



## **ASSIGNMENT # 1**

### **DESIGN DEFECTS & RESTRUCTURING (BCS-8A)**

**HASSAN ALI  
K20-1052**

**16<sup>TH</sup> FEB**

# Assignment #1

DDR, Spring 2024

1

**Question 1:** In this assignment, you need to describe your own example of any design pattern of your choice and give answers to all of the parts of the question 1. (Part A to Part D – see below) But your scenario / example must be a non-programming example explaining any real-life scenario, like the one given below.

## Observer Example

The Observer defines a one to many relationships, so that when one object changes state, the others are notified and updated automatically. Some auctions demonstrate this pattern. Each bidder possesses a numbered paddle that is used to indicate a bid. The auctioneer starts the bidding, and "observes" when a paddle is raised to accept the bid. The acceptance of the bid changes the bid price, which is broadcast to all of the bidders in the form of a new bid.

### Part (A) Give your own example (textual + graphic representation):

Describe your own example of any design pattern that we have covered in the class and give a substitute case of the design pattern being utilized in some other situation. (as shown in the above example) It very well may be a regular day to day example (e.g., how online goods are shipped, how to leave a place of employment and join another organization, ... whatever). In a manner similar to above example, briefly describe your example with a diagram and a brief textual description.

**Important Note:** The model you pick ought not be utilized in the GoF book (also minor varieties of the given examples are not allowed) or one that was utilized in the lecture slides or notes, neither it should be taken from any web article.

### Part (B) The problem, The Intent, The motivation and The Applicability

Clarify how your case "coordinates" the example's essential thought, that is, the way your model is closely resembling the structure example's model. Typically, you should clarify how your model synchronizes with (1) the intent and (2) the problem being discuss. From the GoF patterns, it should coordinate the aim or intent, inspiration and applicability.

### Part (C) UML

Draw a class diagram and interaction diagrams for the model you picked. What you sketch will rely upon the pattern you decided for your model. Utilize standard notations of UML.

### Part (D) Consequences

Document a sensible situation that features one of the consequences (disadvantages) of the design pattern. That is, your case study must be feature an outcome that is a downside in the situation. If your situation should highlight how the solution failed to address your requirements. Else you should portray a situation wherein neglecting to utilize the pattern would bring about negative outcomes. For example, in strategy pattern there is a communication overhead between Strategy and its Context, you'll need tighter coupling between Strategy and Context.

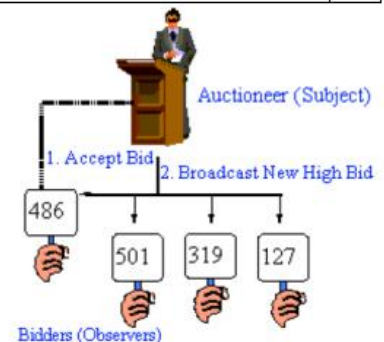
**Part(E)** Write or Generate skeleton Code of class diagram in part (C).

### Question 2 – Pattern identification:

Consider each of the scenarios below and identify the design pattern which is most directly addresses the problem described. Briefly explain your reasoning. (See what maps from GOF patterns and discuss and argue in favor of your thinking)

**Part (A):** You've developed a new implementation of the List interface and you'd like to test the behavior of your new data structure. You've written an algorithm to perform your speed tests, but the algorithm needs to make a great many instances of your list class. You want to test the performance of your list against the performance of ArrayList and LinkedList, but you don't want to have to write your algorithm three times in order for it to be able to create the right kind of list to test.

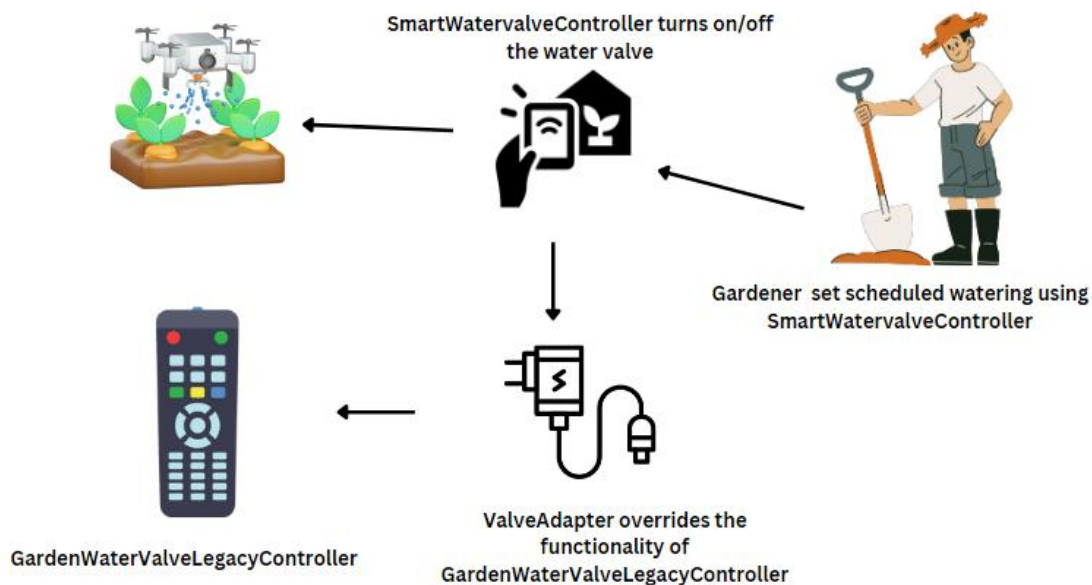
**Part (B)** You've completed a compiler for a new language! There are many parts to your compilation process: parsing, transformation, assembly code generation, and so forth. You'd like to allow other programmers to use an interface to compile their code without resorting to system calls or other command-line invocations – your compiler can just run in their processes – but you don't want those users to have to know how to bring all of the steps of compilation together in order to use your compiler.



