistTreeTable = getPricelistTreeTable();
24
25         this.add(pricelistTreeTable, treeTableColIndex, treeTableRowIndex);
26     }
27
28     private TreeTableView<Object> getPricelistTreeTable() {
29         TreeTableView<Object> treeTableView = new TreeTableView<>();
30         treeTableView.setEditable(false);
31         treeTableView.setPrefHeight(700);
32         treeTableView.setPrefWidth(1100);
33         treeTableView.setColumnResizePolicy(TreeTableView.CONSTRAINED_RESIZE_POLICY);
34
35
36         TreeTableColumn<Object, String> column1 = new TreeTableColumn<>("Navn");
37         column1.setCellValueFactory(new TreeItemPropertyValueFactory<>("name"));
38         column1.setMinWidth(200);
39
40         TreeTableColumn<Object, String> column2 = new TreeTableColumn<>("Pris");
41         column2.setCellValueFactory(new TreeItemPropertyValueFactory<>("price"));
42         column2.setMinWidth(50);
43
44         TreeTableColumn<Object, String> column3 = new TreeTableColumn<>("Produktgruppe");
45         column3.setCellValueFactory(new TreeItemPropertyValueFactory<>("productGroup"));
46
47         treeTableView.getColumns().add(column1);
48         treeTableView.getColumns().add(column2);
```

```
49     treeTableView.getColumns().add(column3);
50
51     TreeItem<Object> root = new TreeItem<>(new Object());
52     treeTableView.setRoot(root);
53     treeTableView.setShowRoot(false);
54
55     for (Pricelist pricelist : Controller.getInstance().getPricelists()) {
56         TreeItem<Object> pricelistTreeItem = new TreeItem<>(pricelist);
57         pricelistTreeItem.setExpanded(false);
58         root.getChildren().add(pricelistTreeItem);
59         for (Product product : pricelist.getProductsWithPrice().keySet()) {
60             ProductAndPriceHelper productAndPriceHelper = new ProductAndPriceHelper(
61                 ↪ product, pricelist.getProductsWithPrice().get(product));
62             if (pricelist.getProductsWithPrice().containsKey(product)) {
63                 TreeItem<Object> productAndPriceHelperTreeItem = new TreeItem<>(
64                     ↪ productAndPriceHelper);
65                 pricelistTreeItem.getChildren().add(productAndPriceHelperTreeItem);
66             }
67         }
68     }
69     return treeTableView;
70 }
71
72 public void update() {
73     pricelistTreeTable = getPricelistTreeTable();
74     this.getChildren().remove(treeTableColIndex, treeTableRowIndex);
75     this.add(pricelistTreeTable, treeTableColIndex, treeTableRowIndex);
76 }
77
78 public static PricelistOverviewPane getInstance() {
79     if (pricelistOverviewPane == null)
80         pricelistOverviewPane = new PricelistOverviewPane();
81     return pricelistOverviewPane;
82 }
```

Chapter 21: gui/overviewpane/ProductOverviewPane.java

```
1 package gui.overviewpane;
2
3 import controller.Controller;
4 import javafx.scene.control.TreeItem;
5 import javafx.scene.control.TreeTableColumn;
6 import javafx.scene.control.TreeTableView;
7 import javafx.scene.control.cell.TreeItemPropertyValueFactory;
8 import javafx.scene.layout.GridPane;
9 import model.ProductGroup;
10 import model.enums.Unit;
11 import model.product.Bundle;
12 import model.product.Container;
13 import model.product.Product;
14 import model.product.rentable.Rentable;
15
16 public class ProductOverviewPane extends GridPane {
17
18     private static ProductOverviewPane productOverviewPane;
19     private TreeTableView<Object> productOverviewTreeTable;
20     private final int treeTableColIndex = 0;
21     private final int treeTableRowIndex = 0;
22
23     public ProductOverviewPane() {
24         getStyleClass().add("gridpane");
25
26         productOverviewTreeTable = getProductOverviewTreeTable();
27         this.add(productOverviewTreeTable, treeTableColIndex, treeTableRowIndex);
28     }
29
30     private TreeTableView<Object> getProductOverviewTreeTable() {
31         TreeTableView<Object> treeTableView = new TreeTableView<>();
32         treeTableView.setEditable(false);
33         treeTableView.setPrefHeight(700);
34         treeTableView.setPrefWidth(1100);
35         treeTableView.setColumnResizePolicy(TreeTableView.CONSTRAINED_RESIZE_POLICY);
36
37         TreeTableColumn<Object, String> column1 = new TreeTableColumn<>("Navn");
38         column1.setCellValueFactory(new TreeItemPropertyValueFactory<>("name"));
39         column1.setMinWidth(200);
40
41         TreeTableColumn<Object, String> column2 = new TreeTableColumn<>("ID");
42         column2.setCellValueFactory(new TreeItemPropertyValueFactory<>("id"));
43
44         TreeTableColumn<Object, String> column3 = new TreeTableColumn<>("Beskrivelse");
45         column3.setCellValueFactory(new TreeItemPropertyValueFactory<>("description"));
46         column3.setMinWidth(300);
47
48         TreeTableColumn<Object, Boolean> column4 = new TreeTableColumn<>("Udlejes");
```

```

49     column4.setCellValueFactory(new TreeItemPropertyValueFactory<>("rentable"));
50
51     TreeTableColumn<Object, String> column5 = new TreeTableColumn<>("Udlejningsafgift");
52     column5.setCellValueFactory(new TreeItemPropertyValueFactory<>("productDeposit"));
53
54     TreeTableColumn<Object, String> column6 = new TreeTableColumn<>("Størrelse");
55     column6.setCellValueFactory(new TreeItemPropertyValueFactory<>("size"));
56
57     TreeTableColumn<Object, String> column7 = new TreeTableColumn<>("Enhed");
58     column7.setCellValueFactory(new TreeItemPropertyValueFactory<>("unit"));
59
60     TreeTableColumn<Object, String> column8 = new TreeTableColumn<>("Sampkningsindhold"
61         ↪ );
62     column8.setCellValueFactory(new TreeItemPropertyValueFactory<>("bundleList"));
63
64     treeTableView.getColumns().add(column1);
65     treeTableView.getColumns().add(column2);
66     treeTableView.getColumns().add(column3);
67     treeTableView.getColumns().add(column4);
68     treeTableView.getColumns().add(column5);
69     treeTableView.getColumns().add(column6);
70     treeTableView.getColumns().add(column7);
71     treeTableView.getColumns().add(column8);
72
73     TreeItem<Object> root = new TreeItem<>(new Object());
74     treeTableView.setRoot(root);
75     treeTableView.setShowRoot(false);
76
77     for (ProductGroup productGroup : Controller.getInstance().getProductGroupList()) {
78
79         TreeItem<Object> productGroupTreeItem = new TreeItem<>(productGroup);
80         productGroupTreeItem.setExpanded(true);
81         root.getChildren().add(productGroupTreeItem);
82
83         for (Product product : Controller.getInstance().getProductList()) {
84
85             ProductExtended productExtended;
86             if (product instanceof Container) {
87                 productExtended = new ProductExtended(product, ((Container) product).
88                     ↪ getSize(),
89                     ((Container) product).getUnit());
90             } else if (product instanceof Bundle) {
91                 productExtended = new ProductExtended(product, product.toString());
92             } else {
93                 productExtended = new ProductExtended(product);
94             }
95
96             if (product.getProductGroup().equals(productGroup)) {
97                 TreeItem<Object> productExtendedTreeItem = new TreeItem<>(
98                     ↪ productExtended);
99                 productGroupTreeItem.getChildren().add(productExtendedTreeItem);
100             }
101         }
102     }
103     return treeTableView;
104
105     public void update() {

```

```

104     productOverviewTreeTable = getProductOverviewTreeTable();
105     this.getChildren().remove(treeTableColIndex, treeTableRowIndex);
106     this.add(productOverviewTreeTable, treeTableColIndex, treeTableRowIndex);
107 }
108
109 public static ProductOverviewPane getInstance() {
110     if (productOverviewPane == null)
111         productOverviewPane = new ProductOverviewPane();
112     return productOverviewPane;
113 }
114
115 //
116
117 /*
118  * Private class to display table Not pretty, but does the job - Cannot extend
119  * since we use id's
120  */
121
122 protected static class ProductExtended {
123
124     private Product product;
125
126     private double size;
127     private Unit unit;
128     private String bundleList;
129
130     public ProductExtended(Product product) {
131         this.product = product;
132         this.size = -1;
133     }
134
135     public ProductExtended(Product product, double size, Unit unit) {
136         this.product = product;
137         this.unit = unit;
138         this.size = size;
139     }
140
141     public ProductExtended(Product product, String bundleList) {
142         this.product = product;
143         this.size = -1;
144         this.bundleList = bundleList;
145     }
146
147     public String getName() {
148         return product.getName();
149     }
150
151     public int getId() {
152         return product.getId();
153     }
154
155     public String getDescription() {
156         return product.getDescription();
157     }
158
159     public boolean getRentable() {

```

```
160         return product.getRentableStrategy() instanceof Rentable;
161     }
162
163     public String getProductDeposit() {
164         return getRentable() ? String.valueOf(product.getRentableStrategy().getDeposit()
165             ↪ ) : "";
166     }
167
168     public String getSize() {
169         return size > -1 ? String.valueOf(size) : "";
170     }
171
172     public String getUnit() {
173         return unit != null ? unit.toString() : "";
174     }
175
176     public String getBundleList() {
177         return bundleList;
178     }
179 }
```

Chapter 22: gui/overviewpane/RentOverviewPane.java

```
1 package gui.overviewpane;
2
3 import controller.Controller;
4 import javafx.beans.property.SimpleStringProperty;
5 import javafx.scene.control.Button;
6 import javafx.scene.control.Label;
7 import javafx.scene.control.TableColumn;
8 import javafx.scene.control.TableView;
9 import javafx.scene.control.cell.PropertyValueFactory;
10 import javafx.scene.layout.GridPane;
11 import model.Rent;
12 import model.Sale;
13 import model.SalesLine;
14 import model.product.Tour;
15
16 import java.time.LocalDateTime;
17 import java.util.List;
18
19 public class RentOverviewPane extends GridPane {
20
21     private static RentOverviewPane rentOverviewPane;
22     private TableView<Object> rentOverviewTable;
23     private TableView<Object> rentSalesLineTable;
24
25     public RentOverviewPane() {
26         getStyleClass().add("gridpane");
27
28         rentOverviewTable = getRentOverviewTable();
29         update();
30         rentOverviewTable.getSelectionModel().selectedItemProperty().addListener((obs,
31             ↪ oldSelection, newSelection) -> {
32             if (newSelection != null) {
33                 updateSalesLineTable();
34             }
35         });
36         this.add(rentOverviewTable, 0, 0, 1, 2);
37
38         Label lblProductsInRent = new Label("Produkter i udlejning:");
39         this.add(lblProductsInRent, 1, 0);
40         rentSalesLineTable = getRentSalesLinesTable();
41         this.add(rentSalesLineTable, 1, 1);
42
43         Button btnUpdate = new Button("Opdatér");
44         btnUpdate.setOnAction(e -> this.update());
45         this.add(btnUpdate, 0, 4);
46     }
47 }
```



```

48 private TableView<Object> getRentOverviewTable() {
49     TableView<Object> tableView = new TableView<>();
50     tableView.setEditable(false);
51     tableView.setPrefHeight(700);
52     tableView.setPrefWidth(800);
53     tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
54
55     TableColumn<Object, String> column1 = new TableColumn<>("Tid");
56     column1.setCellValueFactory(new PropertyValueFactory<>("timestamp"));
57
58     TableColumn<Object, String> column2 = new TableColumn<>("ID");
59     column2.setCellValueFactory(new PropertyValueFactory<>("id"));
60
61     TableColumn<Object, String> column3 = new TableColumn<>("Status");
62     column3.setCellValueFactory(new PropertyValueFactory<>("saleState"));
63
64     TableColumn<Object, String> column4 = new TableColumn<>("Kontakt navn");
65     column4.setCellValueFactory(new PropertyValueFactory<>("contactName"));
66
67     TableColumn<Object, String> column5 = new TableColumn<>("Kontakt information");
68     column5.setCellValueFactory(new PropertyValueFactory<>("contactInformation"));
69
70     TableColumn<Object, String> column6 = new TableColumn<>("Leveringstidspunkt");
71     column6.setCellValueFactory(new PropertyValueFactory<>("deliveryDateAndTime"));
72
73     TableColumn<Object, String> column7 = new TableColumn<>("Returtidspunkt");
74     column7.setCellValueFactory(new PropertyValueFactory<>("returnDateAndTime"));
75
76     tableView.getColumns().add(column1);
77     tableView.getColumns().add(column2);
78     tableView.getColumns().add(column3);
79     tableView.getColumns().add(column4);
80     tableView.getColumns().add(column5);
81     tableView.getColumns().add(column6);
82     tableView.getColumns().add(column7);
83
84     return tableView;
85 }
86
87 private TableView<Object> getRentSaleLinesTable() {
88     TableView<Object> tableView = new TableView<>();
89     tableView.setEditable(false);
90     tableView.setPrefHeight(700);
91     tableView.setPrefWidth(280);
92     tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
93
94     TableColumn<Object, String> column1 = new TableColumn<>("Produkt");
95     column1.setCellValueFactory(new PropertyValueFactory<>("product"));
96
97     TableColumn<Object, String> column2 = new TableColumn<>("Produktgruppe");
98     column2.setCellValueFactory(
99         cellData -> {
100             SalesLine salesLine = ((SalesLine) cellData.getValue());
101             return new SimpleStringProperty(salesLine.getProduct().getProductGroup()
102                 ↪ .getName());
103         });
104     TableColumn<Object, String> column3 = new TableColumn<>("Antal");

```

```

105     column3.setCellValueFactory(new PropertyValueFactory<>("quantity"));
106
107     tableView.getColumns().add(column1);
108     tableView.getColumns().add(column2);
109     tableView.getColumns().add(column3);
110
111     return tableView;
112 }
113
114 private void updateSalesLineTable() {
115     RentExtended rentExtended = (RentExtended) rentOverviewTable.getSelectionModel().
116         ↪ getSelectedItem();
117     rentSalesLineTable.getItems().setAll(rentExtended.getSalesLines());
118 }
119
120 public void update() {
121     rentOverviewTable.getItems().clear();
122     for (Sale sale : Controller.getInstance().getSales()) {
123         if (sale instanceof Rent) {
124             if (sale.getSalesLines().stream().anyMatch(salesLine -> !(salesLine.
125                 ↪ getProduct() instanceof Tour))) {
126                 RentExtended rentExtended = new RentExtended((Rent) sale);
127                 rentOverviewTable.getItems().add(rentExtended);
128             }
129         }
130     }
131 }
132
133 public static RentOverviewPane getInstance() {
134     if (rentOverviewPane == null)
135         rentOverviewPane = new RentOverviewPane();
136     return rentOverviewPane;
137 }
138
139 //
140 ↪ -----
141 ↪
142
143 /*
144  * Private class to display table Not pretty, but does the job
145  */
146
147 protected static class RentExtended {
148
149     private Rent rent;
150
151     public RentExtended(Rent rent) {
152         this.rent = rent;
153     }
154
155     public int getId() {
156         return rent.getId();
157     }
158
159     public String getSaleState() {
160         return rent.getSaleState().toString();
161     }
162 }

```

```
159     public List<SalesLine> getSalesLines() {
160         return rent.getSalesLines();
161     }
162
163     public String getTimestamp() {
164         LocalDateTime time = rent.getTimestamp();
165         return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.
166             ↪ getMonthValue(), time.getYear(), time.getHour(), time.getMinute());
167     }
168
169     public String getContactName() {
170         return rent.getContactName();
171     }
172
173     public String getContactInformation() {
174         return rent.getContactInformation();
175     }
176
177     public String getDeliveryDateAndTime() {
178         LocalDateTime time = rent.getDeliveryDateAndTime();
179         return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.
180             ↪ getMonthValue(), time.getYear(), time.getHour(), time.getMinute());
181     }
182
183     public String getReturnDateAndTime() {
184         LocalDateTime time = rent.getReturnDateAndTime();
185         return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.
186             ↪ getMonthValue(), time.getYear(), time.getHour(), time.getMinute());
187     }
188 }
```

Chapter 23: gui/overviewpane/SaleOverviewPane.java

```
1 package gui.overviewpane;
2
3 import controller.Controller;
4 import javafx.scene.control.TableColumn;
5 import javafx.scene.control.TableView;
6 import javafx.scene.control.cell.PropertyValueFactory;
7 import javafx.scene.layout.GridPane;
8 import model.Rent;
9 import model.Sale;
10
11 import java.time.LocalDateTime;
12
13 public class SaleOverviewPane extends GridPane {
14
15     private static SaleOverviewPane saleOverviewPane;
16     private TableView<Object> saleOverviewTable;
17
18     public SaleOverviewPane() {
19         getStyleClass().add("gridpane");
20         saleOverviewTable = getSaleOverviewTable();
21         update(); // init table with content
22         this.add(saleOverviewTable, 0, 0);
23     }
24
25     private TableView<Object> getSaleOverviewTable() {
26         TableView<Object> tableView = new TableView<>();
27         tableView.setEditable(false);
28         tableView.setPrefHeight(700);
29         tableView.setPrefWidth(1100);
30         tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
31
32         TableColumn<Object, String> column1 = new TableColumn<>("Tid");
33         column1.setCellValueFactory(new PropertyValueFactory<>("timestamp"));
34
35         TableColumn<Object, String> column2 = new TableColumn<>("ID");
36         column2.setCellValueFactory(new PropertyValueFactory<>("id"));
37
38         TableColumn<Object, String> column3 = new TableColumn<>("Status");
39         column3.setCellValueFactory(new PropertyValueFactory<>("saleState"));
40
41         TableColumn<Object, String> column4 = new TableColumn<>("Udlejning");
42         column4.setCellValueFactory(new PropertyValueFactory<>("isRent"));
43
44         tableView.getColumns().add(column1);
45         tableView.getColumns().add(column2);
46         tableView.getColumns().add(column3);
47         tableView.getColumns().add(column4);
48     }
```