## Solution (Adapter Pattern)

### Q1

A) We have a Smart Garden Watering System in which there is a **WaterApp** use to automate garden water schedules. **Valves** that controls the water flow to different areas of garden. **GardenWaterValvesLegacyController** that controls the old valve hardware and a **SmartWaterValveController** that controls the hardware using the Wifi. The WateringApp wants to integrate with new smart SmartWaterValveController to control Valves, however, the old legacy GardenWaterValveLegacyController is still installed to operate the valves manually. Therefore we need a **ValveAdapter** to adapt the new controller interfaces into the commands compatible with the old legacy controller. In this way we can integrate the new controller while reusing the old valve hardware.



### B) The Intent:

The intent of Adapter pattern is to convert one interface into another so that classes can work together that couldn't otherwise because of incompatible interfaces.

My example showcases this by adapting between the new smart water valve controller interfaces and the legacy garden valve controller interfaces, allowing the new and old valve systems to work together from the WateringApp client perspective.

### The Problem:

The Adapter pattern solves problems like:

- Reusing legacy components with incompatible interfaces
- Integrating with 3rd party systems having different interfaces
- Allowing substitution of components without changing clients

In my example, the problem is integrating the new WiFi valve control system while reusing the legacy valves. The ValveAdapter allows this integration and substitution of smart valve controller without changing the WateringApp.

### **The Motivation:**

Typical motivations are:

- Client independence from concrete implementation
- Reuse existing functionality
- Introduce substitutes without changing clients

The motivation in my system is to enable client (WateringApp) independence from underlying valve control hardware. It can automatically control watering while allowing manual operation, substituting newer components over time.

### **The Applicability:**

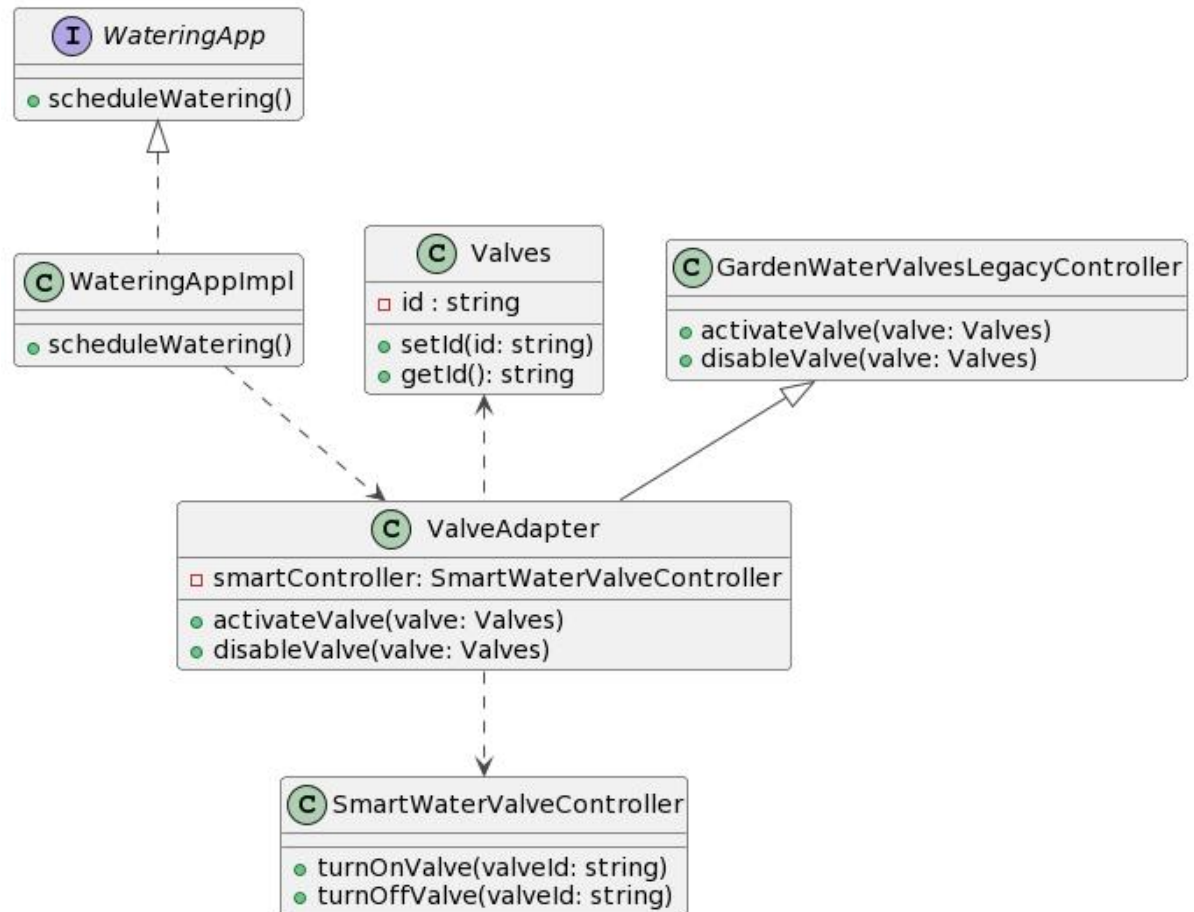
Adapter pattern applies when:

- A class's interface does not match what client requires
- Reusing classes even if interfaces don't match
- Features need to be used from incompatible libraries

In my case, it enables integrating the new SmartWaterValveController with incompatible legacy valve interfaces without replacing entire hardware. Allowing reuse of existing valves while adding new automated features.

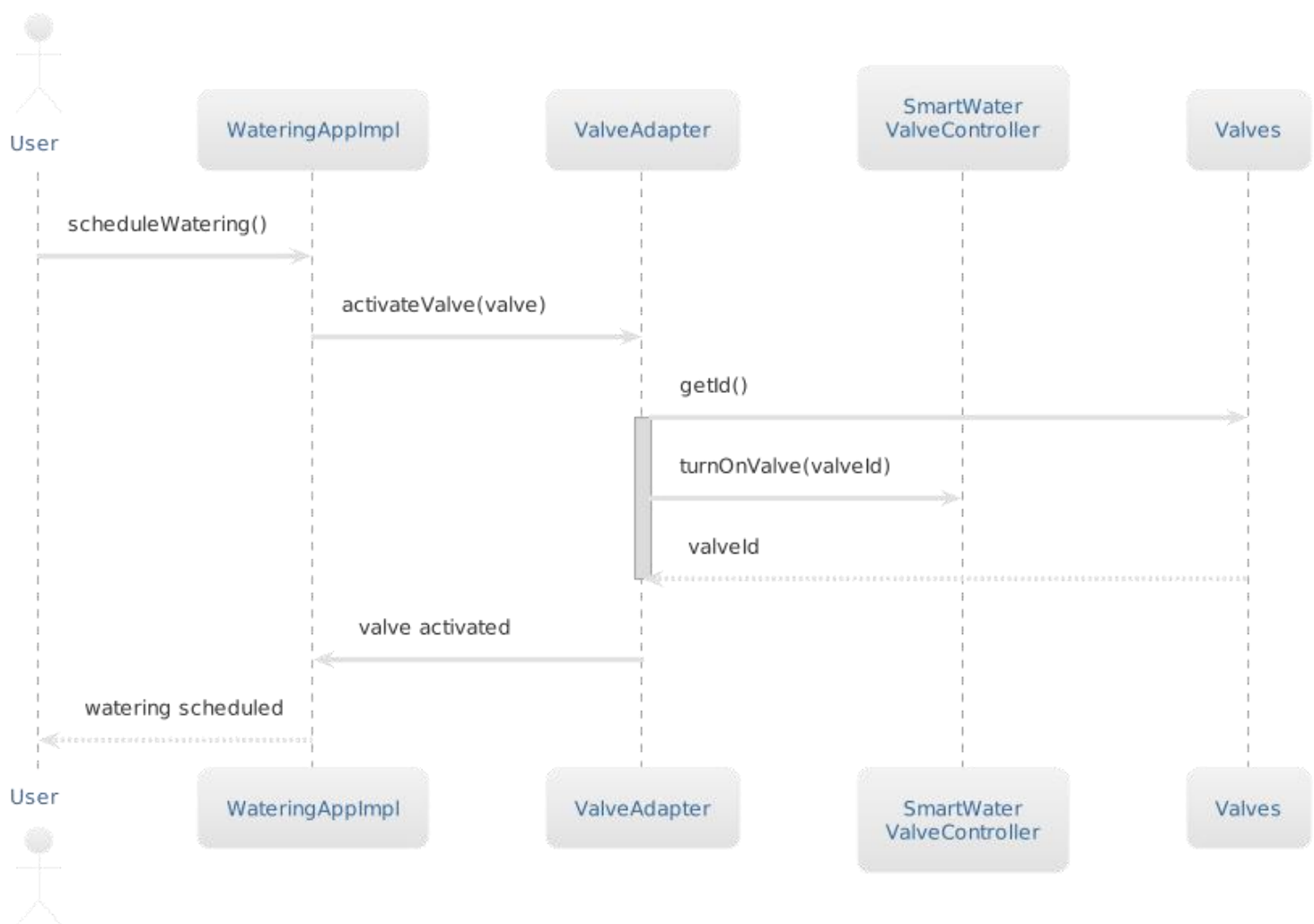
C) I have used PlantUML for creating UML diagrams

## Class Diagram





## Interaction Diagram



**D)** The adaption between the legacy garden valves controllers and the new WiFi-enabled controllers has an added performance overhead. Each time a watering schedule request comes from the WateringApp, the ValveAdapter needs to:

- Identify the appropriate valve based on valve ID
- Map the valve ID to corresponding controller
- Translate the control APIs for that controller
- Route requests & responses between new/legacy controllers

This runtime mapping, translation and routing introduces latency in the valve activation process.

During peak summer watering schedules, when there are lots of parallel requests to open valves across zones, this added latency results in slight delays in valves opening. Over time, the delays add up and impact the overall water distribution.

E)

```
// WateringApp interface
interface WateringApp {
    public void scheduleWatering();
}

// Concrete implementation of WateringApp
class WateringAppImpl implements WateringApp {

    WateringAppImpl() {
        // Constructor logic
    }

    @Override
    public void scheduleWatering() {
        // Schedule watering logic
    }

}

// Valve class
class Valve {

    private String id;

    public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return this.id;
    }

}

// Legacy water valve controller
class GardenWaterValveLegacyController {

    public void activateValve(Valve valve) {
        // Logic to activate valve
    }

    public void disableValve(Valve valve) {
        // Logic to disable valve
    }

}
```

```

// Modern smart valve controller
class SmartWaterValveController {

    public void turnOnValve(String valveId) {
        System.out.println("Turning on valve " + valveId);
    }

    public void turnOffValve(String valveId) {
        System.out.println("Turning off valve " + valveId);
    }

}

// Valve adapter
class ValveAdapter extends GardenWaterValveLegacyController {

    private SmartWaterValveController smartController;

    public ValveAdapter(SmartWaterValveController smartController2)
    {
        this.smartController = smartController2;
    }

    @Override
    public void activateValve(Valve valve) {
        // Adapt to smart controller API
        smartController.turnOnValve(valve.getId());
    }

    @Override
    public void disableValve(Valve valve) {
        // Adapt to smart controller API
        smartController.turnOffValve(valve.getId());
    }

}

class GardenWatering {

    public static void main(String[] args) {

        // Create watering app
        WateringApp wateringApp = new WateringAppImpl();

        // Create legacy valves
        Valve gardenValve1 = new Valve();
        gardenValve1.setId("valve1");

        Valve gardenValve2 = new Valve();
        gardenValve2.setId("valve2");
    }
}