```

49         return tableView;
50     }
51
52
53     public void update() {
54         saleOverviewTable.getItems().clear();
55         for (Sale sale : Controller.getInstance().getSales()) {
56             SaleExtended saleExtended = new SaleExtended(sale);
57             saleOverviewTable.getItems().add(saleExtended);
58         }
59     }
60
61     public static SaleOverviewPane getInstance() {
62         if (saleOverviewPane == null)
63             saleOverviewPane = new SaleOverviewPane();
64         return saleOverviewPane;
65     }
66
67     //
68     ↪
69     ↪
70
71     /*
72     * Private class to display table Not pretty, but does the job
73     */
74
75     protected static class SaleExtended {
76
77         private Sale sale;
78
79         public SaleExtended(Sale sale) {
80             this.sale = sale;
81         }
82
83         public int getId() {
84             return sale.getId();
85         }
86
87         public String getSaleState() {
88             return sale.getSaleState().toString();
89         }
90
91         public String getTimestamp() {
92             LocalDateTime time = sale.getTimestamp();
93             return String.format("%s-%s-%s - %s:%s", time.getDayOfMonth(), time.
94                 ↪ getMonthValue(), time.getYear(),
95                 time.getHour(), time.getMinute());
96         }
97
98         public String getIsRent() {
99             return sale instanceof Rent ? "x" : "";
100         }
101     }

```

Chapter 24: gui/overviewpane/TourOverviewPane.java

```
1 package gui.overviewpane;
2
3 import controller.Controller;
4 import gui.helpers.Calendar;
5 import javafx.beans.property.SimpleStringProperty;
6 import javafx.geometry.Pos;
7 import javafx.scene.control.Button;
8 import javafx.scene.control.TableColumn;
9 import javafx.scene.control.TableView;
10 import javafx.scene.control.cell.PropertyValueFactory;
11 import javafx.scene.layout.GridPane;
12 import javafx.scene.layout.VBox;
13 import model.Rent;
14 import model.Sale;
15
16 import java.util.List;
17
18 public class TourOverviewPane extends GridPane {
19
20     private static TourOverviewPane tourOverviewPane;
21     private TableView<Sale> tourOverviewTable;
22     private VBox vbCalendar;
23
24     public TourOverviewPane() {
25         getStyleClass().add("gridpane");
26
27         tourOverviewTable = getTourOverviewTable();
28         this.add(tourOverviewTable, 0, 0);
29
30         vbCalendar = new VBox();
31         vbCalendar.setAlignment(Pos.CENTER);
32         this.add(vbCalendar, 1, 0);
33
34         Button btnUpdate = new Button("OpdatÃr");
35         btnUpdate.setOnAction(e -> this.update());
36         this.add(btnUpdate, 0, 1);
37         update(); // init table with content
38     }
39
40     public TableView<Sale> getTourOverviewTable() {
41         TableView<Sale> tableView = new TableView<>();
42         tableView.setEditable(false);
43         tableView.setPrefHeight(700);
44         tableView.setPrefWidth(800);
45         tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
46         TableColumn<Sale, String> column1 = new TableColumn<>("Tid");
47         column1.setCellValueFactory(new PropertyValueFactory<>("timestamp"));
48     }
```

```

49     TableColumn<Sale, String> column2 = new TableColumn<>("Salgs ID");
50     column2.setCellValueFactory(new PropertyValueFactory<>("id"));
51
52     TableColumn<Sale, String> column3 = new TableColumn<>("Antal");
53     column3.setCellValueFactory(
54         cellData -> new SimpleStringProperty(String.valueOf(cellData.getValue().
55             ↪ getSalesLines()
56                 .stream().mapToInt(salesLine -> salesLine.getQuantity())
57                 .sum())));
58
59     TableColumn<Sale, String> column4 = new TableColumn<>("Beskrivelse");
60     column4.setCellValueFactory(new PropertyValueFactory<>("description"));
61
62     TableColumn<Sale, String> column5 = new TableColumn<>("Kontaktperson");
63     column5.setCellValueFactory(new PropertyValueFactory<>("contactName"));
64
65     TableColumn<Sale, String> column6 = new TableColumn<>("Kontaktperson");
66     column6.setCellValueFactory(new PropertyValueFactory<>("contactInformation"));
67
68     TableColumn<Sale, String> column7 = new TableColumn<>("Rundvisningstidspunkt");
69     column7.setCellValueFactory(
70         cellData -> new SimpleStringProperty(
71             ((Rent) cellData.getValue()).getDeliveryDateAndTime().toLocalDate().
72             ↪ toString()));
73
74     tableView.getColumns().add(column1);
75     tableView.getColumns().add(column2);
76     tableView.getColumns().add(column3);
77     tableView.getColumns().add(column4);
78     tableView.getColumns().add(column5);
79     tableView.getColumns().add(column6);
80     tableView.getColumns().add(column7);
81
82     return tableView;
83 }
84
85 public void update() {
86     List<Sale> tourSale = Controller.getInstance().getTours();
87     // refreshes the calendar by removing and adding a new
88     vbCalendar.getChildren().clear();
89     vbCalendar.getChildren().add(new Calendar(tourSale));
90     tourOverviewTable.getItems().clear();
91     for (Sale sale : tourSale) {
92         tourOverviewTable.getItems().add(sale);
93     }
94 }
95
96 public static TourOverviewPane getInstance() {
97     if (tourOverviewPane == null)
98         tourOverviewPane = new TourOverviewPane();
99     return tourOverviewPane;
100 }

```

Chapter 25: gui/salepane/PaymentPane.java

```
1 package gui.salepane;
2
3 import controller.Controller;
4 import gui.MainApp;
5 import gui.Navigation;
6 import gui.images.LoadImage;
7 import gui.overviewpane.RentOverviewPane;
8 import gui.overviewpane.SaleOverviewPane;
9 import gui.overviewpane.TourOverviewPane;
10 import javafx.geometry.HPos;
11 import javafx.geometry.Pos;
12 import javafx.scene.control.*;
13 import javafx.scene.control.cell.PropertyValueFactory;
14 import javafx.scene.image.Image;
15 import javafx.scene.image.ImageView;
16 import javafx.scene.layout.GridPane;
17 import javafx.scene.layout.HBox;
18 import javafx.scene.layout.VBox;
19 import javafx.scene.text.Font;
20 import model.Pricelist;
21 import model.Rent;
22 import model.Sale;
23 import model.SalesLine;
24 import model.enums.PaymentMethod;
25 import model.enums.SaleState;
26
27 import java.time.LocalDateTime;
28 import java.util.Map;
29
30 public class PaymentPane extends GridPane {
31
32     private Sale sale;
33     private Pricelist pricelist;
34
35     private static final String IMAGE_FOLDER = "res/images/paymentpane";
36
37     private TextField txfAmount;
38     private ToggleGroup toggleGroup;
39     private Label lblMissingPaymentAmount;
40     private Button btnPayDelay;
41     private Button btnPayComplete;
42
43     private TableView<PaymentMethodAndAmount> transfersTableView;
44
45     private Label lblFinalPriceAmount;
46
47     public PaymentPane(Sale sale, Pricelist pricelist, String discount, String
        ↪ finalPriceAmount) {
```



```

48     this.getStyleClass().add("gridpane");
49
50     this.sale = sale;
51     this.pricelist = pricelist;
52
53     Label lblSaleLine = new Label("Salgslinje:");
54     lblSaleLine.setFont(new Font(16));
55     this.add(lblSaleLine, 3, 0);
56
57     // Table 2
58     TableViewSalesLines tableViewSalesLines = new TableViewSalesLines(sale);
59     this.add(tableViewSalesLines, 3, 1, 2, 4);
60
61     Label lblRentPrice = new Label("Udlejnings pris:");
62     this.add(lblRentPrice, 3, 5);
63     lblRentPrice.setMinWidth(200);
64
65     Label lblRentPriceAmount = new Label("0.0");
66     double totalDeposit = 0d;
67     for (SalesLine salesLine : sale.getSalesLines()) {
68         totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
        ↪ getTotalLineDeposit() : 0;
69     }
70     lblRentPriceAmount.setText(String.valueOf(totalDeposit));
71     setHalignment(lblRentPriceAmount, HPos.RIGHT);
72     this.add(lblRentPriceAmount, 4, 5);
73
74     Label lblTotalPrice = new Label("Samlet pris:");
75     this.add(lblTotalPrice, 3, 6);
76     lblTotalPrice.setMinWidth(200);
77
78     Label lblTotalPriceAmount = new Label();
79     setHalignment(lblTotalPriceAmount, HPos.RIGHT);
80     lblTotalPriceAmount.setText(String.valueOf(sale.totalSalePrice()));
81     this.add(lblTotalPriceAmount, 4, 6);
82
83     Label lblDiscount = new Label("Rabat:");
84     this.add(lblDiscount, 3, 7);
85
86     Label lblDiscountValue = new Label(discount);
87     this.add(lblDiscountValue, 4, 7);
88     lblDiscountValue.setMinWidth(100);
89     lblDiscountValue.setMaxWidth(100);
90     lblDiscountValue.setAlignment(Pos.CENTER_RIGHT);
91     setHalignment(lblDiscountValue, HPos.RIGHT);
92
93     Label txfFinalPrice = new Label("Slut pris:");
94     this.add(txfFinalPrice, 3, 8);
95
96     lblFinalPriceAmount = new Label();
97     lblFinalPriceAmount.setText(finalPriceAmount);
98     this.add(lblFinalPriceAmount, 4, 8);
99     setHalignment(lblFinalPriceAmount, HPos.RIGHT);
100
101     // Placeholder for layout
102     Button btnPlaceholder = new Button("Placeholder");
103     this.add(btnPlaceholder, 3, 9, 2, 1);
104     setHalignment(btnPlaceholder, HPos.CENTER);

```

```

105 btnPlaceholder.setDisable(true);
106 btnPlaceholder.setVisible(false);
107
108 Label lblPayment = new Label("Betalning:");
109 lblPayment.setFont(new Font(16));
110 lblPayment.setPrefWidth(800);
111 this.add(lblPayment, 0, 0, 3, 1);
112
113 VBox vbox = new VBox();
114 this.add(vbox, 0, 1, 1, 4);
115 vbox.setAlignment(Pos.CENTER_LEFT);
116 vbox.setSpacing(5);
117
118 Label lblPaymentMethod = new Label("Vælg betalingsmetode:");
119 vbox.getChildren().add(lblPaymentMethod);
120
121 Map<String, Image> imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);
122 toggleGroup = new ToggleGroup();
123
124 for (PaymentMethod paymentMethod : PaymentMethod.values()) {
125     ToggleButton toggleButton = new ToggleButton(paymentMethod.toString());
126     toggleButton.setUserData(paymentMethod);
127     toggleButton.setToggleGroup(toggleGroup);
128     toggleButton.setContentDisplay(ContentDisplay.LEFT);
129     toggleGroup.selectToggle(toggleButton); // Select last one added
130     vbox.getChildren().add(toggleButton);
131
132     String paymentMethodName = paymentMethod.toString();
133     if (imageMap != null && imageMap.containsKey(paymentMethodName)) {
134         ImageView imageView = new ImageView(imageMap.get(paymentMethodName));
135         imageView.setFitHeight(20);
136         imageView.setFitWidth(20);
137         toggleButton.setGraphic(imageView);
138     }
139 }
140
141 Label lblAmount = new Label("Beløb:");
142 vbox.getChildren().add(lblAmount);
143
144 String suggestedPayment = String
145     .valueOf(Double.parseDouble(lblFinalPriceAmount.getText()) - sale.getPayment
146         ↪ ().calcTotal());
147 txfAmount = new TextField(suggestedPayment);
148 txfAmount.setMaxWidth(120);
149 vbox.getChildren().add(txfAmount);
150
151 Button btnAddTransfer = new Button("Tilføj");
152 vbox.getChildren().add(btnAddTransfer);
153 btnAddTransfer.setOnAction(event -> addTransfer());
154
155 // Table 1
156 transfersTableView = new TableView<>();
157 this.add(transfersTableView, 1, 1, 2, 4);
158 transfersTableView.setPrefWidth(660);
159 transfersTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
160 transfersTableView.setEditable(false);
161 createTransfersTableView();

```

```

162 Label lblMissingPayment = new Label("Manglende beløb til betaling:");
163 this.add(lblMissingPayment, 1, 5);
164
165 lblMissingPaymentAmount = new Label();
166 if (sale instanceof Rent && !sale.getSaleState().equals(SaleState.DELAYED)) {
167     totalDeposit = 0d;
168     for (SalesLine salesLine : sale.getSalesLines()) {
169         totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
170             ↳ getTotalLineDeposit() : 0;
171     }
172     lblMissingPaymentAmount.setText(String.valueOf(totalDeposit));
173     txfAmount.setText(String.valueOf(totalDeposit));
174 } else {
175     lblMissingPaymentAmount.setText(suggestedPayment);
176 }
177 this.add(lblMissingPaymentAmount, 2, 5);
178 setHorizontalAlignment(lblMissingPaymentAmount, HPos.RIGHT);
179
180 HBox hBox = new HBox();
181 this.add(hBox, 0, 7, 3, 1);
182 hBox.setSpacing(100);
183 hBox.setAlignment(Pos.CENTER);
184
185 Button btnDelayPayment = new Button("Udskyd betaling");
186 hBox.getChildren().add(btnDelayPayment);
187 btnDelayPayment.setOnAction(event -> finishSale(SaleState.DELAYED));
188
189 btnPayDelay = new Button("Betal (del)");
190 hBox.getChildren().add(btnPayDelay);
191 btnPayDelay.setDisable(true);
192 btnPayDelay.setOnAction(event -> {
193     finishSale(SaleState.DELAYED);
194     SelectOngoingSale.getInstance().refreshOngoingSales();
195 });
196
197 btnPayComplete = new Button("Betal (afslut)");
198 hBox.getChildren().add(btnPayComplete);
199 btnPayComplete.setDisable(true);
200 btnPayComplete.setOnAction(event -> {
201     finishSale(SaleState.COMPLETED);
202     SelectOngoingSale.getInstance().refreshOngoingSales();
203 });
204 }
205
206 public void addTransfer() {
207     PaymentMethod paymentMethod = (PaymentMethod) toggleGroup.getSelectedToggle().
208     ↳ getUserData();
209     double amount = Double.parseDouble(txfAmount.getText());
210
211     sale.addTransfer(paymentMethod, amount);
212
213     PaymentMethodAndAmount paymentMethodAndAmount = new PaymentMethodAndAmount (
214     ↳ paymentMethod, amount);
215
216     transfersTableView.getItems().add(paymentMethodAndAmount);

```

```

216     double missingPaymentAmount = Double.parseDouble(lblMissingPaymentAmount.getText())
217         ↪ - amount;
218     lblMissingPaymentAmount.setText(String.valueOf(missingPaymentAmount));
219
220     if (missingPaymentAmount <= 0.0)
221         btnPayDelay.setDisable(false);
222     if (sale.getPayment().calcTotal() >= Double.parseDouble(lblFinalPriceAmount.getText()
223         ↪ ()))
224         btnPayComplete.setDisable(false);
225
226 }
227
228 public void createTransfersTableView() {
229     TableColumn<PaymentMethodAndAmount, String> column1 = new TableColumn<>("
230         ↪ Betalingsmetode");
231     column1.setCellValueFactory(new PropertyValueFactory<>("paymentMethod"));
232     column1.setStyle("-fx-alignment: CENTER-LEFT");
233
234     TableColumn<PaymentMethodAndAmount, Double> column2 = new TableColumn<>("Beløb");
235     column2.setCellValueFactory(new PropertyValueFactory<>("amount"));
236     column2.setStyle("-fx-alignment: CENTER-RIGHT");
237
238     transfersTableView.getColumns().add(column1);
239     transfersTableView.getColumns().add(column2);
240
241     for (PaymentMethod paymentMethod : sale.getPayment().getTransfers().keySet()) {
242         transfersTableView.getItems().add(
243             new PaymentMethodAndAmount(paymentMethod, sale.getPayment().getTransfers
244                 ↪ ().get(paymentMethod)));
245     }
246 }
247
248 private void finishSale(SaleState saleState) {
249     if (Controller.getInstance().getSales().contains(sale)) {
250         sale.setAgreedPrice(Double.parseDouble(lblFinalPriceAmount.getText()));
251         sale.setSaleState(saleState);
252         return; // Ikke opret salg hvis allerede eksisterer
253     }
254     Sale savedSale;
255     if (sale instanceof Rent) {
256         String contactName = ((Rent) sale).getContactName();
257         String contactInformation = ((Rent) sale).getContactInformation();
258         LocalDateTime deliveryDateAndTime = ((Rent) sale).getDeliveryDateAndTime();
259         LocalDateTime returnDateAndTime = ((Rent) sale).getReturnDateAndTime();
260         savedSale = Controller.getInstance().createRent(contactName, contactInformation,
261             ↪ deliveryDateAndTime,
262             returnDateAndTime);
263         RentOverviewPane.getInstance().update();
264         TourOverviewPane.getInstance().update();
265     } else {
266         savedSale = Controller.getInstance().createSale();
267         SaleOverviewPane.getInstance().update();
268     }
269     savedSale.setAgreedPrice(Double.parseDouble(lblFinalPriceAmount.getText()));
270     savedSale.setSaleState(saleState);
271     for (SalesLine salesLine : sale.getSalesLines()) {
272         savedSale.updateSalesLine(salesLine.getProduct(), salesLine.getQuantity(),
273             salesLine.getTotalLinePrice() / salesLine.getQuantity());

```

```

269     }
270     for (PaymentMethod paymentMethod : sale.getPayment().getTransfers().keySet()) {
271         savedSale.getPayment().addTransfer(paymentMethod, sale.getPayment().getTransfers
           ↪ ().get(paymentMethod));
272     }
273
274     sale = null;
275
276     Navigation.removeLastBackPane();
277     MainApp.changePane(new SalePane(null, pricelist), false);
278     // 'false'
279 }
280
281 //
           ↪ -----
           ↪
282
283 /*
284  * Class to enable gui (Cannot be private)
285  */
286 protected static class PaymentMethodAndAmount {
287
288     private PaymentMethod paymentMethod;
289     private double amount;
290
291     public PaymentMethodAndAmount(PaymentMethod paymentMethod, double amount) {
292         this.paymentMethod = paymentMethod;
293         this.amount = amount;
294     }
295
296     public PaymentMethod getPaymentMethod() {
297         return paymentMethod;
298     }
299
300     public double getAmount() {
301         return amount;
302     }
303 }
304
305 }

```

Chapter 26: gui/salepane/RentInformationPane.java