```



```
// Create smart valve controller
SmartWaterValveController smartController = new
SmartWaterValveController();
```

```
// Create valve adapter
ValveAdapter adapter = new ValveAdapter(smartController);
```

```
// Schedule watering using adapter for both valve types
wateringApp.scheduleWatering();
adapter.activateValve(gardenValve1);
```

```
wateringApp.scheduleWatering();
adapter.activateValve(gardenValve2);
```

```
// Disable valves
adapter.disableValve(gardenValve1);
adapter.disableValve(gardenValve2);
}
```

```
}
```

```
GardenWatering.java U X
ProblemSolving > GardenWatering.java > SmartWaterValveController > turnOffValve(String)
1
2 // WateringApp interface
3 interface WateringApp {
4     public void scheduleWatering();
5 }
6
7 // Concrete implementation of WateringApp
8 class WateringAppImpl implements WateringApp {
9
10     WateringAppImpl() {
11         // Constructor logic
12     }
13
14     @Override
15     public void scheduleWatering() {
16         // Schedule watering logic
17     }
18
19 }
20
21 // Valve class
22 class Valve {
23
24     private String id;
25
26     public void setId(String id) {
27         this.id = id;
28     }
29
30     public String getId() {
31         return this.id;
32     }
33
34 }
35
36 // Legacy water valve controller
37 class GardenWaterValveLegacyController {
38
39     public void activateValve(Valve valve) {
40
41     }
42
43 }
44
45 }
46
47 }
48
49 }
50
51 }
52
53 }
54
55 }
56
57 }
58
59 }
60
61 }
62
63 }
64
65 }
66
67 }
68
69 }
70
71 }
72
73 }
74
75 }
76
77 }
78
79 }
80
81 }
82
83 }
84
85 }
86
87 }
88
89 }
90
91 }
92
93 }
94
95 }
96
97 }
98
99 }
100
101 }
102
103 }
104
105 }
106
107 }
108
109 }
110
111 }
112
113 }
114
115 }
116
117 }
118
119 }
120
121 }
122
123 }
124
125 }
126
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
142
143 }
144
145 }
146
147 }
148
149 }
150
151 }
152
153 }
154
155 }
156
157 }
158
159 }
160
161 }
162
163 }
164
165 }
166
167 }
168
169 }
170
171 }
172
173 }
174
175 }
176
177 }
178
179 }
180
181 }
182
183 }
184
185 }
186
187 }
188
189 }
190
191 }
192
193 }
194
195 }
196
197 }
198
199 }
200
201 }
202
203 }
204
205 }
206
207 }
208
209 }
210
211 }
212
213 }
214
215 }
216
217 }
218
219 }
220
221 }
222
223 }
224
225 }
226
227 }
228
229 }
230
231 }
232
233 }
234
235 }
236
237 }
238
239 }
240
241 }
242
243 }
244
245 }
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311 }
312
313 }
314
315 }
316
317 }
318
319 }
320
321 }
322
323 }
324
325 }
326
327 }
328
329 }
330
331 }
332
333 }
334
335 }
336
337 }
338
339 }
340
341 }
342
343 }
344
345 }
346
347 }
348
349 }
350
351 }
352
353 }
354
355 }
356
357 }
358
359 }
360
361 }
362
363 }
364
365 }
366
367 }
368
369 }
370
371 }
372
373 }
374
375 }
376
377 }
378
379 }
380
381 }
382
383 }
384
385 }
386
387 }
388
389 }
390
391 }
392
393 }
394
395 }
396
397 }
398
399 }
400
401 }
402
403 }
404
405 }
406
407 }
408
409 }
410
411 }
412
413 }
414
415 }
416
417 }
418
419 }
420
421 }
422
423 }
424
425 }
426
427 }
428
429 }
430
431 }
432
433 }
434
435 }
436
437 }
438
439 }
440
441 }
442
443 }
444
445 }
446
447 }
448
449 }
450
451 }
452
453 }
454
455 }
456
457 }
458
459 }
460
461 }
462
463 }
464
465 }
466
467 }
468
469 }
470
471 }
472
473 }
474
475 }
476
477 }
478
479 }
480
481 }
482
483 }
484
485 }
486
487 }
488
489 }
490
491 }
492
493 }
494
495 }
496
497 }
498
499 }
500
501 }
502
503 }
504
505 }
506
507 }
508
509 }
510
511 }
512
513 }
514
515 }
516
517 }
518
519 }
520
521 }
522
523 }
524
525 }
526
527 }
528
529 }
530
531 }
532
533 }
534
535 }
536
537 }
538
539 }
540
541 }
542
543 }
544
545 }
546
547 }
548
549 }
550
551 }
552
553 }
554
555 }
556
557 }
558
559 }
560
561 }
562
563 }
564
565 }
566
567 }
568
569 }
570
571 }
572
573 }
574
575 }
576
577 }
578
579 }
580
581 }
582
583 }
584
585 }
586
587 }
588
589 }
590
591 }
592
593 }
594
595 }
596
597 }
598
599 }
600
601 }
602
603 }
604
605 }
606
607 }
608
609 }
610
611 }
612
613 }
614
615 }
616
617 }
618
619 }
620
621 }
622
623 }
624
625 }
626
627 }
628
629 }
630
631 }
632
633 }
634
635 }
636
637 }
638
639 }
640
641 }
642
643 }
644
645 }
646
647 }
648
649 }
650
651 }
652
653 }
654
655 }
656
657 }
658
659 }