```
1 package gui.salepane;
2
3 import controller.Controller;
4 import gui.MainApp;
5 import gui.helpers.ProductDatePicker;
6 import gui.helpers.ValidationHelper;
7 import javafx.geometry.HPos;
8 import javafx.geometry.Pos;
9 import javafx.geometry.VPos;
10 import javafx.scene.control.*;
11 import javafx.scene.layout.GridPane;
12 import javafx.scene.text.Font;
13 import model.Pricelist;
14 import model.Rent;
15 import model.Sale;
16 import model.SalesLine;
17
18 import java.time.LocalDateTime;
19
20 public class RentInformationPane extends GridPane {
21
22     private Sale sale;
23     private Pricelist pricelist;
24     private String discount;
25     private boolean tour;
26
27     private Label lblFinalPriceAmount;
28
29     private TextField txfContactName;
30     private TextArea txaContactInformation;
31     private DatePicker dpDeliveryDateAndTime;
32     private DatePicker dpReturnDateAndTime;
33     private ProductDatePicker productDatePicker;
34
35     public RentInformationPane(Sale sale, Pricelist pricelist, String discount, String
36         ↪ finalPriceAmount, boolean tour) {
37         this.getStyleClass().add("gridpane");
38
39         this.sale = sale;
40         this.pricelist = pricelist;
41         this.discount = discount;
42         this.tour = tour;
43
44         Label lblRentInformation = new Label("Kontakt information:");
45         lblRentInformation.setFont(new Font(16));
46         lblRentInformation.setPrefWidth(800);
47         this.add(lblRentInformation, 0, 0, 3, 1);
```

```

48 //
49 Label lblContactName = new Label("Kontakt navn:");
50 this.add(lblContactName, 0, 1);
51 txfContactName = new TextField();
52 this.add(txfContactName, 1, 1);
53
54 Label lblContactInformation = new Label("Kontakt information:");
55 setValignment(lblContactInformation, VPos.TOP);
56 this.add(lblContactInformation, 0, 2);
57 txaContactInformation = new TextArea();
58 txaContactInformation.setPrefColumnCount(3);
59 this.add(txaContactInformation, 1, 2);
60
61 if (!tour) {
62     Label lblDeliveryDateAndTime = new Label("Afleverings tidspunkt:");
63     this.add(lblDeliveryDateAndTime, 0, 3);
64     dpDeliveryDateAndTime = new DatePicker();
65     this.add(dpDeliveryDateAndTime, 1, 3);
66
67     Label lblReturnDateAndTime = new Label("Returnerings tidspunkt:");
68     this.add(lblReturnDateAndTime, 0, 4);
69     dpReturnDateAndTime = new DatePicker();
70     this.add(dpReturnDateAndTime, 1, 4);
71 } else {
72     Label lblReturnDateAndTime = new Label("Rundvisnings tidspunkt:");
73     this.add(lblReturnDateAndTime, 0, 3);
74     productDatePicker = new ProductDatePicker(Controller.getInstance().getRents());
75     this.add(productDatePicker, 1, 3);
76 }
77
78 Button btnDelayPayment = new Button("Betalning");
79 this.add(btnDelayPayment, 0, 7, 3, 1);
80 setHalignment(btnDelayPayment, HPos.CENTER);
81 btnDelayPayment.setOnAction(event -> goToPayment());
82
83 Label lblSaleLine = new Label("Salgslinje:");
84 lblSaleLine.setFont(new Font(16));
85 this.add(lblSaleLine, 3, 0);
86
87 // Table 2
88 TableViewSalesLines salesLineTableView = new TableViewSalesLines(sale);
89 this.add(salesLineTableView, 3, 1, 2, 4);
90
91 Label lblRentPrice = new Label("Udlejnings pris:");
92 this.add(lblRentPrice, 3, 5);
93 lblRentPrice.setMinWidth(200);
94
95 Label lblRentPriceAmount = new Label("0.0");
96 double totalDeposit = 0d;
97 for (SalesLine salesLine : sale.getSalesLines()) {
98     totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
99         ↪ getTotalLineDeposit() : 0;
100 }
101 lblRentPriceAmount.setText(String.valueOf(totalDeposit));
102 setHalignment(lblRentPriceAmount, HPos.RIGHT);
103 this.add(lblRentPriceAmount, 4, 5);
104
105 Label lblTotalPrice = new Label("Samlet pris:");

```

```

105     this.add(lblTotalPrice, 3, 6);
106     lblTotalPrice.setMinWidth(200);
107
108     Label lblTotalPriceAmount = new Label();
109     setHalignment(lblTotalPriceAmount, HPos.RIGHT);
110     lblTotalPriceAmount.setText(String.valueOf(sale.totalSalePrice()));
111     this.add(lblTotalPriceAmount, 4, 6);
112
113     Label lblDiscount = new Label("Rabat:");
114     this.add(lblDiscount, 3, 7);
115
116     Label lblDiscountValue = new Label(discount);
117     this.add(lblDiscountValue, 4, 7);
118     lblDiscountValue.setMinWidth(100);
119     lblDiscountValue.setMaxWidth(100);
120     lblDiscountValue.setAlignment(Pos.CENTER_RIGHT);
121     setHalignment(lblDiscountValue, HPos.RIGHT);
122
123     Label txfFinalPrice = new Label("Slut pris:");
124     this.add(txfFinalPrice, 3, 8);
125
126     lblFinalPriceAmount = new Label();
127     lblFinalPriceAmount.setText(finalPriceAmount);
128     this.add(lblFinalPriceAmount, 4, 8);
129     setHalignment(lblFinalPriceAmount, HPos.RIGHT);
130
131     // Placeholder for layout
132     Button btnPlaceholder = new Button("Placeholder");
133     this.add(btnPlaceholder, 3, 9, 2, 1);
134     setHalignment(btnPlaceholder, HPos.CENTER);
135     btnPlaceholder.setDisable(true);
136     btnPlaceholder.setVisible(false);
137 }
138
139 private void goToPayment() {
140     boolean hasError = false;
141     String contactName = txfContactName.getText().trim();
142     if (!ValidationHelper.isStringBetween0to30characters(contactName)) {
143         txfContactName.getStyleClass().add("error");
144         hasError = true;
145     }
146     String contactInformation = txaContactInformation.getText().trim();
147     if (!contactInformation.matches("((.|\\n)*)") || contactInformation.isEmpty()) {
148         txaContactInformation.getStyleClass().add("error");
149         hasError = true;
150     }
151
152     if (!hasError) {
153         Rent rent;
154         if (!tour) {
155             LocalDateTime deliveryDateAndTime = dpDeliveryDateAndTime.getValue().
156                 ↪ atStartOfDay();
157             LocalDateTime returnDateAndTime = dpReturnDateAndTime.getValue().
158                 ↪ atStartOfDay();
159             rent = new Rent(contactName, contactInformation, deliveryDateAndTime,
160                 ↪ returnDateAndTime);
161         } else {
162             LocalDateTime tourDateAndTime = productDatePicker.getValue().atStartOfDay();

```



```
160         rent = new Rent(contactName, contactInformation, tourDateAndTime,
161                           ↪ tourDateAndTime);
162     }
163     for (SalesLine salesLine : sale.getSalesLines()) {
164         rent.updateSalesLine(salesLine.getProduct(), salesLine.getQuantity(),
165                             ↪ salesLine.getPrice());
166     }
167     MainApp.changePane(new PaymentPane(rent, pricelist, discount,
168                                       ↪ lblFinalPriceAmount.getText()), true);
169 }
```

Chapter 27: gui/salepane/SaleOptionsPane.java

```
1 package gui.salepane;
2
3 import gui.LeftSideBar;
4 import gui.LeftSideBarInterface;
5 import javafx.geometry.Insets;
6 import javafx.geometry.Pos;
7 import javafx.scene.control.Label;
8 import javafx.scene.layout.BorderPane;
9 import javafx.scene.layout.Pane;
10
11 import java.util.LinkedHashMap;
12 import java.util.Map;
13
14 public class SaleOptionsPane extends BorderPane implements LeftSideBarInterface {
15
16     private static SaleOptionsPane saleOptionsPane;
17
18     private LeftSideBar leftSideBar;
19
20     public SaleOptionsPane() {
21         this.setPadding(new Insets(20));
22
23         //Creating panes
24         Map<String, Pane> navigationMap = new LinkedHashMap<>();
25         navigationMap.put("Salg/udlejning", SelectPricelistPane.getInstance());
26         navigationMap.put("Uafsluttede salg", SelectOngoingSale.getInstance());
27
28         leftSideBar = new LeftSideBar(navigationMap, this);
29         setLeft(leftSideBar);
30         changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
31     }
32
33     @Override
34     public void changeSelected(String nameOfButton) {
35         SelectOngoingSale.getInstance().refreshOngoingSales();
36         Label title = new Label(nameOfButton);
37         title.getStyleClass().add("title");
38         setAlignment(title, Pos.CENTER);
39         setTop(title);
40         setCenter(leftSideBar.getPane(nameOfButton));
41     }
42
43     public static SaleOptionsPane getInstance() {
44         if (saleOptionsPane == null)
45             saleOptionsPane = new SaleOptionsPane();
46         return saleOptionsPane;
47     }
48 }
```

Chapter 28: gui/salepane/SalePane.java

```
1 package gui.salepane;
2
3 import gui.MainApp;
4 import gui.helpers.ProductAndPriceHelper;
5 import gui.helpers.ValidationHelper;
6 import gui.images.LoadImage;
7 import javafx.beans.property.SimpleStringProperty;
8 import javafx.collections.FXCollections;
9 import javafx.collections.ObservableList;
10 import javafx.geometry.HPos;
11 import javafx.geometry.Pos;
12 import javafx.scene.control.*;
13 import javafx.scene.control.cell.PropertyValueFactory;
14 import javafx.scene.control.cell.TextFieldTableCell;
15 import javafx.scene.image.Image;
16 import javafx.scene.image.ImageView;
17 import javafx.scene.layout.GridPane;
18 import javafx.scene.layout.HBox;
19 import javafx.scene.text.Font;
20 import javafx.scene.text.TextAlignment;
21 import javafx.util.converter.DoubleStringConverter;
22 import model.Pricelist;
23 import model.Sale;
24 import model.SalesLine;
25 import model.enums.SaleState;
26 import model.product.Product;
27 import model.product.Tour;
28 import model.product.rentable.Rentable;
29
30 import java.text.DecimalFormat;
31 import java.util.Map;
32
33 public class SalePane extends GridPane {
34
35     private static final String IMAGE_FOLDER = "res/images/salespane";
36     private Map<String, Image> imageMap;
37
38     private Sale sale;
39     private Pricelist pricelist;
40
41     private TableView<ProductAndPriceHelper> productTableView;
42     private TableView<SalesLine> salesLineTableView;
43
44     private Label lblRentPriceAmount;
45     private Label lblTotalPriceAmount;
46     private TextField txfDiscount;
47     private Label lblFinalPriceAmount;
48 }
```

```
49 private Button btnCompleteSale;
50
51 public SalePane(Sale sale, Pricelist pricelist) {
52     this.getStyleClass().add("gridpane");
53
54     this.sale = sale;
55     this.pricelist = pricelist;
56
57     // Load images
58     imageMap = LoadImage.loadImagesFromFolder(IMAGE_FOLDER);
59
60     Label lblPricelist = new Label("Prisliste:");
61     lblPricelist.setFont(new Font(16));
62     this.add(lblPricelist, 0, 0);
63
64     // Table 1
65     productTableView = new TableView<>();
66     this.add(productTableView, 0, 1, 1, 6);
67     productTableView.setPrefHeight(600);
68     productTableView.setPrefWidth(800);
69     productTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
70     productTableView.setEditable(false);
71     createProductAndPriceHelperTable();
72
73     Label lblSaleLine = new Label("Salgslinje:");
74     lblSaleLine.setFont(new Font(16));
75     this.add(lblSaleLine, 1, 0);
76
77     // Table 2
78     salesLineTableView = new TableView<>();
79     this.add(salesLineTableView, 1, 1, 2, 1);
80     salesLineTableView.setPrefHeight(450);
81     salesLineTableView.setPrefWidth(400);
82     salesLineTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
83     salesLineTableView.setEditable(true);
84     salesLineTableView.setPlaceholder(new Label("Ingen produkter"));
85     createSalesLineTable();
86
87     Label lblRentPrice = new Label("Udlejnings pris:");
88     this.add(lblRentPrice, 1, 2);
89     lblRentPrice.setMinWidth(200);
90
91     lblRentPriceAmount = new Label("0.0");
92     setHalignment(lblRentPriceAmount, HPos.RIGHT);
93     this.add(lblRentPriceAmount, 2, 2);
94
95     Label lblTotalPrice = new Label("Samlet pris:");
96     this.add(lblTotalPrice, 1, 3);
97     lblTotalPrice.setMinWidth(200);
98
99     lblTotalPriceAmount = new Label("0.0");
100     setHalignment(lblTotalPriceAmount, HPos.RIGHT);
101     this.add(lblTotalPriceAmount, 2, 3);
102
103     Label lblDiscount = new Label("Rabat: (kan være %)");
104     this.add(lblDiscount, 1, 4);
105
106     txfDiscount = new TextField("0.0");
```

```

107     this.add(txfDiscount, 2, 4);
108     txfDiscount.setMinWidth(100);
109     txfDiscount.setMaxWidth(100);
110     txfDiscount.setAlignment(Pos.CENTER_RIGHT);
111     setHalignment(txfDiscount, HPos.RIGHT);
112
113     txfDiscount.textProperty().addListener((observable -> {
114         double finalPrice;
115         if (!txfDiscount.getText().isEmpty() && ValidationHelper.isPercentNumber(
116             ↳ txfDiscount.getText())) {
117             String substring = txfDiscount.getText().substring(0, txfDiscount.getText().
118                 ↳ length() - 1);
119             double procent = (100 - Double.parseDouble(substring)) / 100;
120             finalPrice = Double.parseDouble(lblTotalPriceAmount.getText()) * procent;
121         } else if (!txfDiscount.getText().isEmpty() && ValidationHelper.isNumber(
122             ↳ txfDiscount.getText())) {
123             finalPrice = Double.parseDouble(lblTotalPriceAmount.getText())
124                 - Double.parseDouble(txfDiscount.getText());
125         } else
126             finalPrice = Double.parseDouble(lblTotalPriceAmount.getText());
127         lblFinalPriceAmount.setText(String.valueOf(new DecimalFormat("#0.00").format(
128             ↳ finalPrice)));
129     }));
130
131     Label txfFinalPrice = new Label("Slut pris:");
132     this.add(txfFinalPrice, 1, 5);
133
134     lblFinalPriceAmount = new Label("0.0");
135     this.add(lblFinalPriceAmount, 2, 5);
136     setHalignment(lblFinalPriceAmount, HPos.RIGHT);
137
138     btnCompleteSale = new Button("Gå til betaling");
139     this.add(btnCompleteSale, 1, 6, 2, 1);
140     btnCompleteSale.setOnAction(event -> goToPayment());
141     setHalignment(btnCompleteSale, HPos.CENTER);
142     btnCompleteSale.setDisable(true);
143
144     if (sale != null) {
145         updateSalesLineGui();
146         lblFinalPriceAmount.setText(String.valueOf(sale.getAgreedPrice()));
147     }
148 }
149
150 private void createProductAndPriceHelperTable() {
151     TableColumn<ProductAndPriceHelper, String> column1 = new TableColumn<>("Produkt");
152     column1.setCellValueFactory(new PropertyValueFactory<>("product"));
153     column1.setStyle("-fx-alignment: CENTER-LEFT");
154
155     TableColumn<ProductAndPriceHelper, String> column2 = new TableColumn<>("
156         ↳ Produktgruppe");
157     column2.setCellValueFactory(new PropertyValueFactory<>("productGroup"));
158     column2.setStyle("-fx-alignment: CENTER");
159
160     TableColumn<ProductAndPriceHelper, String> column3 = new TableColumn<>("");
161     column3.setCellFactory(param -> new TableCell<ProductAndPriceHelper, String>() {
162         final Button btnMinus = new Button("-");
163         final TextField txfAmount = new TextField("1");
164         final Button btnPlus = new Button("+");

```

```

160
161     @Override
162     public void updateItem(String item, boolean empty) {
163         setGraphic(null);
164         if (!empty) {
165             btnMinus.setOnAction(event -> {
166                 ProductAndPriceHelper ProductAndPriceHelper = getTableView().
167                     ↪ getItems().get(getIndex());
168                 if (ValidationHelper.isNumber(txfAmount.getText()))
169                     updateSalesLine(ProductAndPriceHelper.getProduct(), -Integer.
170                         ↪ parseInt(txfAmount.getText()), ProductAndPriceHelper.
171                         ↪ getPrice());
172             });
173             if (imageMap != null && imageMap.containsKey("minus")) {
174                 ImageView imageView = new ImageView(imageMap.get("minus"));
175                 imageView.setFitWidth(13);
176                 imageView.setFitHeight(13);
177                 btnPlus.setGraphic(imageView);
178                 btnPlus.setText(""); // Remove text and use button instead
179             }
180
181             txfAmount.setMinWidth(35);
182             txfAmount.setMaxWidth(35);
183             txfAmount.setAlignment(Pos.CENTER);
184
185             btnPlus.setOnAction(event -> {
186                 ProductAndPriceHelper ProductAndPriceHelper = getTableView().
187                     ↪ getItems().get(getIndex());
188                 if (ValidationHelper.isNumber(txfAmount.getText()))
189                     updateSalesLine(ProductAndPriceHelper.getProduct(), Integer.
190                         ↪ parseInt(txfAmount.getText()), ProductAndPriceHelper.
191                         ↪ getPrice());
192             });
193             if (imageMap != null && imageMap.containsKey("plus")) {
194                 ImageView imageView = new ImageView(imageMap.get("plus"));
195                 imageView.setFitWidth(13);
196                 imageView.setFitHeight(13);
197                 btnPlus.setGraphic(imageView);
198                 btnPlus.setText(""); // Remove text and use button instead
199             }
200
201             HBox hBox = new HBox();
202             hBox.setSpacing(5);
203             hBox.setAlignment(Pos.CENTER);
204             hBox.getChildren().addAll(btnMinus, txfAmount, btnPlus);
205             setGraphic(hBox);
206         }
207     });
208     column3.setMinWidth(120);
209     column3.setMaxWidth(120);
210
211     TableColumn<ProductAndPriceHelper, Double> column4 = new TableColumn<>("Pris");
212
213     TableColumn<ProductAndPriceHelper, String> column5 = new TableColumn<>("Udlejning");
214     column5.setCellValueFactory(new PropertyValueFactory<>("deposit"));
215     column5.setStyle("-fx-alignment: CENTER-RIGHT");
216     column5.setMinWidth(120);

```

```

212     column5.setMaxWidth(120);
213
214     TableColumn<ProductAndPriceHelper, Double> column6 = new TableColumn<>("Salg");
215     column6.setCellValueFactory(new PropertyValueFactory<>("price"));
216     column6.setStyle("-fx-alignment: CENTER-RIGHT");
217     column6.setMinWidth(120);
218     column6.setMaxWidth(120);
219
220     productTableView.getColumns().add(column1);
221     productTableView.getColumns().add(column2);
222     productTableView.getColumns().add(column3);
223     productTableView.getColumns().add(column4);
224
225     column4.getColumns().add(column5);
226     column4.getColumns().add(column6);
227
228     ObservableList<ProductAndPriceHelper> productAndPriceHelperObservableList;
229     productAndPriceHelperObservableList = FXCollections
230         .observableList(ProductAndPriceHelper.parseToProductAndPrice(pricelist.
231             ↪ getProductsWithPrice()));
232     productTableView.setItems(productAndPriceHelperObservableList);
233 }
234
235 public void createSalesLineTable() {
236     TableColumn<SalesLine, String> column1 = new TableColumn<>("Navn");
237     column1.setCellValueFactory(new PropertyValueFactory<>("product"));
238
239     TableColumn<SalesLine, Integer> column2 = new TableColumn<>("Antal");
240     column2.setCellValueFactory(new PropertyValueFactory<>("quantity"));
241     column2.setStyle("-fx-alignment: CENTER");
242     column2.setMinWidth(60);
243     column2.setMaxWidth(60);
244
245     TableColumn<SalesLine, Double> column3 = new TableColumn<>("Samlet Pris");
246
247     TableColumn<SalesLine, String> column4 = new TableColumn<>("Udlejning");
248     column4.setCellValueFactory(cellData -> cellData.getValue().getTotalLineDeposit() >=
249         ↪ 0
250         ? new SimpleStringProperty(String.valueOf(cellData.getValue().
251             ↪ getTotalLineDeposit()))
252         : new SimpleStringProperty(""));
253     column4.setStyle("-fx-alignment: CENTER-RIGHT");
254
255     TableColumn<SalesLine, Double> column5 = new TableColumn<>("Salg");
256     column5.setCellValueFactory(new PropertyValueFactory<>("totalLinePrice"));
257     column5.setStyle("-fx-alignment: CENTER-RIGHT");
258
259     TableColumn<SalesLine, String> column6 = new TableColumn<>("");
260     column6.setCellFactory(param -> new TableCell<SalesLine, String>() {
261         final Button btnRemove = new Button("X");
262
263         @Override
264         public void updateItem(String item, boolean empty) {
265             setGraphic(null);
266             if (!empty) {
267                 btnRemove.setOnAction(event -> {
268                     SalesLine salesLine = getTableView().getItems().get(getIndex());

```

```

266         updateSalesLine(salesLine.getProduct(), -salesLine.getQuantity(),
267             ↪ salesLine.getPrice());
268     });
269     if (imageMap != null && imageMap.containsKey("cross")) {
270         ImageView imageView = new ImageView(imageMap.get("cross"));
271         imageView.setFitWidth(13);
272         imageView.setFitHeight(13);
273         btnRemove.setGraphic(imageView);
274         btnRemove.setText(""); // Remove text and use button instead
275     }
276     setAlignment(Pos.CENTER);
277     setGraphic(btnRemove);
278 }
279 });
280 column6.setMinWidth(40);
281 column6.setMaxWidth(40);
282
283 Label lblNyStykPris = new Label("Ny Styk Pris");
284 lblNyStykPris.setWrapText(true);
285 lblNyStykPris.setTextAlignment(TextAlignment.CENTER);
286 TableColumn<SalesLine, Double> column7 = new TableColumn<>();
287 column7.setCellValueFactory(new PropertyValueFactory<>("price"));
288 column7.setCellFactory(TextFieldTableCell.forTableColumn(new DoubleStringConverter()
289     ↪ ));
290 column7.setOnEditCommit(event -> {
291     event.getRowValue().setNewPrice(event.getNewValue());
292     updateSalesLineGui();
293 });
294 column7.setEditable(true);
295 column7.setMinWidth(60);
296 column7.setMaxWidth(60);
297 column7.setStyle("-fx-alignment: CENTER-RIGHT");
298 column7.setGraphic(lblNyStykPris);
299
300 salesLineTableView.getColumns().add(column1);
301 salesLineTableView.getColumns().add(column2);
302 salesLineTableView.getColumns().add(column3);
303 column3.getColumns().add(column4);
304 column3.getColumns().add(column5);
305 salesLineTableView.getColumns().add(column6);
306 salesLineTableView.getColumns().add(column7);
307 }
308
309 private void updateSalesLine(Product product, int quantity, double price) {
310     if (sale == null)
311         sale = new Sale();
312
313     sale.updateSalesLine(product, quantity, price);
314     updateSalesLineGui();
315 }
316
317 private void updateSalesLineGui() {
318     salesLineTableView.getItems().setAll(sale.getSalesLines());
319     double totalDeposit = 0d;
320     for (SalesLine salesLine : sale.getSalesLines()) {
321         totalDeposit += salesLine.getTotalLineDeposit() >= 0 ? salesLine.
322             ↪ getTotalLineDeposit() : 0;

```



```

321     }
322     lblRentPriceAmount.setText(String.valueOf(totalDeposit));
323     lblTotalPriceAmount.setText(String.valueOf(sale.totalSalePrice()));
324     double finalPrice = !txfDiscount.getText().isEmpty() && ValidationHelper.isNumber(
325         ↪ txfDiscount.getText())
326         ? Double.parseDouble(lblTotalPriceAmount.getText()) - Double.parseDouble(
327             ↪ txfDiscount.getText())
328         : Double.parseDouble(lblTotalPriceAmount.getText());
329     lblFinalPriceAmount.setText(Double.toString(finalPrice));
330
331     if (sale.getSalesLines().isEmpty())
332         btnCompleteSale.setDisable(true);
333     else
334         btnCompleteSale.setDisable(false);
335
336     if (containsRent())
337         btnCompleteSale.setText("Videre");
338     else
339         btnCompleteSale.setText("Til betaling");
340 }
341
342 private void goToPayment() {
343     // Og hvis ikke en tour der er igangværende
344     if (containsRent() && !sale.getState().equals(State.DELAYED))
345         MainApp.changePane(new RentInformationPane(sale, pricelist, txfDiscount.getText()
346             ↪ (), lblFinalPriceAmount.getText(), isTourOnly()), true);
347     else
348         MainApp.changePane(new PaymentPane(sale, pricelist, txfDiscount.getText(),
349             ↪ lblFinalPriceAmount.getText()), true);
350 }
351
352 private boolean containsRent() {
353     for (SalesLine salesLine : sale.getSalesLines()) {
354         if (salesLine.getProduct().getRentableStrategy() instanceof Rentable)
355             return true;
356     }
357     return false;
358 }
359
360 private boolean isTourOnly() {
361     for (SalesLine salesLine : sale.getSalesLines()) {
362         if (salesLine.getProduct() instanceof Tour && sale.getSalesLines().size() == 1)
363             return true;
364     }
365     return false;
366 }
367 }

```

Chapter 29: gui/salepane/SelectOngoingSale.java

```
1 package gui.salepane;
2
3 import controller.Controller;
4 import gui.MainApp;
5 import javafx.collections.FXCollections;
6 import javafx.collections.ObservableList;
7 import javafx.beans.property.SimpleStringProperty;
8 import javafx.geometry.Pos;
9 import javafx.scene.control.ChoiceBox;
10 import javafx.scene.control.TableColumn;
11 import javafx.scene.control.TableView;
12 import javafx.scene.control.cell.PropertyValueFactory;
13 import javafx.scene.layout.BorderPane;
14 import javafx.scene.layout.HBox;
15 import model.Pricelist;
16 import model.Rent;
17 import model.Sale;
18 import model.enums.SaleState;
19
20 import java.util.LinkedList;
21 import java.time.LocalDateTime;
22
23 public class SelectOngoingSale extends BorderPane {
24
25     private static SelectOngoingSale selectOngoingSale;
26
27     private ObservableList<Sale> obsOngoingSales;
28     private ChoiceBox<Pricelist> pricelistChoiceBox;
29
30     private TableView<Sale> ongoingSaleTableView;
31     private HBox hbContent;
32
33     private SelectOngoingSale() {
34         obsOngoingSales = FXCollections.observableList(new LinkedList<>());
35
36         hbContent = new HBox();
37         hbContent.setSpacing(20);
38         hbContent.setAlignment(Pos.CENTER);
39         this.setCenter(hbContent);
40         ongoingSaleTableView = getOngoingSaleTableView();
41         hbContent.getChildren().add(ongoingSaleTableView);
42
43         pricelistChoiceBox = new ChoiceBox<>();
44         pricelistChoiceBox.getItems().addAll(Controller.getInstance().getPricelists());
45         pricelistChoiceBox.getSelectionModel().selectFirst();
46         hbContent.getChildren().add(pricelistChoiceBox);
47         refreshOngoingSales();
48     }
```

```

49
50 public void refreshOngoingSales() {
51     obsOngoingSales.clear(); // because a sale can be completed (not just new sales
52                               ↪ appearing)
53     for (Sale sale : Controller.getInstance().getSales()) {
54         if (!sale.getState().equals(SaleState.COMPLETED))
55             obsOngoingSales.add(sale);
56     }
57     hbContent.getChildren().clear();
58     hbContent.getChildren().addAll(ongoingSaleTableView, pricelistChoiceBox);
59 }
60
61 private TableView<Sale> getOngoingSaleTableView() {
62     TableView<Sale> ongoingSaleTableView = new TableView<>();
63     ongoingSaleTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
64     ongoingSaleTableView.setEditable(false);
65     ongoingSaleTableView.setPrefHeight(500);
66     ongoingSaleTableView.setPrefWidth(500);
67
68     ongoingSaleTableView.getSelectionModel().selectedItemProperty().addListener((
69         ↪ observable, oldValue, newValue) -> goToSale(ongoingSaleTableView.
70         ↪ getSelectionModel().getSelectedItem()));
71
72     TableColumn<Sale, String> column1 = new TableColumn<>("Tidspunkt");
73     column1.setCellValueFactory(cellData -> {
74         LocalDateTime time = cellData.getValue().getTimestamp();
75         return new SimpleStringProperty(String.format("%s-%s-%s - %s:%s", time.
76             ↪ getDayOfMonth(),
77             time.getMonthValue(), time.getYear(), time.getHour(), time.getMinute()))
78         ↪ ;
79     });
80
81     TableColumn<Sale, String> column2 = new TableColumn<>("ID");
82     column2.setCellValueFactory(new PropertyValueFactory<>("id"));
83
84     TableColumn<Sale, String> column3 = new TableColumn<>("Kontakt navn");
85     column3.setCellValueFactory(cellData -> (cellData.getValue() instanceof Rent
86         ? new SimpleStringProperty(((Rent) cellData.getValue()).getContactName())
87         : new SimpleStringProperty("")));
88
89     TableColumn<Sale, String> column4 = new TableColumn<>("Kontakt information");
90     column4.setCellValueFactory(cellData -> (cellData.getValue() instanceof Rent
91         ? new SimpleStringProperty(((Rent) cellData.getValue()).
92             ↪ getContactInformation())
93         : new SimpleStringProperty("")));
94
95     ongoingSaleTableView.getColumns().add(column1);
96     ongoingSaleTableView.getColumns().add(column2);
97     ongoingSaleTableView.getColumns().add(column3);
98     ongoingSaleTableView.getColumns().add(column4);
99
100     ongoingSaleTableView.setItems(obsOngoingSales);
101     return ongoingSaleTableView;
102 }
103
104 public void goToSale(Sale sale) {

```

```
101     MainApp.changePane(new SalePane(sale, pricelistChoiceBox.getSelectionModel().
102         ↪ getSelectedItem()), true);
103 }
104 public static SelectOngoingSale getInstance() {
105     if (selectOngoingSale == null)
106         selectOngoingSale = new SelectOngoingSale();
107     return selectOngoingSale;
108 }
109
110 }
```

Chapter 30: gui/salepane/SelectPricelistPane.java

```
1 package gui.salepane;
2
3 import controller.Controller;
4 import gui.MainApp;
5 import javafx.geometry.Pos;
6 import javafx.scene.control.Button;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.Pagination;
9 import javafx.scene.layout.BorderPane;
10 import javafx.scene.layout.HBox;
11 import javafx.scene.layout.VBox;
12 import javafx.scene.text.Font;
13 import javafx.scene.text.TextAlignment;
14 import model.Pricelist;
15
16 import java.util.LinkedList;
17 import java.util.List;
18
19 public class SelectPricelistPane extends BorderPane {
20
21     private static SelectPricelistPane selectPricelistPane;
22
23     private static List<Button> buttonList;
24
25     private static final int BUTTONS_PER_PAGE = 5;
26     private static Pagination pagination;
27
28     private SelectPricelistPane() {
29         VBox vBox = new VBox();
30         vBox.setAlignment(Pos.CENTER);
31         vBox.setSpacing(10);
32         this.setCenter(vBox);
33
34         Label lblSelect = new Label("Vælg prisliste:");
35         lblSelect.setFont(new Font(24));
36         vBox.getChildren().add(lblSelect);
37
38         pagination = new Pagination();
39
40         buttonList = new LinkedList<>();
41
42         fillList();
43
44         pagination.setCurrentPageIndex(0);
45         pagination.setMaxPageIndicatorCount(3);
46
47         pagination.setPageFactory((pageIndex) -> {
48
```

```

49         HBox hBox = new HBox();
50         hBox.setAlignment(Pos.CENTER);
51         hBox.setSpacing(50);
52
53         for (int i = 0; i < BUTTONS_PER_PAGE; i++) {
54             int index = i + (pageIndex * BUTTONS_PER_PAGE);
55             if (index > buttonList.size() - 1) break;
56             Button button = buttonList.get(index);
57             button.setWrapText(true);
58             button.setTextAlignment(TextAlignment.CENTER);
59
60             VBox vBox1 = new VBox();
61             vBox1.setAlignment(Pos.CENTER);
62             vBox1.setSpacing(10);
63
64             vBox1.getChildren().add(button);
65             vBox1.getChildren().add(new Label(button.getText()));
66
67             hBox.getChildren().add(vBox1);
68         }
69         int i = ((buttonList.size() - 1) / BUTTONS_PER_PAGE) + 1;
70         pagination.setPageCount(i);
71
72         return hBox;
73     });
74
75     vbox.getChildren().add(pagination);
76 }
77
78 private void fillList() {
79     for (Pricelist pricelist : Controller.getInstance().getPricelists()) {
80         Button button = new Button(pricelist.getName());
81         button.setPrefHeight(75);
82         button.setPrefWidth(75);
83         button.setOnAction(event -> selectedPricelist(pricelist));
84         buttonList.add(button);
85     }
86 }
87
88 public void addPricelist(Pricelist pricelist) {
89     Button button = new Button(pricelist.getName());
90     button.setPrefHeight(75);
91     button.setPrefWidth(75);
92     button.setOnAction(event -> selectedPricelist(pricelist));
93     buttonList.add(button);
94 }
95
96 private static void selectedPricelist(Pricelist pricelist) {
97     MainApp.changePane(new SalePane(null, pricelist), true);
98 }
99
100 public static SelectPricelistPane getInstance() {
101     if (selectPricelistPane == null)
102         selectPricelistPane = new SelectPricelistPane();
103     return selectPricelistPane;
104 }
105 }

```

Chapter 31: gui/salepane/TableViewSalesLines.java

```
1 package gui.salepane;
2
3 import gui.images.LoadImage;
4 import javafx.beans.property.SimpleStringProperty;
5 import javafx.geometry.Pos;
6 import javafx.scene.control.*;
7 import javafx.scene.control.cell.PropertyValueFactory;
8 import javafx.scene.image.Image;
9 import javafx.scene.image.ImageView;
10 import model.Sale;
11 import model.SalesLine;
12
13 public class TableViewSalesLines extends TableView<SalesLine> {
14
15     public TableViewSalesLines(Sale sale) {
16         setPrefHeight(450);
17         setPrefWidth(400);
18         setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
19         setEditable(false);
20         setPlaceholder(new Label("Ingen produkter"));
21
22         TableColumn<SalesLine, String> column1 = new TableColumn<>("Navn");
23         column1.setCellValueFactory(new PropertyValueFactory<>("product"));
24         column1.setStyle("-fx-alignment: CENTER-LEFT");
25
26         TableColumn<SalesLine, Integer> column2 = new TableColumn<>("Antal");
27         column2.setCellValueFactory(new PropertyValueFactory<>("quantity"));
28         column2.setStyle("-fx-alignment: CENTER");
29
30         TableColumn<SalesLine, Double> column3 = new TableColumn<>("Samlet Pris");
31
32         TableColumn<SalesLine, String> column4 = new TableColumn<>("Udlejning");
33         column4.setCellValueFactory(cellData -> cellData.getValue().getTotalLineDeposit() >=
34             ↪ 0
35             ? new SimpleStringProperty(String.valueOf(cellData.getValue().
36                 ↪ getTotalLineDeposit()))
37             : new SimpleStringProperty(""));
38         column4.setStyle("-fx-alignment: CENTER-RIGHT");
39
40         TableColumn<SalesLine, Double> column5 = new TableColumn<>("Salg");
41         column5.setCellValueFactory(new PropertyValueFactory<>("totalLinePrice"));
42         column5.setStyle("-fx-alignment: CENTER-RIGHT");
43
44         TableColumn<SalesLine, String> column6 = new TableColumn<>("");
45         column6.setCellFactory(param -> new TableCell<SalesLine, String>() {
46             final Button btnRemove = new Button("X");
47
48             @Override
```

```
47     public void updateItem(String item, boolean empty) {
48         setGraphic(null);
49         if (!empty) {
50             Image image = LoadImage.loadImage("res/images/salespane/cross.png");
51             if (image != null) {
52                 ImageView imageView = new ImageView(image);
53                 imageView.setFitWidth(13);
54                 imageView.setFitHeight(13);
55                 btnRemove.setGraphic(imageView);
56                 btnRemove.setText(""); // Remove text and use button instead
57             }
58             btnRemove.setDisable(true);
59             setAlignment(Pos.CENTER);
60             setGraphic(btnRemove);
61         }
62     }
63 });
64 column6.setMinWidth(40);
65 column6.setMaxWidth(40);
66
67 getColumns().add(column1);
68 getColumns().add(column2);
69 getColumns().add(column3);
70 column3.getColumns().add(column4);
71 column3.getColumns().add(column5);
72 getColumns().add(column6);
73
74 getItems().addAll(sale.getSalesLines());
75 }
76
77 }
```


Chapter 32: gui/salepane/TableViewSalesLines.java