660
661 }
662
663 }
664
665 }
666
667 }
668
669 }
670
671 }
672
673 }
674
675 }
676
677 }
678
679 }
680
681 }
682
683 }
684
685 }
686
687 }
688
689 }
690
691 }
692
693 }
694
695 }
696
697 }
698
699 }
700
701 }
702
703 }
704
705 }
706
707 }
708
709 }
710
711 }
712
713 }
714
715 }
716
717 }
718
719 }
720
721 }
722
723 }
724
725 }
726
727 }
728
729 }
730
731 }
732
733 }
734
735 }
736
737 }
738
739 }
740
741 }
742
743 }
744
745 }
746
747 }
748
749 }
750
751 }
752
753 }
754
755 }
756
757 }
758
759 }
760
761 }
762
763 }
764
765 }
766
767 }
768
769 }
770
771 }
772
773 }
774
775 }
776
777 }
778
779 }
780
781 }
782
783 }
784
785 }
786
787 }
788
789 }
790
791 }
792
793 }
794
795 }
796
797 }
798
799 }
800
801 }
802
803 }
804
805 }
806
807 }
808
809 }
810
811 }
812
813 }
814
815 }
816
817 }
818
819 }
820
821 }
822
823 }
824
825 }
826
827 }
828
829 }
830
831 }
832
833 }
834
835 }
836
837 }
838
839 }
840
841 }
842
843 }
844
845 }
846
847 }
848
849 }
850
851 }
852
853 }
854
855 }
856
857 }
858
859 }
860
861 }
862
863 }
864
865 }
866
867 }
868
869 }
870
871 }
872
873 }
874
875 }
876
877 }
878
879 }
880
881 }
882
883 }
884
885 }
886
887 }
888
889 }
890
891 }
892
893 }
894
895 }
896
897 }
898
899 }
900
901 }
902
903 }
904
905 }
906
907 }
908
909 }
910
911 }
912
913 }
914
915 }
916
917 }
918
919 }
920
921 }
922
923 }
924
925 }
926
927 }
928
929 }
930
931 }
932
933 }
934
935 }
936
937 }
938
939 }
940
941 }
942
943 }
944
945 }
946
947 }
948
949 }
950
951 }
952
953 }
954
955 }
956
957 }
958
959 }
960
961 }
962
963 }
964
965 }
966
967 }
968
969 }
970
971 }
972
973 }
974
975 }
976
977 }
978
979 }
980
981 }
982
983 }
984
985 }
986
987 }
988
989 }
990
991 }
992
993 }
994
995 }
996
997 }
998
999 }
1000
1001 }
1002
1003 }
1004
1005 }
1006
1007 }
1008
1009 }
1010
1011 }
1012
1013 }
1014
1015 }
1016
1017 }
1018
1019 }
1020
1021 }
1022
1023 }
1024
1025 }
1026
1027 }
1028
1029 }
1030
1031 }
1032
1033 }
1034
1035 }
1036
1037 }
1038
1039 }
1040
1041 }
1042
1043 }
1044
1045 }
1046
1047 }
1048
1049 }
1050
1051 }
1052
1053 }
1054
1055 }
1056
1057 }
1058
1059 }
1060
1061 }
1062
1063 }
1064
1065 }
1066
1067 }
1068
1069 }
1070
1071 }
1072
1073 }
1074
1075 }
1076
1077 }
1078
1079 }
1080
1081 }
1082
1083 }
1084
1085 }
1086
1087 }
1088
1089 }
1090
1091 }
1092
1093 }
1094
1095 }
1096
1097 }
1098
1099 }
1100
1101 }
1102
1103 }
1104
1105 }
1106
1107 }
1108
1109 }
1110
1111 }
1112
1113 }
1114
1115 }
1116
1117 }
1118
1119 }
1120
1121 }
1122
1123 }
1124
1125 }
1126
1127 }
1128
1129 }
1130
1131 }
1132
1133 }
1134
1135 }
1136
1137 }
1138
1139 }
1140
1141 }
1142
1143 }
1144
1145 }
1146
1147 }
1148
1149 }
1150
1151 }
1152
1153 }
1154
1155 }
1156
1157 }
1158
1159 }
1160
1161 }
1162
1163 }
1164
1165 }
1166
1167 }
1168
1169 }
1170
1171 }
1172
1173 }
1174
1175 }
1176
1177 }
1178
1179 }
1180
1181 }
1182
1183 }
1184
1185 }
1186
1187 }
1188
1189 }
1190
1191 }
1192
1193 }
1194
1195 }
1196
1197 }
1198
1199 }
1200
1201 }
1202
1203 }
1204
1205 }
1206
1207 }
1208
1209 }
1210
1211 }
1212
1213 }
1214
1215 }
1216
1217 }
1218
1219 }
1220
1221 }
1222
1223 }
1224
1225 }
1226
1227 }
1228
1229 }
1230
1231 }
1232
1233 }
1234
1235 }
1236
1237 }
1238
1239 }
1240
1241 }
1242
1243 }
1244
1245 }
1246
1247 }
1248
1249 }
1250
1251 }
1252
1253 }
1254
1255 }
1256
1257 }
1258
1259 }
1260
1261 }
1262
1263 }
1264
1265 }
1266
1267 }
1268
1269 }
1270
1271 }
1272
1273 }
1274
1275 }
1276
1277 }
1278
1279 }
1280
1281 }
1282
1283 }
1284
1285 }
1286
1287 }
1288
1289 }
1290
1291 }
1292
1293 }
1294
1295 }
1296
1297 }
1298
1299 }
1300
1301 }
1302
1303 }
1304
1305 }
1306
1307 }
1308
1309 }
1310
1311 }
1312
1313 }
1314
1315 }
1316
1317 }
1318
1319 }
1320
1321 }
1322
1323 }
1324
1325 }
1326
1327 }
1328
1329 }
1330
1331 }
1332
1333 }
1334
1335 }
1336
1337 }
1338
1339 }
1340
1341 }
1342
1343 }
1344
1345 }
1346
1347 }
1348
1349 }
1350
1351 }
1352
1353 }
1354
1355 }
1356
1357 }
1358
1359 }
1360
1361 }
1362
1363 }
1364
1365 }
1366
1367 }
1368
1369 }
1370
1371 }
1372
1373 }
1374
1375 }
1376
1377 }
1378
1379 }
1380
1381 }
1382
1383 }
1384
1385 }
1386
1387 }
1388
1389 }
1390
1391 }
1392
1393 }
1394
1395 }
1396
1397 }
1398
1399 }
1400
1401 }
1402
1403 }
1404
1405 }