```
1 package gui.salepane;
2
3 import gui.images.LoadImage;
4 import javafx.beans.property.SimpleStringProperty;
5 import javafx.geometry.Pos;
6 import javafx.scene.control.*;
7 import javafx.scene.control.cell.PropertyValueFactory;
8 import javafx.scene.image.Image;
9 import javafx.scene.image.ImageView;
10 import model.Sale;
11 import model.SalesLine;
12
13 public class TableViewSalesLines extends TableView<SalesLine> {
14
15     public TableViewSalesLines(Sale sale) {
16         setPrefHeight(450);
17         setPrefWidth(400);
18         setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
19         setEditable(false);
20         setPlaceholder(new Label("Ingen produkter"));
21
22         TableColumn<SalesLine, String> column1 = new TableColumn<>("Navn");
23         column1.setCellValueFactory(new PropertyValueFactory<>("product"));
24         column1.setStyle("-fx-alignment: CENTER-LEFT");
25
26         TableColumn<SalesLine, Integer> column2 = new TableColumn<>("Antal");
27         column2.setCellValueFactory(new PropertyValueFactory<>("quantity"));
28         column2.setStyle("-fx-alignment: CENTER");
29
30         TableColumn<SalesLine, Double> column3 = new TableColumn<>("Samlet Pris");
31
32         TableColumn<SalesLine, String> column4 = new TableColumn<>("Udlejning");
33         column4.setCellValueFactory(cellData -> cellData.getValue().getTotalLineDeposit() >=
34             ↪ 0
35             ? new SimpleStringProperty(String.valueOf(cellData.getValue().
36                 ↪ getTotalLineDeposit()))
37             : new SimpleStringProperty(""));
38         column4.setStyle("-fx-alignment: CENTER-RIGHT");
39
40         TableColumn<SalesLine, Double> column5 = new TableColumn<>("Salg");
41         column5.setCellValueFactory(new PropertyValueFactory<>("totalLinePrice"));
42         column5.setStyle("-fx-alignment: CENTER-RIGHT");
43
44         TableColumn<SalesLine, String> column6 = new TableColumn<>("");
45         column6.setCellFactory(param -> new TableCell<SalesLine, String>() {
46             final Button btnRemove = new Button("X");
47
48             @Override
```

```
47     public void updateItem(String item, boolean empty) {
48         setGraphic(null);
49         if (!empty) {
50             Image image = LoadImage.loadImage("res/images/salespane/cross.png");
51             if (image != null) {
52                 ImageView imageView = new ImageView(image);
53                 imageView.setFitWidth(13);
54                 imageView.setFitHeight(13);
55                 btnRemove.setGraphic(imageView);
56                 btnRemove.setText(""); // Remove text and use button instead
57             }
58             btnRemove.setDisable(true);
59             setAlignment(Pos.CENTER);
60             setGraphic(btnRemove);
61         }
62     }
63 });
64 column6.setMinWidth(40);
65 column6.setMaxWidth(40);
66
67 getColumns().add(column1);
68 getColumns().add(column2);
69 getColumns().add(column3);
70 column3.getColumns().add(column4);
71 column3.getColumns().add(column5);
72 getColumns().add(column6);
73
74 getItems().addAll(sale.getSalesLines());
75 }
76
77 }
```

Chapter 33: gui/StatisticPane.java

```
1 package gui;
2
3 import gui.statistics.ExportPane;
4 import gui.statistics.FilterPane;
5 import gui.statistics.GraphsPane;
6 import gui.statistics.ReportTextPane;
7 import javafx.geometry.Insets;
8 import javafx.geometry.Pos;
9 import javafx.scene.control.Label;
10 import javafx.scene.layout.BorderPane;
11 import javafx.scene.layout.Pane;
12
13 import java.util.LinkedHashMap;
14 import java.util.Map;
15
16 public class StatisticPane extends BorderPane implements LeftSideBarInterface {
17
18     private static StatisticPane statisticPane = getInstance();
19
20     private Label lblTitle;
21     private LeftSideBar leftSideBar;
22
23     private StatisticPane() {
24         this.setPadding(new Insets(20));
25
26         // Initialize
27         lblTitle = new Label();
28         lblTitle.setPrefWidth(700);
29         lblTitle.getStyleClass().add("subtitle");
30         lblTitle.setPrefWidth(700);
31         this.setTop(lblTitle);
32
33         Map<String, Pane> navigationMap = new LinkedHashMap<>();
34         navigationMap.put("Filtrer", FilterPane.getInstance());
35         navigationMap.put("Grafer", GraphsPane.getInstance());
36         navigationMap.put("Tekstrapport", ReportTextPane.getInstance());
37         navigationMap.put("EksportÃr rapport", ExportPane.getInstance());
38         leftSideBar = new LeftSideBar(navigationMap, this);
39         setLeft(leftSideBar);
40
41         changeSelected((String) navigationMap.keySet().toArray()[0]); // selects first entry
42     }
43
44     public void setTitle(String title) {
45         lblTitle.setText(title);
46     }
47
48     @Override
```

```
49     public void changeSelected(String nameOfButton) {
50         Label title = new Label(nameOfButton);
51         title.getStyleClass().add("title");
52         setAlignment(title, Pos.CENTER);
53         setTop(title);
54         setCenter(leftSideBar.getPane(nameOfButton));
55     }
56
57     public static StatisticPane getInstance() {
58         if (statisticPane == null)
59             statisticPane = new StatisticPane();
60         return statisticPane;
61     }
62
63
64 }
```

Chapter 34: gui/statistics/ExportPane.java

```
1 package gui.statistics;
2
3 import javafx.geometry.Pos;
4 import javafx.scene.control.Alert;
5 import javafx.scene.control.Button;
6 import javafx.scene.layout.GridPane;
7 import javafx.scene.layout.HBox;
8 import javafx.stage.DirectoryChooser;
9 import model.Sale;
10 import model.SalesLine;
11
12 import java.io.File;
13 import java.io.FileWriter;
14 import java.io.IOException;
15
16 public class ExportPane extends GridPane {
17     private static ExportPane exportPane;
18
19     private ExportPane() {
20         this.getStyleClass().add("gridpane");
21         this.setAlignment(Pos.CENTER);
22
23         // Buttons
24         Button btnSaveTxt = new Button("Gem .txt");
25         Button btnSaveCSV = new Button("Gem .csv");
26         btnSaveTxt.getStyleClass().add("important");
27         btnSaveCSV.getStyleClass().add("important");
28         HBox hbButtons = new HBox(btnSaveTxt, btnSaveCSV);
29         hbButtons.setSpacing(100);
30
31         // Listeners
32         btnSaveCSV.setOnAction(e -> exportCsv());
33         btnSaveTxt.setOnAction(e -> exportTxt());
34
35         // GUI
36         this.add(hbButtons, 0, 0);
37     }
38
39     // Export
40     private void exportTxt() {
41         File directory = new DirectoryChooser().showDialog(null);
42         if (directory != null) {
43             String filename = "salgslinjer for perioden " + FilterPane.getInstance().
44                 ↪ getDateTimeFrom().toLocalDate() + "-" + FilterPane.getInstance().
45                 ↪ getDateTimeTo().toLocalDate()
46                 + ".txt";
47             File fileOutput = new File(directory + File.separator + filename);
48             writeFile(fileOutput, ReportTextPane.getInstance().getText());
49         }
50     }
51 }
```

```
47     }
48 }
49
50
51 private void exportCsv() {
52     File directory = new DirectoryChooser().showDialog(null);
53     if (directory != null) {
54         String filename = "salgslinjer for perioden " + FilterPane.getInstance().
55             ↪ getDateTimeFrom().toLocalDate() + "-" + FilterPane.getInstance().
56             ↪ getDateTimeTo().toLocalDate()
57             + ".csv";
58         File fileOutput = new File(directory + File.separator + filename);
59         StringBuilder sbOutputContent = new StringBuilder();
60         sbOutputContent.append("dato,salgID,produkt,antal,pris i kr.");
61         for (Sale sale : FilterPane.getInstance().getSalesFromFilter()) {
62             for (SalesLine salesLine : sale.getSalesLines()) {
63                 sbOutputContent.append("\n");
64                 sbOutputContent.append(String.format("\'%s\'", sale.
65                     ↪ getTimestamp(), sale.getId(),
66                     salesLine.getProduct().getName(), salesLine.getQuantity(),
67                     ↪ salesLine.getTotalLinePrice()));
68             }
69         }
70         writeFile(fileOutput, sbOutputContent.toString());
71     }
72 }
73
74 private void writeFile(File file, String content) {
75     try (FileWriter fileWriter = new FileWriter(file)) {
76         fileWriter.write(content);
77     } catch (IOException e) {
78         Alert alert = new Alert(Alert.AlertType.ERROR);
79         alert.setContentText("Fejl: kan ikke gemme filen.");
80         alert.showAndWait();
81     }
82 }
83
84 public static ExportPane getInstance() {
85     if (exportPane == null)
86         exportPane = new ExportPane();
87     return exportPane;
88 }
```

Chapter 35: gui/statistics/FilterPane.java

```
1 package gui.statistics;
2
3 import controller.Controller;
4 import gui.StatisticPane;
5 import gui.helpers.DateIntervalPicker;
6 import javafx.collections.ListChangeListener;
7 import javafx.scene.control.Button;
8 import javafx.scene.control.Label;
9 import javafx.scene.control.ListView;
10 import javafx.scene.control.SelectionMode;
11 import javafx.scene.layout.GridPane;
12 import javafx.scene.layout.VBox;
13 import model.Pricelist;
14 import model.Sale;
15 import model.enums.SaleState;
16 import model.product.Product;
17
18 import java.time.LocalDateTime;
19 import java.util.List;
20 import java.util.Map;
21 import java.util.Set;
22 import java.util.stream.Collectors;
23
24 public class FilterPane extends GridPane {
25     private static FilterPane filterPane;
26
27     private DateIntervalPicker datePicker;
28     private ListView<Pricelist> lwPricelists;
29     private ListView<Product> lwProducts;
30     private PieChartExtended pcProductsUnwrapBundles;
31     private Label lblProductQuantityOverview;
32
33     private FilterPane() {
34         this.getStyleClass().add("gridpane");
35
36         datePicker = new DateIntervalPicker();
37         // listviews
38         lwPricelists = new ListView<>();
39         lwPricelists.getItems().setAll(Controller.getInstance().getPricelists());
40         VBox vbPricelist = new VBox(new Label("Prislister"), lwPricelists);
41         vbPricelist.setMaxHeight(200);
42         lwProducts = new ListView<>();
43         lwProducts.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
44         VBox vbProducts = new VBox(new Label("Produkter (fra valgte prislister)"),
45             ↪ lwProducts);
46         vbProducts.setMaxHeight(200);
47         Button btnApply = new Button("Anvend filter");
48         Button btnSelectAll = new Button("Vælg alt");
```

```

48 VBox vbFilterButtons = new VBox(btnApply, btnSelectAll);
49 // btnApply.getStyleClass().add("important");
50 pcProductsUnwrapBundles = new PieChartExtended("Alle salg for perioden");
51 lblProductQuantityOverview = new Label();
52
53
54 // Listeners
55 lwPricelists.getSelectionModel().getSelectedItem()
56     .addListener((ListChangeListener<Pricelist>) c -> updateProductList());
57 btnSelectAll.setOnAction(e -> {
58     lwProducts.getSelectionModel().selectAll();
59     updateReport();
60 });
61 btnApply.setOnAction(e -> updateReport());
62
63 this.add(datePicker, 0, 1);
64 this.add(vbPricelist, 0, 2);
65 this.add(vbProducts, 1, 2);
66 this.add(vbFilterButtons, 2, 2);
67 this.add(pcProductsUnwrapBundles, 0, 3);
68 this.add(lblProductQuantityOverview, 1, 3);
69 }
70
71 /**
72  * Updates the list of products to reflect the Pricelist selection
73  * contains instead of having it inside lwProducts
74  */
75 private void updateProductList() {
76     lwProducts.getItems().clear();
77     lwPricelists.getSelectionModel().getSelectedItem()
78         .forEach(pricelist -> pricelist.getProductsWithPrice()
79             .forEach((product, price) -> {
80                 if (!lwProducts.getItems().contains(product))
81                     lwProducts.getItems().add(product);
82             }));
83     lwProducts.getSelectionModel().selectAll();
84 }
85
86 public List<Product> getSelectedProducts() {
87     return lwProducts.getSelectionModel().getSelectedItem();
88 }
89
90 public LocalDateTime getDateFrom() {
91     return datePicker.getDateFrom().atStartOfDay();
92 }
93
94 public LocalDateTime getDateTo() {
95     return datePicker.getDateTo().atStartOfDay();
96 }
97
98 /**
99  * When any changes to products or date interval is made, this method is called
100  * and updates all relevant content.
101  * <p>
102  * - method is private because it can be updated by change of filter through other
103  *   ↪ panes
104  */
105 public void updateReport() {

```



```

105     StatisticPane.getInstance().setTitle(
106         String.format("Salgsoplysning fra: [%s - %s]", datePicker.getDateFrom(),
107             ↳ datePicker.getDateTo()));
108     Set<Sale> salesInDateRange = getSalesFromFilter();
109     Map<Product, Integer> productsWithQuantity = GraphsPane.getInstance().
110         ↳ getProductsWithQuantity(
111             salesInDateRange.stream().flatMap(sale -> sale.getSalesLines().stream()).
112             ↳ collect(Collectors.toSet()));
113     Map<Product, Integer> bundleProductsWithQuantity = GraphsPane.getInstance().
114         ↳ unwrapBundleFromProductsWithQuantity(productsWithQuantity);
115     Map<Product, Integer> unwrappedBundleWithProductWithQuantity = GraphsPane.getInstance
116         ↳ ().mergeAndRemoveBundleFromProductWithQuantityMaps(
117             bundleProductsWithQuantity, productsWithQuantity);
118
119     GraphsPane.getInstance().setPcProducts(productsWithQuantity);
120     GraphsPane.getInstance().setPcBundles(bundleProductsWithQuantity);
121     GraphsPane.getInstance().setPcProductsUnwrapBundles(
122         ↳ unwrappedBundleWithProductWithQuantity);
123     pcProductsUnwrapBundles.setData(GraphsPane.getInstance().
124         ↳ getPieSlicesFromProductsWithQuantity(unwrappedBundleWithProductWithQuantity));
125     GraphsPane.getInstance().setPcPaymentMethods(salesInDateRange);
126     updateLblProductQuantityOverview(unwrappedBundleWithProductWithQuantity);
127
128     ReportTextPane.getInstance().setText(salesInDateRange,
129         ↳ unwrappedBundleWithProductWithQuantity);
130 }
131
132 private void updateLblProductQuantityOverview(Map<Product, Integer> productQuantityMap)
133     ↳ {
134     StringBuilder sbOverview = new StringBuilder();
135     for (Product product : productQuantityMap.keySet()) {
136         sbOverview.append(product.getName() + ": " + productQuantityMap.get(product) + "
137             ↳ \n");
138     }
139     lblProductQuantityOverview.setText(sbOverview.toString());
140 }
141
142 public Set<Sale> getSalesFromFilter() {
143     List<Product> productList = FilterPane.getInstance().getSelectedProducts();
144     LocalDateTime dateTimeFrom = FilterPane.getInstance().getDateTimeFrom();
145     LocalDateTime dateTimeTo = FilterPane.getInstance().getDateTimeTo();
146
147     return Controller.getInstance().getSales().stream()
148         .filter(sale -> (sale.getTimestamp().isAfter(dateTimeFrom) && sale.
149             ↳ getTimestamp().isBefore(dateTimeTo)))
150         .filter(sale -> sale.getSaleState().equals(SaleState.COMPLETED))
151         .filter(sale -> sale.getSalesLines().stream()
152             ↳ .anyMatch(salesLine -> productList.contains(salesLine.getProduct())))
153             ↳ .collect(Collectors.toSet());
154 }
155
156 public static FilterPane getInstance() {
157     if (filterPane == null)
158         filterPane = new FilterPane();
159     return filterPane;
160 }

```

151 | }

Chapter 36: gui/statistics/GraphsPane.java

```
1 package gui.statistics;
2
3 import javafx.scene.chart.PieChart;
4 import javafx.scene.layout.GridPane;
5 import model.Sale;
6 import model.SalesLine;
7 import model.enums.PaymentMethod;
8 import model.product.Bundle;
9 import model.product.Product;
10
11 import java.util.HashMap;
12 import java.util.List;
13 import java.util.Map;
14 import java.util.Set;
15 import java.util.stream.Collectors;
16
17 public class GraphsPane extends GridPane {
18     private static GraphsPane graphsPane;
19
20
21     private PieChartExtended pcProducts; // products where bundles show as slice
22     private PieChartExtended pcBundles; // products sold by bundle in selection
23     private PieChartExtended pcProductsUnwrapBundles; // products where bundles are "
24         ↪ unwrapped"
25     private PieChartExtended pcPaymentMethods;
26
27     private GraphsPane() {
28         this.getStyleClass().add("gridpane");
29
30         // piecharts
31         pcProducts = new PieChartExtended("Produkter (bundt vises som sammenpakning)");
32         pcBundles = new PieChartExtended("Produkter solgt gennem bundter");
33         pcProductsUnwrapBundles = new PieChartExtended("Produkter (bundters produkter inkl.)
34             ↪ ");
35         pcPaymentMethods = new PieChartExtended("Betalingsmetoder");
36
37         this.add(pcProducts, 0, 0);
38         this.add(pcBundles, 1, 0);
39         this.add(pcProductsUnwrapBundles, 0, 1);
40         this.add(pcPaymentMethods, 1, 1);
41     }
42
43     public Map<Product, Integer> unwrapBundleFromProductsWithQuantity(Map<Product, Integer>
44         ↪ productsWithQuantity) {
45         Map<Product, Integer> unwrappedProductsWithQuantity = new HashMap<>();
46         productsWithQuantity.keySet().stream().filter(product -> product instanceof Bundle)
47             .forEach(bundle -> ((Bundle) bundle).getProductsWithQuantity())
```

```

45         .forEach((product, quantity) -> unwrappedProductsWithQuantity.put(
46             ↪ product,
47             unwrappedProductsWithQuantity.getDefault(product, 0)
48             + quantity * productsWithQuantity.get(bundle)));
49     return unwrappedProductsWithQuantity;
50 }
51
52 public Map<Product, Integer> getProductsWithQuantity(Set<SalesLine> salesLines) {
53     HashMap<Product, Integer> products = new HashMap<>();
54     for (SalesLine salesLine : salesLines) {
55         products.put(salesLine.getProduct(), products.getDefault(salesLine.getProduct()
56             ↪ (), 0) + 1);
57     }
58     return products;
59 }
60
61 public Map<Product, Integer> mergeAndRemoveBundleFromProductWithQuantityMaps(
62     Map<Product, Integer> productWithQuantityAndBundle, Map<Product, Integer>
63     ↪ productWithQuantity) {
64     // Remove all instances of bundle
65     productWithQuantityAndBundle.entrySet().removeIf(productQuantity -> productQuantity.
66     ↪ getKey() instanceof Bundle);
67
68     for (Product product : productWithQuantityAndBundle.keySet()) {
69         productWithQuantity.put(product,
70             productWithQuantity.getDefault(product, 0) +
71             ↪ productWithQuantityAndBundle.get(product));
72     }
73
74     return productWithQuantity;
75 }
76
77 public List<PieChart.Data> getPieSlicesFromProductsWithQuantity(Map<Product, Integer>
78     ↪ productQuantity) {
79     return productQuantity.keySet().stream()
80         .map(product -> new PieChart.Data(product.getName(), productQuantity.get(
81             ↪ product)))
82         .collect(Collectors.toList());
83 }
84
85 public void setPcProducts(Map<Product, Integer> productsWithQuantity) {
86     pcProducts.setData(getPieSlicesFromProductsWithQuantity(productsWithQuantity));
87 }
88
89 public void setPcBundles(Map<Product, Integer> productsWithQuantity) {
90     pcBundles.setData(getPieSlicesFromProductsWithQuantity(productsWithQuantity));
91 }
92
93 public void setPcProductsUnwrapBundles(Map<Product, Integer> productsWithQuantity) {
94     pcProductsUnwrapBundles.setData(getPieSlicesFromProductsWithQuantity(
95     ↪ productsWithQuantity));
96 }
97
98 public void setPcPaymentMethods(Set<Sale> sales) {
99     pcPaymentMethods.setData(getPieSlicesFromPaymentMethod(getPaymentAmountMap(sales)));
100 }

```

```
95
96 public Map<PaymentMethod, Double> getPaymentAmountMap(Set<Sale> sales) {
97     HashMap<PaymentMethod, Double> paymentAmountMap = new HashMap<>();
98     for (Sale sale : sales) {
99         Map<PaymentMethod, Double> transfers = sale.getPayment().getTransfers();
100         for (PaymentMethod paymentMethod : transfers.keySet()) {
101             paymentAmountMap.put(paymentMethod, paymentAmountMap.getOrDefault(
102                 ↪ paymentMethod, 0d) + transfers.get(paymentMethod));
103         }
104     }
105     return paymentAmountMap;
106 }
107
108 public List<PieChart.Data> getPieSlicesFromPaymentMethod(Map<PaymentMethod, Double>
109     ↪ paymentMethodAmountMap) {
110     return paymentMethodAmountMap.keySet().stream()
111         .map(paymentMethod -> new PieChart.Data(paymentMethod.name(),
112             ↪ paymentMethodAmountMap.get(paymentMethod)))
113         .collect(Collectors.toList());
114 }
115
116 public static GraphsPane getInstance() {
117     if (graphsPane == null)
118         graphsPane = new GraphsPane();
119     return graphsPane;
120 }
```

Chapter 37: gui/statistics/PieChartExtended.java

```
1 package gui.statistics;
2
3 import javafx.scene.chart.PieChart;
4 import javafx.scene.control.Label;
5 import javafx.scene.input.MouseEvent;
6 import javafx.scene.paint.Color;
7
8 import java.util.List;
9
10 public class PieChartExtended extends PieChart {
11
12     public PieChartExtended(String title) {
13         setTitle(title);
14     }
15
16     public void setData(List<PieChart.Data> dataPoints) {
17         this.getData().setAll(dataPoints);
18
19         final Label caption = new Label("");
20         caption.setTextFill(Color.DARKORANGE);
21         caption.setStyle("-fx-font: 24 arial;");
22         this.getChildren().add(caption);
23         for (final PieChart.Data data : this.getData()) {
24             data.getNode().addEventHandler(MouseEvent.MOUSE_ENTERED, e -> {
25                 // Position not known before Node has been initialized
26                 caption.setTranslateX(e.getSceneX() - getLayoutX());
27                 caption.setTranslateY(e.getSceneY());
28                 caption.setText(String.valueOf(data.getPieValue()));
29             });
30
31             data.getNode().addEventHandler(MouseEvent.MOUSE_EXITED, e -> caption.setText(""))
32                 ↪ );
33         }
34     }
35 }
```

Chapter 38: gui/statistics/ReportTextPane.java

```
1 package gui.statistics;
2
3 import javafx.geometry.Pos;
4 import javafx.scene.control.TextArea;
5 import javafx.scene.layout.GridPane;
6 import model.Sale;
7 import model.SalesLine;
8 import model.enums.PaymentMethod;
9 import model.product.Product;
10
11 import java.util.Map;
12 import java.util.Set;
13
14 public class ReportTextPane extends GridPane {
15     private static ReportTextPane reportTextPane;
16
17     private TextArea txaHistory;
18
19     private ReportTextPane() {
20         this.getStyleClass().add("gridpane");
21         this.setAlignment(Pos.CENTER);
22
23         txaHistory = new TextArea();
24         txaHistory.setMinWidth(500);
25         txaHistory.setMinHeight(600);
26         this.add(txaHistory, 0, 0);
27     }
28
29     private String writeSalesReport(Set<Sale> sales, Map<Product, Integer>
        ↪ productsWithQuantity) {
30         Map<PaymentMethod, Double> paymentAmountMap = GraphsPane.getInstance().
        ↪ getPaymentAmountMap(sales);
31         StringBuilder sbLog = new StringBuilder();
32         sbLog.append(String.format("Sales for perioden: %s-%s\n", FilterPane.getInstance().
        ↪ getDateFrom().toLocalDate(), FilterPane.getInstance().getDateTo().
        ↪ toLocalDate()));
33         sbLog.append(String.format("* antal salg: %s\n", sales.size()));
34
35         sbLog.append("\n----- Produkt oversigt: \n");
36         for (PaymentMethod paymentMethod : paymentAmountMap.keySet()) {
37             sbLog.append("- " + paymentMethod.name().toLowerCase() + ": " + paymentAmountMap
        ↪ .get(paymentMethod) + "kr\n");
38         }
39
40         sbLog.append("\n----- Salgsmetoder: \n");
41         sbLog.append(String.format("%3s) %4s: %30s\n", "#", "produkt", "antal"));
42         for (int i = 0; i < productsWithQuantity.size(); i++) {
43             Product product = (Product) productsWithQuantity.keySet().toArray()[i];
```

```

44         sbLog.append(String.format("%3s) %4s: %30s%n", i + 1, product,
45             ↪ productsWithQuantity.get(product)));
46     }
47     StringBuilder sbSales = new StringBuilder();
48     sbSales.append(String.format("Alle salg: %n"));
49     for (Sale sale : sales) {
50         sbSales.append(String.format("-----%n"));
51         sbSales.append(String.format("Salg: d. %s %s handlet for %skr%n", sale.
52             ↪ getTimestamp().toLocalDate(),
53             sale.getTimestamp().toLocalTime(), sale.totalSalePrice()));
54         for (SalesLine salesLine : sale.getSalesLines()) {
55             sbSales.append(String.format("  - %s x %s : %skr%n", salesLine.getProduct().
56             ↪ getName(),
57             salesLine.getQuantity(), salesLine.getTotalLinePrice()));
58         }
59         sbSales.append(String.format("%n"));
60     }
61     return sbLog.toString() + "\n" + sbSales.toString();
62 }
63
64 public void setText(Set<Sale> salesInDateRange, Map<Product, Integer>
65     ↪ unwrappedBundleWithProductWithQuantity) {
66     txaHistory.setText(writeSalesReport(salesInDateRange,
67     ↪ unwrappedBundleWithProductWithQuantity));
68 }
69
70 public String getText() {
71     return txaHistory.getText();
72 }
73
74 public static ReportTextPane getInstance() {
75     if (reportTextPane == null)
76         reportTextPane = new ReportTextPane();
77     return reportTextPane;
78 }
79 }

```


Chapter 39: model/enums/PaymentMethod.java

```
1 package model.enums;
2
3 public enum PaymentMethod {
4
5     MOBILEPAY("MobilePay"),
6     VOUCHER("Klippekort"),
7     PAYMENTCARD("Betalingkort"),
8     CASH("Kontant");
9
10    PaymentMethod(String text) {
11        this.text = text;
12    }
13
14    private final String text;
15
16    @Override
17    public String toString() {
18        return text;
19    }
20 }
```

Chapter 40: model/enums/SaleSate.java

```
1 package model.enums;  
2  
3 public enum SaleState {  
4  
5     INITIATED,  
6     DELAYED,  
7     COMPLETED  
8  
9 }
```

Chapter 41: model/enums/Unit.java

```
1 package model.enums;  
2  
3 public enum Unit {  
4  
5     kg,  
6     l,  
7     cl  
8  
9 }
```

Chapter 42: model/Payment.java

```
1 package model;
2
3 import model.enums.PaymentMethod;
4
5 import java.io.Serializable;
6 import java.util.HashMap;
7 import java.util.Map;
8
9 /**
10  * Creates a payment which tracks the state (finished, initiated etc)
11  * and the amount currently assigned to a sale.
12  * A payment consist of multiple payments (transfers) which continuously can be added.
13  * A transfer has an amount and a source (paymentcard, cash etc).
14  *
15  * @author: Mathias, Kasper, Alexander
16  */
17 public class Payment implements Serializable {
18
19     private Map<PaymentMethod, Double> transfers = new HashMap<>();
20
21     /**
22      * Payment is empty until a transfer is added.
23      * thus empty constructor
24      */
25     public Payment() {
26     }
27
28     /**
29      * Adds a transfer to the payment of a sale.
30      * If adding an already added paymentMethod, then the previous and new amount will
31      *     ↪ be added.
32      * Therefore, one amount is stored per paymentMethod.
33      *
34      * @param paymentMethod, method of the payment.
35      * @param amount, the amount of money spend in the transfer.
36      */
37     public void addTransfer(PaymentMethod paymentMethod, double amount) {
38         transfers.put(paymentMethod, transfers.getDefault(paymentMethod, 0.0) +
39             ↪ amount);
40     }
41
42     /**
43      * calculate the total amount received for the payment.
44      *
45      * @return the total amount for the payment.
46      */
47     public double calcTotal() {
48         double total = 0;
49     }
50 }
```

```
47         for (double amount : transfers.values()) {
48             total += amount;
49         }
50         return total;
51     }
52
53     /**
54      * Get the transfers as a map. So that each key represent a
55      * paymentMethod and corresponding value is amount transferred through that method.
56      *
57      * @return the list of paymentMethods paired with the associated amount
58      */
59     public Map<PaymentMethod, Double> getTransfers() {
60         return transfers;
61     }
62 }
```

Chapter 43: model/Pricelist.java

```
1 package model;
2
3 import model.product.Product;
4
5 import java.io.Serializable;
6 import java.util.HashMap;
7 import java.util.Map;
8
9 public class Pricelist implements Serializable {
10
11     private String name;
12     private Map<Product, Double> productsWithPrice = new HashMap<>();
13
14     public Pricelist(String name) {
15         this.name = name;
16     }
17
18     /**
19      * Adds a product with a price to this pricelist, if they aren't connected
20      * Has a 0..* relationship with Product
21      *
22      * @param product, the choosen product
23      * @param price, the price of the product on this pricelist
24      */
25     public void addProductWithPrice(Product product, double price) {
26         if (!productsWithPrice.containsKey(product)) {
27             productsWithPrice.put(product, price);
28             product.addPricelist(this, price);
29         }
30     }
31
32     /**
33      * Remove a product with price from this pricelist.
34      * Has a 0..* relationship with Product
35      *
36      * @param product, the product you want to remove
37      */
38     public void removeProductWithPrice(Product product) {
39         if (productsWithPrice.containsKey(product)) {
40             productsWithPrice.remove(product);
41             product.removePricelist(this);
42         }
43     }
44
45     /**
46      * update a price on a product in this pricelist
47      *
48      * @param product, the product you want to update
```

```
49  * @param price,    the updated price
50  * @throws IllegalArgumentException if product is not in pricelist
51  */
52  public void updateProductPrice(Product product, double price) {
53      if (productsWithPrice.containsKey(product)) {
54          productsWithPrice.put(product, price);
55      }
56  }
57
58  /**
59   * gets the price of a chosen product in this pricelist
60   *
61   * @param product, the product you want the price from
62   * @return the price, from this pricelist, of the chosen product
63   */
64  public double getPriceOfProduct(Product product) {
65      try {
66          return productsWithPrice.get(product);
67      } catch (Exception e) {
68          throw new IllegalArgumentException("Produkt findes ikke i listen");
69      }
70  }
71
72  /**
73   * Returns the name of the pricelist
74   *
75   * @return name of the pricelist
76   */
77  public String getName() {
78      return name;
79  }
80
81  /**
82   * Get the product and its paired price.
83   *
84   * @return map of product as key and price as value
85   */
86  public Map<Product, Double> getProductsWithPrice() {
87      return productsWithPrice;
88  }
89
90  @Override
91  public String toString() {
92      return name;
93  }
94  }
```

Chapter 44: model/product/Bundle.java

```
1 package model.product;
2
3 import model.ProductGroup;
4
5 import java.util.HashMap;
6 import java.util.Map;
7
8 public class Bundle extends Product {
9
10     // Link to the Product class (--> 0..*)
11     private Map<Product, Integer> productsWithQuantity = new HashMap<>();
12
13     /**
14      * Creates a bundle which is almost like a product.
15      * However it can hold a list of other products (and a quantity mapped to each).
16      * Can't hold another bundle!
17      *
18      * @param name of the bundle
19      * @param productGroup that the bundle belongs to
20      * @param description of the bundle
21      */
22     public Bundle(String name, ProductGroup productGroup, String description) {
23         super(name, productGroup, description);
24     }
25
26     /**
27      * The content of the bundle key is product and value is quantity
28      * @return product as key associated with quantity as value
29      */
30     public HashMap<Product, Integer> getProductsWithQuantity() {
31         return new HashMap<>(productsWithQuantity);
32     }
33
34     /**
35      * adds a product to this bundle
36      *
37      * @param product, the product you want added to the bundle
38      * @pre product is not of Bundle type
39      */
40     public void addProduct(Product product) {
41         productsWithQuantity.put(product, productsWithQuantity.getOrDefault(product, 0) + 1)
42         ↪ ;
43     }
44
45     /**
46      * removes a product from this bundle
47      *
48      * @param product, the product you want to remove from the bundle
```



```
48  */
49  public void removeProduct(Product product) {
50      if (productsWithQuantity.get(product) > 0)
51          productsWithQuantity.put(product, productsWithQuantity.get(product) - 1);
52  }
53
54  @Override
55  public String toString() {
56      StringBuilder stringBuilder = new StringBuilder();
57      stringBuilder.append(getName() + ": ");
58      for (Product product : productsWithQuantity.keySet()) {
59          stringBuilder.append(product.getName() + "(" + productsWithQuantity.get(product)
60              ↪      + ")");
61      }
62      // strips last 2 characters to avoid trailing ", "
63      return stringBuilder.toString().substring(0, stringBuilder.toString().length() - 2);
64  }
```

Chapter 45: model/product/Container.java

```
1 package model.product;
2
3 import model.ProductGroup;
4 import model.enums.Unit;
5
6 public class Container extends Product {
7
8     private double size;
9     private Unit unit;
10
11     /**
12      * Creates a product that is a container with all parameters
13      * used to set attributes.
14      * A container is different from a product because it can have a size for a given unit.
15      *     eg: 1 litre, 2 kg
16      * @param name of the container
17      * @param productGroup that the container belongs to
18      * @param description of the container
19      * @param size of the unit
20      * @param unit of the size
21      */
22     public Container(String name, ProductGroup productGroup, String description, double size
23         ↪ , Unit unit) {
24         super(name, productGroup, description);
25         this.size = size;
26         this.unit = unit;
27     }
28
29     /**
30      * Size is set with a given Unit in mind.
31      * Remember to not compare apples to oranges!
32      *
33      * @return the size of the unit
34      */
35     public double getSize() {
36         return size;
37     }
38
39     /**
40      * @return the unit for the size
41      */
42     public Unit getUnit() {
43         return unit;
44     }
45 }
```

Chapter 46: model/product/Product.java

```
1 package model.product;
2
3 import controller.Controller;
4 import model.Pricelist;
5 import model.ProductGroup;
6 import model.product.rentable.NotRentable;
7 import model.product.rentable.RentableStrategy;
8
9 import java.io.Serializable;
10 import java.util.HashSet;
11 import java.util.Set;
12
13 public class Product implements Serializable {
14
15     private final int id;
16     private String name;
17     private String description;
18     private ProductGroup productGroup;
19     private Set<Pricelist> pricelists = new HashSet<>();
20     private RentableStrategy rentableStrategy;
21
22     /**
23      * Creates a product without a description.
24      *
25      * @pre productGroup != null
26      *
27      * @param name          of the product
28      * @param productGroup where the product belongs to
29      */
30     public Product(String name, ProductGroup productGroup) {
31         this.name = name;
32         this.productGroup = productGroup;
33         this.description = "";
34         id = Controller.getInstance().getProductList().size() > 0 ? Controller.
35             ↪ getInstance().getProductList().get(Controller.getInstance().
36             ↪ getProductList().size() - 1).id + 1 : 1;
37         this.rentableStrategy = new NotRentable();
38     }
39
40     /**
41      * Overloads the default constructor.
42      * The only difference is the description
43      * @param name (see other construct doc)
44      * @param productGroup (see other construct doc)
45      * @param description of the product
46      */
47     public Product(String name, ProductGroup productGroup, String description) {
48         this(name, productGroup);
```

```

47         this.description = description;
48     }
49
50     /**
51     * Adds this product to a chosen pricelist Has a bidirbidirectionalectional 0...*
52     * ↪ relationship
53     * with Pricelist
54     *
55     * @param pricelist, the pricelist you want the product added to
56     * @param price, the price of the product on the chosen pricelist
57     */
58     public void addPricelist(Pricelist pricelist, double price) {
59         if (!pricelists.contains(pricelist)) {
60             pricelist.addProductWithPrice(this, price);
61             pricelists.add(pricelist);
62         }
63     }
64
65     /**
66     * removes this product from a chosen pricelist Has a bidirectional 0...*
67     * relationship with Pricelist
68     *
69     * @param pricelist, the pricelist the product are removed from
70     */
71     public void removePricelist(Pricelist pricelist) {
72         if (pricelists.contains(pricelist)) {
73             pricelist.removeProductWithPrice(this);
74             pricelists.remove(pricelist);
75         }
76     }
77
78     /**
79     * all of the pricelists the product is a part of
80     * @return all of the pricelists product is part of
81     */
82     public Set<Pricelist> getPricelists() {
83         return pricelists;
84     }
85
86     /**
87     * Replace the old rentableStrategy with a new one
88     * @param rentableStrategy to replace existing
89     */
90     public void setRentableStrategy(RentableStrategy rentableStrategy) {
91         this.rentableStrategy = rentableStrategy;
92     }
93
94     /**
95     * name of the product
96     * @return name of the product
97     */
98     public String getName() {
99         return name;
100     }
101
102     /**
103     *
104     * @return the description of the product, can be empty string

```