1406
1407 }
1408
1409 }
1410
1411 }
1412
1413 }
1414
1415 }
1416
1417 }
1418
1419 }
1420
1421 }
1422
1423 }
1424
1425 }
1426
1427 }
1428
1429 }
1430
1431 }
1432
1433 }
1434
1435 }
1436
1437 }
1438
1439 }
1440
1441 }
1442
1443 }
1444
1445 }
1446
1447 }
1448
1449 }
1450
1451 }
1452
1453 }
1454
1455 }
1456
1457 }
1458
1459 }
1460
1461 }
1462
1463 }
1464
1465 }
1466
1467 }
1468
1469 }
1470
1471 }
1472
1473 }
1474
1475 }
1476
1477 }
1478
1479 }
1480
1481 }
1482
1483 }
1484
1485 }
1486
1487 }
1488
1489 }
1490
1491 }
1492
1493 }
1494
1495 }
1496
1497 }
1498
1499 }
1500
1501 }
1502
1503 }
1504
1505 }
1506
1507 }
1508
1509 }
1510
1511 }
1512
1513 }
1514
1515 }
1516
1517 }
1518
1519 }
1520
1521 }
1522
1523 }
1524
1525 }
1526
1527 }
1528
1529 }
1530
1531 }
1532
1533 }
1534
1535 }
1536
1537 }
1538
1539 }
1540
1541 }
1542
1543 }
1544
1545 }
1546
1547 }
1548
1549 }
1550
1551 }
1552
1553 }
1554
1555 }
1556
1557 }
1558
1559 }
1560
1561 }
1562
1563 }
1564
1565 }
1566
1567 }
1568
1569 }
1570
1571 }
1572
1573 }
1574
1575 }
1576
1577 }
1578
1579 }
1580
1581 }
1582
1583 }
1584
1585 }
1586
1587 }
1588
1589 }
1590
1591 }
1592
1593 }
1594
1595 }
1596
1597 }
1598
1599 }
1600
1601 }
1602
1603 }
1604
1605 }
1606
1607 }
1608
1609 }
1610
1611 }
1612
1613 }
1614
1615 }
1616
1617 }
1618
1619 }
1620
1621 }
1622
1623 }
1624
1625 }
1626
1627 }
1628
1629 }
1630
1631 }
1632
1633 }
1634
1635 }
1636
1637 }
1638
1639 }
1640
1641 }
1642
1643 }
1644
1645 }
1646
1647 }
1648
1649 }
1650
1651 }
1652
1653 }
1654
1655 }
1656
1657 }
1658
1659 }
1660
1661 }
1662
1663 }
1664
1665 }
1666
1667 }
1668
1669 }
1670
1671 }
1672
1673 }
1674
1675 }
1676
1677 }
1678
1679 }
1680
1681 }
1682
1683 }
1684
1685 }
1686
1687 }
1688
1689 }
1690
1691 }
1692
1693 }
1694
1695 }
1696
1697 }
1698
1699 }
1700
1701 }
1702
1703 }
1704
1705 }
1706
1707 }
1708
1709 }
1710
1711 }
1712
1713 }
1714
1715 }
1716
1717 }
1718
1719 }
1720
1721 }
1722
1723 }
1724
1725 }
1726
1727 }
1728
1729 }
1730
1731 }
1732
1733 }
1734
1735 }
1736
1737 }
1738
1739 }
1740
1741 }
1742
1743 }
1744
1745 }
1746
1747 }
1748
1749 }
1750
1751 }
1752
1753 }
1754
1755 }
1756
1757 }
1758
1759 }
1760
1761 }
1762
1763 }
1764
1765 }
1766
1767 }
1768
1769 }
1770
1771 }
1772
1773 }
1774
1775 }
1776
1777 }
1778
1779 }
1780
1781 }
1782
1783 }
1784
1785 }
1786
1787 }
1788
1789 }
1790
1791 }
1792
1793 }
1794
1795 }
1796
1797 }
1798
1799 }
1800
1801 }
1802
1803 }
1804
1805 }
1806
1807 }
1808
1809 }
1810
1811 }
1812
1813 }
1814
1815 }
1816
1817 }
1818
1819 }
1820
1821 }
1822
1823 }
1824
1825 }
1826
1827 }
1828
1829 }
1830
1831 }
1832
1833 }
1834
1835 }
1836
1837 }
1838
1839 }
1840
1841 }
1842
1843 }
1844
1845 }
1846
1847 }
1848
1849 }
1850
1851 }
1852
1853 }
1854
1855 }
1856
1857 }
1858
1859 }
1860
1861 }
1862
1863 }
1864
1865 }
1866
1867 }
1868
1869 }
1870
1871 }
1872
1873 }
1874
1875 }
1876
1877 }
1878
1879 }
1880
1881 }
1882
1883 }
1884
1885 }
1886
1887 }
1888
1889 }
1890
1891 }
1892
1893 }
1894
1895 }
1896
1897 }
1898
1899 }
1900
1901 }
1902
1903 }
1904
1905 }
1906
1907 }
1908
1909 }
1910
1911 }
1912
1913 }
1914
1915 }
1916
1917 }
1918
1919 }
1920
1921 }
1922
1923 }
1924
1925 }
1926
1927 }
1928
1929 }
1930
1931 }
1932
1933 }
1934
1935 }
1936
1937 }
1938
1939 }
1940
1941 }
1942
1943 }
1944
1945 }
1946
1947 }
1948
1949 }
1950
1951 }
1952
1953 }
1954
1955 }
1956
1957 }
1958
1959 }
1960
1961 }
1962
1963 }
1964
1965 }
1966
1967 }
1968
1969 }
1970
1971 }
1972
1973 }
1974
1975 }
1976
1977 }
1978
1979 }
1980
1981 }
1982
1983 }
1984
1985 }
1986
1987 }
1988
1989 }
1990
1991 }
1992
1993 }
1994
1995 }
1996
1997 }
1998
1999 }
2000
2001 }
2002
2003 }
2004
2005 }
2006
2007 }
2008
2009 }
2010
2011 }
2012
2013 }
2014
2015 }
2016
2017 }
2018
2019 }
2020
2021 }
2022
2023 }
2024
2025 }
2026
2027 }
2028
2029 }
2030
2031 }
2032
2033 }
2034
2035 }
2036
2037 }
2038
2039 }
2040
2041 }
2042
2043 }
2044
2045 }
2046
2047 }
2048
2049 }
2050
2051 }
2052
2053 }
2054
2055 }
2056
2057 }
2058
2059 }
2060
2061 }
2062
2063 }
2064
2065 }
2066
2067 }
2068
2069 }
2070
2071 }
2072
2073 }
2074
2075 }
2076
2077 }
2078
2079 }
2080
2081 }
2082
2083 }
2084
2085 }
2086
2087 }
2088
2089 }
2090
2091 }
2092
2093 }
2094
2095 }
2096
2097 }
2098
2099 }
2100
2101 }
2102
2103 }
2104
2105 }
2106
2107 }
2108
2109 }
2110
2111 }
2112
2113 }
2114
2115 }
2116
2117 }
2118
2119 }
2120
2121 }
2122
2123 }
2124
2125 }
2126
2127 }
2128
2129 }
2130
2131 }
2132
2133 }
2134
2135 }
2136
2137 }
2138
2139 }
2140
2141 }
2142
2143 }
2144
2145 }
2146
2147 }
2148
2149 }
2150
2151 }
2152
2153 }
2154
2155 }
2156
2157 }
2158
2159 }
2160
2161 }
2162
2163 }
2164
2165 }
2166
2167 }
2168
2169 }
2170
2171 }
2172
2173 }
2174
2175 }
2176
2177 }
2178
2179 }
2180
2181 }
2182
2183 }
2184
2185 }
2186
2187 }
2188
2189 }
2190
2191 }
2192
2193 }
2194
2195 }
2196
2197 }
2198
2199 }
2200
2201 }
2202
2203 }
2204
2205 }
2206
2207 }
2208
2209 }
2210
2211 }
2212
2213 }
2214
2215 }
2216
2217 }
2218
2219 }
2220
2221 }
2222
2223 }
2224
2225 }
2226
2227 }
2228
2229 }
2230
2231 }
2232
2233 }
2234
2235 }
2236
2237 }
2238
2239 }
2240
2241 }
2242
2243 }
2244
2245 }
2246
2247 }
2248
2249 }
2250
2251 }
2252
2253 }
2254
2255 }
2256
2257 }
2258
2259 }
2260
2261 }
2262
2263 }
2264
2265 }
2266
2267 }
2268
2269 }
2270
2271 }
2272
2273 }
2274
2275 }
2276
2277 }
2278
2279 }
2280
2281 }
2282
2283 }
2284
2285 }
2286
2287 }
2288
2289 }
2290
2291 }
2292
2293 }
2294
2295 }
2296
2297 }
2298
2299 }
2300
2301 }
2302
2303 }
2304
2305 }
2306
2307 }
2308
2309 }
2310
2311 }
2312
2313 }
2314
2315 }
2316
2317 }
2318
2319 }
2320
2321 }
2322
2323 }
2324
2325 }
2326
2327 }
2328
2329 }
2330
2331 }
2332
2333 }
2334
2335 }
2336
2337 }
2338
2339 }
2340
2341 }
2342
2343 }
2344
2345 }
2346
2347 }
2348
2349 }
2350
2351 }
2352
2353 }
2354
2355 }
2356
2357 }
2358
2359 }
2360
2361 }
2362
2363 }
2364
2365 }
2
```

```
GardenWatering.java U X
ProblemSolving > GardenWatering.java > SmartWaterValveController > turnOffValve(String)
63 class ValveAdapter extends GardenWaterValveLegacyController {
78 public void disableValve(Valve valve) {
79     smartController.turnOffValve(valve.getId());
81 }
82
83 }
84
85 class GardenWatering {
86
87     Run | Debug
88     public static void main(String[] args) {
89
90         // Create watering app
91         WateringApp wateringApp = new WateringAppImpl();
92
93         // Create legacy valves
94         Valve gardenValve1 = new Valve();
95         gardenValve1.setId(id:"valve1");
96
97         Valve gardenValve2 = new Valve();
98         gardenValve2.setId(id:"valve2");
99
100        // Create smart valve controller
101        SmartWaterValveController smartController = new SmartWaterValveController();
102
103        // Create valve adapter
104        ValveAdapter adapter = new ValveAdapter(smartController);
105
106        // Schedule watering using adapter for both valve types
107        wateringApp.scheduleWatering();
108        adapter.activateValve(gardenValve1);
109
110        wateringApp.scheduleWatering();
111        adapter.activateValve(gardenValve2);
112
113        // Disable valves
114        adapter.disableValve(gardenValve1);
115        adapter.disableValve(gardenValve2);
116    }
```

## Q2

A) This scenario dealing with creating different types of lists to test an algorithm aligns very well with the Factory pattern.

The key motivation is to instantiate related but differing classes based on type without specifying concrete classes explicitly. Factory handles creating objects so client code can focus on using them.

So you could have a ListFactory that can produce ArrayList, LinkedList, or MyCustomList based on input parameter. The test algorithm code just calls factory to get required type of list, remaining isolated from actual instantiation.

B) The scenario of allowing client programs to invoke a compiler without knowing compilation steps matches very closely with Facade pattern.

Key motivations of Facade are:

- Provide a simple, unified interface hiding subsystem complexity
- Decouple client from intricate internal implementations
- Improve readability, maintainability of client code

So you could have a `CompilerFacade` that exposes clean `compile()` interfaces to client code, while internally it coordinates between parser, generators, optimizers etc. to perform full compilation without exposing client to that complexity.