```
104     */
105     public String getDescription() {
106         return description;
107     }
108
109     /**
110     * the group of which this product belongs to
111     * @return the group which is responsible for the product
112     */
113     public ProductGroup getProductGroup() {
114         return productGroup;
115     }
116
117     /**
118     * The method from the rentable strategy pattern
119     * (see report)
120     * @return the deposit of the product
121     */
122     public double getDeposit() {
123         return rentableStrategy.getDeposit();
124     }
125
126     /**
127     * The rentable strategy which counts for this product (see report)
128     * @return rentable strategy this product uses
129     */
130     public RentableStrategy getRentableStrategy() {
131         return rentableStrategy;
132     }
133
134     /**
135     * the products id
136     * @return id of the product
137     */
138     public int getId() {
139         return id;
140     }
141
142     @Override
143     public String toString() {
144         return name;
145     }
146 }
```

Chapter 47: model/product/Tour.java

```
1 package model.product;
2
3 import model.ProductGroup;
4
5 public class Tour extends Product {
6
7     /**
8      * All parameters are similar to Product.
9      * This class is only for identification (see the report)
10     * @param name name of the tour
11     * @param productGroup that the tour belongs to
12     * @param description of the tour
13     */
14     public Tour(String name, ProductGroup productGroup, String description) {
15         super(name, productGroup, description);
16     }
17 }
```

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

```
1 package model.product.rentable;
2
3 import java.io.Serializable;
4
5 /**
6  * Described in the report
7  */
8 public class NotRentable implements RentableStrategy, Serializable {
9
10     @Override
11     public double getDeposit() {
12         return -1;
13     }
14 }
```

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

```
1 package model.product.rentable;
2
3 import java.io.Serializable;
4
5 /**
6  * Described in the report
7  */
8 public class Rentable implements RentableStrategy, Serializable {
9     private double deposit;
10
11     /**
12      * Must have a deposit amount, however nothing prohibits it from being 0 (free)
13      * @param deposit amount to be paid for a rent
14      */
15     public Rentable(double deposit) {
16         this.deposit = deposit;
17     }
18
19     @Override
20     public double getDeposit() {
21         return deposit;
22     }
23
24 }
```


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

```
1 package model.product.rentable;
2
3 /**
4  * Described in the report
5  */
6 public interface RentableStrategy {
7     double getDeposit();
8 }
```

Chapter 51: model/ProductGroup.java

```
1 package model;
2
3 import model.enums.Unit;
4 import model.product.Bundle;
5 import model.product.Container;
6 import model.product.Product;
7 import model.product.Tour;
8
9 import java.io.Serializable;
10 import java.util.HashSet;
11 import java.util.Set;
12
13 public class ProductGroup implements Serializable {
14     private String name;
15     private Set<Product> products = new HashSet<>();
16
17     /**
18      * Creates a product group without any products
19      *
20      * @param name of the product group
21      */
22     public ProductGroup(String name) {
23         this.name = name;
24     }
25
26     /**
27      * creates a product, and adds it to this productgroup
28      *
29      * @param name, the name of the product
30      * @param description, description of the product
31      * @return the created product
32      */
33     public Product createProduct(String name, String description) {
34         Product product = new Product(name, this, description);
35         products.add(product);
36         return product;
37     }
38
39     /**
40      * Creates a container which is a subclass of product.
41      * It's different because it can contain size and unit, which is appropriate for beer.
42      * @param name of the container
43      * @param description belonging to the container
44      * @param size amount of the containers unit attribute
45      * @param unit of the measured size eg. litre or kg
46      * @return the newly created container product
47      */
48 }
```

```

48 public Container createContainerProduct(String name, String description, double size,
49 ↪ Unit unit) {
50     Container container = new Container(name, this, description, size, unit);
51     products.add(container);
52     return container;
53 }
54 /**
55  * Creates a tour which is almost equivalent to product.
56  * however, very desirable for identifying this particular type of product
57  * - amount of people on the tour is stored in the sale
58  *
59  * @param name of the tour
60  * @param description belonging to the tour
61  * @return the newly created tour product
62  */
63 public Tour createTourProduct(String name, String description) {
64     Tour tour = new Tour(name, this, description);
65     products.add(tour);
66     return tour;
67 }
68
69 /**
70  * Creates a bundle product.
71  * It's a little special because it can hold a list of other products.
72  *
73  * @param name of the bundle
74  * @param description belonging to the bundle
75  * @return the newly created bundle product
76  */
77 public Bundle createBundleProduct(String name, String description) {
78     Bundle bundle = new Bundle(name, this, description);
79     products.add(bundle);
80     return bundle;
81 }
82
83 /**
84  * deletes/removes the chosen product from the productgroup
85  *
86  * @param product, the product you want to delete
87  */
88 public void deleteProduct(Product product) {
89     products.remove(product);
90 }
91
92 /**
93  * The current name of the productGroup
94  *
95  * @return the name of the productGroup
96  */
97 public String getName() {
98     return name;
99 }
100
101 /**
102  * The list of products that belongs to the productgroup
103  * @return a set of products in the group
104  */

```

```
105     public Set<Product> getProducts() {  
106         return products;  
107     }  
108  
109     @Override  
110     public String toString() {  
111         return name;  
112     }  
113  
114 }
```

Chapter 52: model/Rent.java

```
1 package model;
2
3 import java.time.LocalDateTime;
4
5 public class Rent extends Sale {
6
7     private String contactName;
8     private String contactInformation;
9     private LocalDateTime deliveryDateAndTime;
10    private LocalDateTime returnDateAndTime;
11
12    /**
13     * Creates a rent sale.
14     * - this is also used by the tour class
15     * @param contactName of the person responsible for returning or the product(s)
16     * @param contactInformation phonenummer, mail or any other means of contact information
17     * @param deliveryDateAndTime the date and time which the rented product is either
18     *     ↪ picked up or delivered
19     * @param returnDateAndTime the date and time when the product must be returned
20     */
21    public Rent(String contactName, String contactInformation, LocalDateTime
22     ↪ deliveryDateAndTime,
23                LocalDateTime returnDateAndTime) {
24        super();
25        this.contactName = contactName;
26        this.contactInformation = contactInformation;
27        this.deliveryDateAndTime = deliveryDateAndTime;
28        this.returnDateAndTime = returnDateAndTime;
29    }
30
31    /**
32     * Gets the person attached to the rent
33     * @return name of person responsible for returning the product
34     */
35    public String getContactName() {
36        return contactName;
37    }
38
39    /**
40     * Information about the person
41     * @return information about the person
42     */
43    public String getContactInformation() {
44        return contactInformation;
45    }
46
47    /**
```

```
47     * When the product(s) are picked up or delivered.
48     * If tour, when it's agreed to start.
49     * @return date and time agreed
50     */
51     public LocalDateTime getDeliveryDateAndTime() {
52         return deliveryDateAndTime;
53     }
54
55     /**
56     * When product(s) must be returned
57     * @return date and time agreed for return
58     */
59     public LocalDateTime getReturnDateAndTime() {
60         return returnDateAndTime;
61     }
62 }
```

Chapter 53: model/Sale.java

```
1 package model;
2
3 import controller.Controller;
4 import model.enums.PaymentMethod;
5 import model.enums.SaleState;
6 import model.product.Product;
7
8 import java.io.Serializable;
9 import java.time.LocalDateTime;
10 import java.util.LinkedList;
11 import java.util.List;
12
13 public class Sale implements Serializable {
14     private final int id;
15     private LocalDateTime timestamp;
16     private double agreedPrice;
17     private Payment payment;
18     private List<SalesLine> salesLines = new LinkedList<>();
19     private SaleState saleState;
20
21     /**
22      * Creates a sale, which is initiated with:
23      * - an id which calculated is (because of serializable) to be last id of known product
24      *   ↪ + 1
25      * - (-1) as default agreed price, which is also discount. Uses -1 because of JavaFX
26      * - time of when the sale is initiated
27      * - the state of the sale is default initiated until set otherwise
28      */
29     public Sale() {
30         id = Controller.getInstance().getSales().size() > 0 ? Controller.getInstance().
31             ↪ getSales().get(Controller.getInstance().getSales().size() - 1).id + 1 : 1;
32         agreedPrice = -1;
33         timestamp = LocalDateTime.now();
34         payment = new Payment();
35         saleState = SaleState.INITIATED;
36     }
37
38     /**
39      * Either changes a salesline if similar product and price is found.
40      * Else it creates a new salesline.
41      *
42      * @param product that is either added or updated on salesline
43      * @param quantity amount to modify with
44      * @param price PER 1 quantity of the product
45      * @return the created or updated salesline
46      * @pre: product not null
47      * @invariants: quantity see test report for further description
48      */
49 }
```

```

47 public SalesLine updateSalesLine(Product product, int quantity, double price) {
48     // Check if saleLine exist
49     boolean saleLineFound = false;
50     int i = 0;
51     while (i < salesLines.size() && !saleLineFound) {
52         boolean productsSame = salesLines.get(i).getProduct().equals(product);
53         boolean priceSame = salesLines.get(i).getPrice() == price;
54         if (productsSame && priceSame)
55             saleLineFound = true;
56         else
57             i++;
58     }
59
60     // If not found, create new salesLine
61     SalesLine salesLine = !saleLineFound ? createSalesLine(product, quantity, price) :
        ↪ salesLines.get(i);
62
63     // Add and return if new created
64     if (!saleLineFound) {
65         salesLines.add(salesLine);
66         return salesLine;
67     }
68
69     // Update quantity
70     salesLine.setQuantity(salesLine.getQuantity() + quantity);
71
72     // If quantity == 0, remove
73     if (salesLine.getQuantity() == 0)
74         removeSalesLine(salesLine);
75
76     return salesLine;
77 }
78
79 /**
80  * Creates a new salesline. This method is private and is supposed to be called through
81  * the public interface method updateSalesLine()
82  * @param product to be added to the salesline
83  * @param quantity of the product in the salesline
84  * @param price price PER product
85  * @return SalesLine
86  */
87 private SalesLine createSalesLine(Product product, int quantity, double price) {
88     return new SalesLine(product, quantity, price);
89 }
90
91 /**
92  * Removes salesline from sale.
93  * @param salesLine to be removed
94  */
95 private void removeSalesLine(SalesLine salesLine) {
96     salesLines.remove(salesLine);
97 }
98
99 /**
100  * Calculate the total price of the sale.
101  *
102  * @return the total price of the sale.
103  */

```



```
104 public double totalSalePrice() {
105     double total = 0;
106     for (SalesLine salesLine : salesLines) {
107         total += salesLine.getTotalLinePrice();
108     }
109     return total;
110 }
111
112 /**
113  * Returns the payment attached to this sale.
114  * A payment can hold multiple transfers, and is read this payment.
115  * @return payment with transfers for this sale
116  */
117 public Payment getPayment() {
118     return payment;
119 }
120
121 /**
122  * Set's the timestamp for the sale,
123  * unused in the application but used to make repeatable tests.
124  * @param timestamp only use for testing!
125  */
126 public void setTimestamp(LocalDateTime timestamp) {
127     this.timestamp = timestamp;
128 }
129
130 /**
131  * Returns when the sale has been initiated or rather object has been created.
132  * @return date and time of sale initiation
133  */
134 public LocalDateTime getTimestamp() {
135     return timestamp;
136 }
137
138 /**
139  * Adds a transfer to the payment of a sale.
140  *
141  * @param paymentMethod, method of the payment.
142  * @param amount, the amount of money spend in the transfer.
143  */
144 public void addTransfer(PaymentMethod paymentMethod, double amount) {
145     payment.addTransfer(paymentMethod, amount);
146 }
147
148 /**
149  * Changes the state of the sale,
150  * @param saleState to be changed to
151  */
152 public void setSaleState(SaleState saleState) {
153     this.saleState = saleState;
154 }
155
156 /**
157  * Gets the state of the sale
158  * @return the state of the sale
159  */
160 public SaleState getSaleState() {
161     return saleState;
```

```
162     }
163
164     /**
165      * @return The id of the sale
166      */
167     public int getId() {
168         return id;
169     }
170
171     /**
172      * All the saleslines created through the sale.
173      *
174      * @return a list of saleslines
175      */
176     public List<SalesLine> getSalesLines() {
177         return new LinkedList<>(salesLines);
178     }
179
180     /**
181      * The agreed price for the entire sale AND thus also saleslines.
182      * @return the agreed price for the entire sale
183      */
184     public double getAgreedPrice() {
185         return agreedPrice;
186     }
187
188     /**
189      * Set a agreed price (most likely a discount) for the entire sale
190      * @param agreedPrice of the entire sale
191      */
192     public void setAgreedPrice(double agreedPrice) {
193         this.agreedPrice = agreedPrice;
194     }
195 }
```

Chapter 54: model/SalesLine.java

```
1 package model;
2
3 import model.product.Product;
4 import java.io.Serializable;
5
6 public class SalesLine implements Serializable {
7
8     private Product product;
9     private int quantity;
10    private double price;
11
12    /**
13     * Creates a salesline with the values from the attributes
14     * @param product in the salesline
15     * @param quantity of the products in the salesline
16     * @param price per product in the salesline
17     */
18    public SalesLine(Product product, int quantity, double price) {
19        this.quantity = quantity;
20        this.product = product;
21        this.price = price;
22    }
23
24    /**
25     * the amount of products sold through this salesline
26     * @return quantity of products sold through this salesline
27     */
28    public int getQuantity() {
29        return quantity;
30    }
31
32    /**
33     * sets the amount of products to be sold through this salesline
34     * @param quantity of products sold in this salesline, overwrites old value
35     */
36    public void setQuantity(int quantity) {
37        this.quantity = quantity;
38    }
39
40    /**
41     * the product in this salesline
42     * @return the product in the salesline
43     */
44    public Product getProduct() {
45        return product;
46    }
47
48    /**
```

```
49     * the price per product in this salesline
50     * @return price per product in the salesline
51     */
52     public double getPrice() {
53         return price;
54     }
55
56     /**
57     * Used if you want to set a new price to the product in the salesline.
58     *
59     * @param price, the new price
60     */
61     public void setNewPrice(double price) {
62         this.price = price;
63     }
64
65     /**
66     * Calculate the total price of the salesline
67     *
68     * @return the total price of the salesline
69     */
70     public double getTotalLinePrice() {
71         return price * quantity;
72     }
73
74     /**
75     * Calculate the total deposit of the salesline
76     *
77     * @return the total deposit of the salesline
78     */
79     public double getTotalLineDeposit() {
80         return product.getDeposit() * quantity;
81     }
82
83 }
```

Chapter 55: model/SalesLine.java

```
1 package model;
2
3 import model.product.Product;
4 import java.io.Serializable;
5
6 public class SalesLine implements Serializable {
7
8     private Product product;
9     private int quantity;
10    private double price;
11
12    /**
13     * Creates a salesline with the values from the attributes
14     * @param product in the salesline
15     * @param quantity of the products in the salesline
16     * @param price per product in the salesline
17     */
18    public SalesLine(Product product, int quantity, double price) {
19        this.quantity = quantity;
20        this.product = product;
21        this.price = price;
22    }
23
24    /**
25     * the amount of products sold through this salesline
26     * @return quantity of products sold through this salesline
27     */
28    public int getQuantity() {
29        return quantity;
30    }
31
32    /**
33     * sets the amount of products to be sold through this salesline
34     * @param quantity of products sold in this salesline, overwrites old value
35     */
36    public void setQuantity(int quantity) {
37        this.quantity = quantity;
38    }
39
40    /**
41     * the product in this salesline
42     * @return the product in the salesline
43     */
44    public Product getProduct() {
45        return product;
46    }
47
48    /**
```

```
49     * the price per product in this salesline
50     * @return price per product in the salesline
51     */
52     public double getPrice() {
53         return price;
54     }
55
56     /**
57     * Used if you want to set a new price to the product in the salesline.
58     *
59     * @param price, the new price
60     */
61     public void setNewPrice(double price) {
62         this.price = price;
63     }
64
65     /**
66     * Calculate the total price of the salesline
67     *
68     * @return the total price of the salesline
69     */
70     public double getTotalLinePrice() {
71         return price * quantity;
72     }
73
74     /**
75     * Calculate the total deposit of the salesline
76     *
77     * @return the total deposit of the salesline
78     */
79     public double getTotalLineDeposit() {
80         return product.getDeposit() * quantity;
81     }
82
83 }
```

Chapter 56: storage/Storage.java

```
1 package storage;
2
3 import model.Pricelist;
4 import model.ProductGroup;
5 import model.Sale;
6 import model.product.Product;
7
8 import java.io.Serializable;
9 import java.util.LinkedList;
10 import java.util.List;
11
12 public class Storage implements Serializable {
13     private static Storage storage;
14
15     private List<ProductGroup> productGroups = new LinkedList<>();
16     private List<Product> products = new LinkedList<>();
17     private List<Pricelist> pricelists = new LinkedList<>();
18     private List<Sale> sales = new LinkedList<>();
19
20     public void addProductGroup(ProductGroup productGroup) {
21         productGroups.add(productGroup);
22     }
23
24     public void addProduct(Product product) {
25         products.add(product);
26     }
27
28     public void addPriceList(Pricelist pricelist) {
29         pricelists.add(pricelist);
30     }
31
32     public void addSale(Sale sale) {
33         sales.add(sale);
34     }
35
36     //
37     ↪ -----
38     ↪
39
40     public List<ProductGroup> getProductGroupList() {
41         return new LinkedList<>(productGroups);
42     }
43
44     public List<Product> getProductList() {
45         return new LinkedList<>(products);
46     }
47
48     public List<Pricelist> getPricelists() {
```

```
47         return new LinkedList<>(pricelists);
48     }
49
50     public List<Sale> getSales() {
51         return new LinkedList<>(sales);
52     }
53
54     //
55     //
56     public static Storage getInstance() {
57         if (storage == null) {
58             storage = new Storage();
59         }
60         return storage;
61     }
62
63 }
```


Chapter 57: tests/Integration/UseCase4Test.java

```
1 package Tests.Integration;
2
3 import controller.Controller;
4 import model.Pricelist;
5 import model.ProductGroup;
6 import model.Sale;
7 import model.SalesLine;
8 import model.product.Product;
9 import org.junit.Before;
10 import org.junit.Test;
11
12 import static org.junit.Assert.assertEquals;
13
14 public class UseCase4Test {
15     private Pricelist pricelist;
16     private Product productChips;
17     private Product productPeanuts;
18
19     @Before
20     public void setUp() {
21         Controller.getInstance().initStorageOnly();
22         pricelist = new Pricelist("Fredagsbar");
23         ProductGroup productGroupSnacks = new ProductGroup("Snacks");
24         productChips = new Product("Chips", productGroupSnacks);
25         productPeanuts = new Product("Peanuts", productGroupSnacks);
26         pricelist.addProductWithPrice(productChips, 15);
27         pricelist.addProductWithPrice(productPeanuts, 10);
28     }
29
30     @Test
31     public void IT1() {
32         Sale sale = new Sale();
33         sale.updateSalesLine(productChips, 1, pricelist.getPriceOfProduct(productChips));
34         sale.updateSalesLine(productPeanuts, 1, pricelist.getPriceOfProduct(productPeanuts))
35             ↪ ;
36         assertEquals(2, sale.getSalesLines().size());
37     }
38
39     @Test
40     public void IT2() {
41         Sale sale = new Sale();
42         SalesLine salesLine = sale.updateSalesLine(productChips, 1, pricelist.
43             ↪ getPriceOfProduct(productChips));
44         sale.updateSalesLine(productPeanuts, 1, pricelist.getPriceOfProduct(productPeanuts))
45             ↪ ;
46         sale.updateSalesLine(productChips, 2, pricelist.getPriceOfProduct(productChips));
47         assertEquals(3, salesLine.getQuantity());
48     }
49 }
```

```
46
47     @Test
48     public void IT3() {
49         Sale sale = new Sale();
50         SalesLine salesLine = sale.updateSalesLine(productChips, 1, pricelist.
51             ↪ getPriceOfProduct(productChips));
52         sale.updateSalesLine(productPeanuts, 1, pricelist.getPriceOfProduct(productPeanuts))
53             ↪ ;
54         salesLine.setNewPrice(5);
55         assertEquals(5, salesLine.getPrice(), 0.001);
56         assertEquals(15, sale.totalSalePrice(), 0.001);
57     }
```

Chapter 58: tests/Unit/UseCase4Test.java

```
1 package Tests.Unit;
2
3 import controller.Controller;
4 import model.ProductGroup;
5 import model.product.Bundle;
6 import model.product.Product;
7 import org.junit.Before;
8 import org.junit.Test;
9
10 import java.util.HashMap;
11
12 import static org.junit.Assert.*;
13
14 public class BundleTest {
15     private ProductGroup pgBar;
16     private Bundle bundle;
17     private Product p1;
18     private Product p2;
19     private Product p3;
20
21     /*
22      * Da vi ikke overskriver metoder fra superklassen "Product" regner vi med at disse
23      * ↪ virker,
24      * såfremt Product består sine tests.
25      */
26
27     @Before
28     public void setUp() throws Exception {
29         Controller.getInstance().initStorageOnly();
30         pgBar = new ProductGroup("Gaveøsker");
31         bundle = new Bundle("Gaveøske 1", pgBar, "");
32         p1 = new Product("Klosterbryg", pgBar);
33         p2 = new Product("Forårsbryg", pgBar);
34         p3 = new Product("Kellsberg", pgBar);
35     }
36
37     @Test
38     public void addAndGetProductExpect1Quantity() {
39         bundle.addProduct(p1);
40         HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
41         ↪ ();
42         int quantity = bundleProductsAndQuantity.get(p1);
43         assertEquals(1, quantity);
44     }
45
46     @Test
47     public void addAndGetProductExpect1Quantity2() {
48         bundle.addProduct(p1);
```

```
47     bundle.addProduct(p1);
48     HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
    ↪ ();
49     int quantity = bundleProductsAndQuantity.get(p1);
50     assertEquals(2, quantity);
51 }
52
53 @Test
54 public void addMultipleProductsAndGetProductExpect1Quantity1() {
55     bundle.addProduct(p1);
56     bundle.addProduct(p3);
57     HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
    ↪ ();
58     int quantity = bundleProductsAndQuantity.get(p1);
59     assertEquals(1, quantity);
60 }
61
62 @Test
63 public void addProduct2TimesRemoveItOnceExpect1Quantity() {
64     bundle.addProduct(p1);
65     bundle.addProduct(p1);
66     bundle.removeProduct(p1);
67     HashMap<Product, Integer> bundleProductsAndQuantity = bundle.getProductsWithQuantity
    ↪ ();
68     int quantity = bundleProductsAndQuantity.get(p1);
69     assertEquals(1, quantity);
70 }
71
72 @org.junit.Test
73 public void getName() {
74     Bundle bundleGift1 = new Bundle("Gaveæske 1", pgBar, "");
75     assertEquals("Gaveæske 1", bundleGift1.getName());
76 }
77 }
```

Chapter 59: tests/Unit/ContainerTest.java

```
1 package Tests.Unit;
2
3 import controller.Controller;
4 import model.ProductGroup;
5 import model.enums.Unit;
6 import model.product.Container;
7 import org.junit.Before;
8 import org.junit.Test;
9
10 import static org.junit.Assert.assertEquals;
11
12 public class ContainerTest {
13     private Container container;
14     private ProductGroup pgBar;
15
16     @Before
17     public void setUp() throws Exception {
18         Controller.getInstance().initStorageOnly();
19         pgBar = new ProductGroup("Ø1");
20         container = new Container("Klosterbryg", pgBar, "", 60, Unit.cl);
21     }
22
23
24     @Test
25     public void getSize() {
26         assertEquals(60, container.getSize(), 0.01);
27     }
28
29     @Test
30     public void getUnit() {
31         assertEquals(Unit.cl, container.getUnit());
32     }
33 }
```

Chapter 60: tests/Unit/PaymentTest.java

```
1 package Tests.Unit;
2
3 import model.Payment;
4 import model.enums.PaymentMethod;
5 import org.junit.Before;
6 import org.junit.Test;
7
8 import static org.junit.Assert.assertEquals;
9
10 public class PaymentTest {
11
12     @Before
13     public void setUp() throws Exception {
14     }
15
16     @Test
17     public void payment0Transfer() {
18         Payment payment = new Payment();
19         assertEquals(payment.calcTotal(), 0, 0.001);
20     }
21
22     @Test
23     public void payment1TransferCash() {
24         // 1 payment with cash
25         Payment payment = new Payment();
26         payment.addTransfer(PaymentMethod.CASH, 200);
27         assertEquals(payment.calcTotal(), 200, 0.001);
28     }
29
30     @Test
31     public void payment2TransferCash() {
32         // 1 payment with cash
33         Payment payment = new Payment();
34         payment.addTransfer(PaymentMethod.CASH, 200);
35         payment.addTransfer(PaymentMethod.CASH, 200);
36         assertEquals(payment.calcTotal(), 400, 0.001);
37     }
38
39     @Test
40     public void payment2TransferCashAndPaymentCard() {
41         // 2 payments total, 1 with cash and 1 with card
42         Payment payment = new Payment();
43         payment.addTransfer(PaymentMethod.CASH, 200);
44         payment.addTransfer(PaymentMethod.PAYMENTCARD, 200);
45         assertEquals(payment.calcTotal(), 400, 0.001);
46     }
47
48 }
```

```
49     @Test
50     public void paymentNegativeForReturnSale() {
51         // 2 payments total, 1 with cash and 1 with card
52         Payment payment = new Payment();
53         payment.addTransfer(PaymentMethod.CASH, -200);
54         assertEquals(payment.calcTotal(), -200, 0.001);
55     }
56 }
```

Chapter 61: tests/Unit/PricelistTest.java

```
1 package Tests.Unit;
2
3 import model.Pricelist;
4 import model.ProductGroup;
5 import model.enums.Unit;
6 import model.product.Container;
7 import org.junit.Before;
8 import org.junit.Test;
9
10 import static org.junit.Assert.*;
11
12 public class PricelistTest {
13     private ProductGroup pgBar;
14     private Container klosterbryg;
15     private Container kellsberg;
16     private Pricelist pricelist;
17
18     @Before
19     public void setUp() throws Exception {
20         pgBar = new ProductGroup("Ø1");
21         klosterbryg = new Container("Klosterbryg", pgBar, "", 0.60, Unit.cl);
22         kellsberg = new Container("Kellsberg", pgBar, "", 60, Unit.cl);
23         pricelist = new Pricelist("Bar");
24     }
25
26     @Test
27     public void addAndGetProductWithPrice() {
28         pricelist.addProductWithPrice(kellsberg, 70);
29         assertTrue(pricelist.getProductsWithPrice().containsKey(kellsberg));
30     }
31
32     @Test
33     public void removeProductWithPrice() {
34         pricelist.addProductWithPrice(kellsberg, 70);
35         pricelist.removeProductWithPrice(kellsberg);
36         assertFalse(pricelist.getProductsWithPrice().containsKey(kellsberg));
37     }
38
39     @Test
40     public void updateProductPriceAndPriceOfProduct() {
41         pricelist.addProductWithPrice(kellsberg, 70);
42         pricelist.updateProductPrice(kellsberg, 90);
43         assertEquals(90, pricelist.getPriceOfProduct(kellsberg), 0.001);
44     }
45
46     @Test(expected = IllegalArgumentException.class)
47     public void getPriceOfNotExistingProduct() {
48         pricelist.getPriceOfProduct(klosterbryg);
49     }
50 }
```



```
49     }
50
51     @Test
52     public void getName() {
53         Pricelist pricelist = new Pricelist("Rundvisning");
54         assertEquals("Rundvisning", pricelist.getName());
55     }
56
57 }
```

Chapter 62: tests/Unit/ProductTest.java

```
1 package Tests.Unit;
2
3 import controller.Controller;
4 import model.Pricelist;
5 import model.ProductGroup;
6 import model.product.Product;
7
8 import static org.junit.Assert.*;
9
10 public class ProductTest {
11     private Pricelist plBar;
12     private Product product;
13
14     @org.junit.Before
15     public void setUp() {
16         Controller.getInstance().initStorageOnly();
17         ProductGroup pgBar = new ProductGroup("Ø1");
18         plBar = new Pricelist("Bar pricelist");
19         product = new Product("Klosterbryg", pgBar);
20     }
21
22     @org.junit.Test
23     public void addAndRemovePriceList() {
24         /*
25          Det kan diskuteres hvor denne metode bør testes, om det er en Integrationstest.
26          Vi tester den her, og betragter metoderne add og remove der opretholder
27          den dobbelttrettede associering og som sideeffekt påvirker Pricelist.
28          */
29
30         // Add virker dobbeltrettet
31         product.addPricelist(plBar, 70.0);
32         assertEquals(70.0, plBar.getPriceOfProduct(product), 0.001);
33         // Remove virker dobbeltrettet
34         product.removePricelist(plBar);
35         assertFalse(plBar.getProductsWithPrice().containsKey(product));
36     }
37
38     @org.junit.Test
39     public void getName() {
40         assertEquals("Klosterbryg", product.getName());
41     }
42 }
```

Chapter 63: tests/Unit/RentAndTourTest.java

```
1 package Tests.Unit;
2
3 import controller.Controller;
4 import model.ProductGroup;
5 import model.Rent;
6 import model.product.Tour;
7 import model.product.rentable.Rentable;
8 import org.junit.Before;
9 import org.junit.Test;
10
11 import java.time.LocalDateTime;
12
13 import static org.junit.Assert.assertEquals;
14 import static org.junit.Assert.assertTrue;
15
16 public class RentAndTourTest {
17     private LocalDateTime date;
18     private String contactPersonName;
19     private String contactPersonNumber;
20     private Tour tour;
21     private Rent rent;
22
23     @Before
24     public void setUp() {
25         Controller.getInstance().initStorageOnly();
26         //TODO : UPDATE UPDATE UPDATE
27         ProductGroup pgBar = new ProductGroup("Rundvisninger");
28         date = LocalDateTime.of(2020, 01, 01, 12, 30);
29         contactPersonName = "Mikael";
30         contactPersonNumber = "12345678";
31         tour = new Tour("Rundvisning 100kr", pgBar, "");
32         tour.setRentableStrategy(new Rentable(100));
33         rent = new Rent(contactPersonName, contactPersonNumber, date, date);
34     }
35
36     @Test
37     public void tourRentableTest() {
38         assertTrue(tour.getRentableStrategy() instanceof Rentable);
39     }
40
41     @Test
42     public void getStartDateAndTime() {
43         assertEquals(date, rent.getDeliveryDateAndTime());
44     }
45
46     @Test
47     public void getContactPersonName() {
48         assertEquals(contactPersonName, rent.getContactName());
```

```
49     }
50
51     @Test
52     public void getContactPersonNumber() {
53         assertEquals(contactPersonNumber, rent.getContactInformation());
54     }
55 }
```

Chapter 64: tests/Unit/SalesLineTest.java

```
1 package Tests.Unit;
2
3 import model.ProductGroup;
4 import model.SalesLine;
5 import model.product.Bundle;
6 import model.product.Product;
7 import org.junit.Before;
8 import org.junit.Test;
9
10 import static org.junit.Assert.*;
11
12 public class SalesLineTest {
13     private Product p1;
14
15     @Before
16     public void setUp() throws Exception {
17         ProductGroup pgBundles = new ProductGroup("Gavepakker");
18         p1 = new Bundle("Valentinspakke til Karsten", pgBundles, "");
19     }
20
21     @Test
22     public void setAndGetQuantity() {
23         SalesLine salesLine = new SalesLine(p1, 1, 200);
24         assertEquals(1, salesLine.getQuantity());
25         salesLine.setQuantity(5);
26         assertEquals(5, salesLine.getQuantity());
27     }
28
29
30     @Test
31     public void getProduct() {
32         SalesLine salesLine = new SalesLine(p1, 1, 200);
33         assertEquals(p1, salesLine.getProduct());
34     }
35
36     @Test
37     public void totalLinePrice1Product() {
38         SalesLine salesLine = new SalesLine(p1, 1, 200);
39         assertEquals(200, salesLine.getTotalLinePrice(), 0.001);
40     }
41
42     @Test
43     public void totalLinePrice2Products() {
44         SalesLine salesLine = new SalesLine(p1, 2, 200);
45         assertEquals(400, salesLine.getTotalLinePrice(), 0.001);
46     }
47
48     @Test
```

```
49     public void setNewPrice() {  
50         SalesLine salesLine = new SalesLine(p1, 1, 200);  
51         salesLine.setNewPrice(100);  
52         assertEquals(100, salesLine.getTotalLinePrice(), 0.001);  
53     }  
54  
55  
56 }
```

Chapter 65: tests/Unit/SaleTest.java

```
1 package Tests.Unit;
2
3 import controller.Controller;
4 import model.ProductGroup;
5 import model.Sale;
6 import model.SalesLine;
7 import model.enums.PaymentMethod;
8 import model.product.Product;
9 import org.junit.Before;
10 import org.junit.Test;
11
12 import static org.junit.Assert.*;
13
14 public class SaleTest {
15     private Product p1;
16     private Product p2;
17
18     @Before
19     public void setUp() {
20         Controller.getInstance().initStorageOnly();
21         ProductGroup pgBundles = new ProductGroup("Snacks");
22         p1 = new Product("Chips", pgBundles);
23         p2 = new Product("Peanuts", pgBundles);
24     }
25
26     // Metode: updateSales(...)
27
28     @Test
29     public void TC1addToSale1SalesLine() {
30         Sale sale = new Sale();
31         SalesLine salesLine = sale.updateSalesLine(p1, 1, 10);
32         assertTrue(sale.getSalesLines().contains(salesLine));
33     }
34
35     @Test
36     public void TC2totalSalePriceAddToSale() {
37         Sale sale = new Sale();
38         sale.updateSalesLine(p1, 1, 10);
39         assertEquals(10, sale.totalSalePrice(), 0.001);
40     }
41
42     @Test
43     public void TC3addSameProductTwiceToSalesLine() {
44         Sale sale = new Sale();
45         sale.updateSalesLine(p1, 1, 10);
46         sale.updateSalesLine(p1, 1, 10);
47         assertEquals(1, sale.getSalesLines().size());
48         assertEquals(20, sale.totalSalePrice(), 0.001);
49     }
50 }
```

```
49     }
50
51     @Test
52     public void TC4addSameProductTwiceDifferentPriceToSalesLine() {
53         Sale sale = new Sale();
54         sale.updateSalesLine(p1, 1, 10);
55         sale.updateSalesLine(p1, 1, 15);
56         assertEquals(2, sale.getSalesLines().size());
57         assertEquals(25, sale.totalSalePrice(), 0.001);
58     }
59
60     @Test
61     public void TC5addTwoProductsToSalesLine() {
62         Sale sale = new Sale();
63         sale.updateSalesLine(p1, 1, 10);
64         sale.updateSalesLine(p2, 1, 5);
65         assertEquals(2, sale.getSalesLines().size());
66         assertEquals(15, sale.totalSalePrice(), 0.001);
67     }
68
69     @Test
70     public void TC6ifQuantityOfSalesIs0Remove() {
71         Sale sale = new Sale();
72         SalesLine salineLine1 = sale.updateSalesLine(p1, 1, 10);
73         assertEquals(1, sale.getSalesLines().size());
74         sale.updateSalesLine(p1, -1, 10);
75         assertEquals(0, sale.getSalesLines().size());
76         assertFalse(sale.getSalesLines().contains(salineLine1));
77         assertEquals(0, sale.totalSalePrice(), 0.001);
78     }
79
80     // Metode: addTransfer(...)
81
82     @Test
83     public void TC7addTransfer() {
84         Sale sale = new Sale();
85         sale.addTransfer(PaymentMethod.CASH, 10);
86         assertTrue(sale.getPayment().getTransfers().containsKey(PaymentMethod.CASH));
87         assertEquals(10, sale.getPayment().calcTotal(), 0.0);
88     }
89
90     @Test
91     public void TC8add2TransferDifferentPaymentMethods() {
92         Sale sale = new Sale();
93         sale.addTransfer(PaymentMethod.CASH, 10);
94         sale.addTransfer(PaymentMethod.MOBILEPAY, 15);
95         assertTrue(sale.getPayment().getTransfers().containsKey(PaymentMethod.CASH));
96         assertTrue(sale.getPayment().getTransfers().containsKey(PaymentMethod.MOBILEPAY));
97         assertEquals(25, sale.getPayment().calcTotal(), 0.0);
98     }
99
100 }
```


Chapter 66: tests/Unit/UnitSuite.java

```
1 package Tests.Unit;
2
3 import org.junit.runner.RunWith;
4 import org.junit.runners.Suite;
5 import org.junit.runners.Suite.SuiteClasses;
6
7 @RunWith(Suite.class)
8 @SuiteClasses({ProductTest.class, BundleTest.class, ContainerTest.class, RentAndTourTest.
9     ↪ class, PaymentTest.class, PricelistTest.class, SalesLineTest.class, SaleTest.class})
9 public class UnitSuite {
10
11 }
```