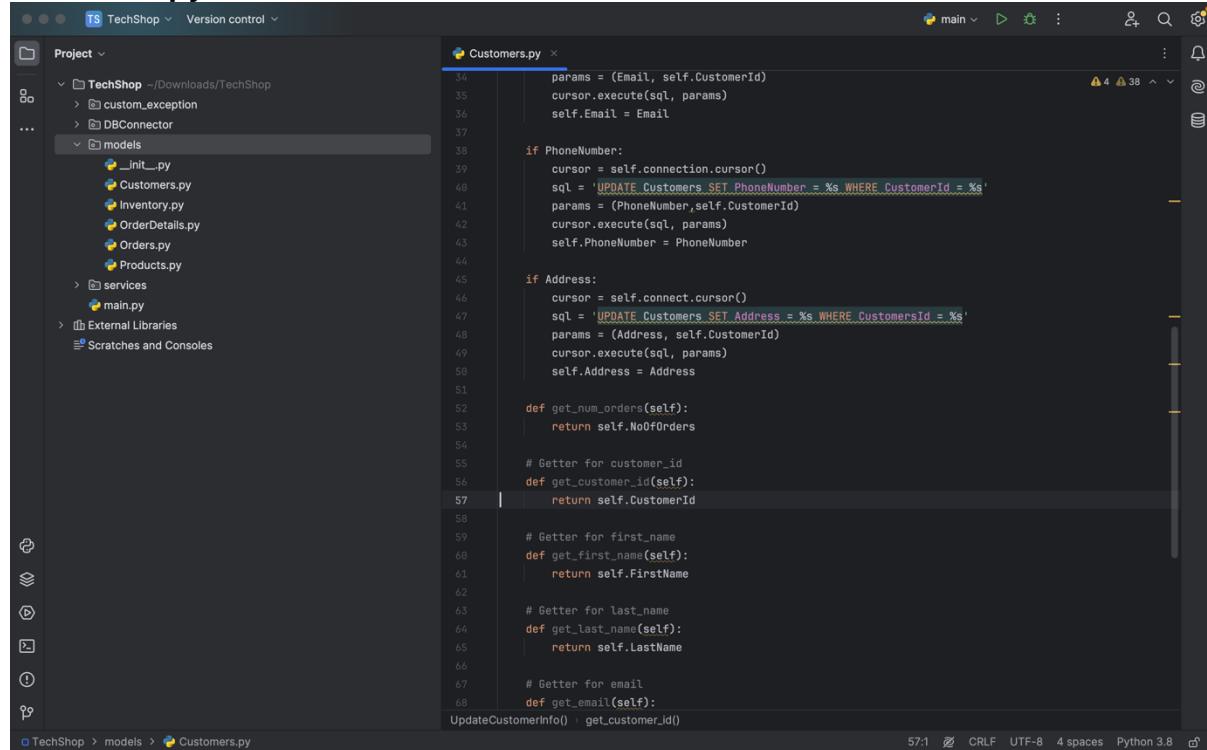


TechShop, an electronic gadgets shop

By

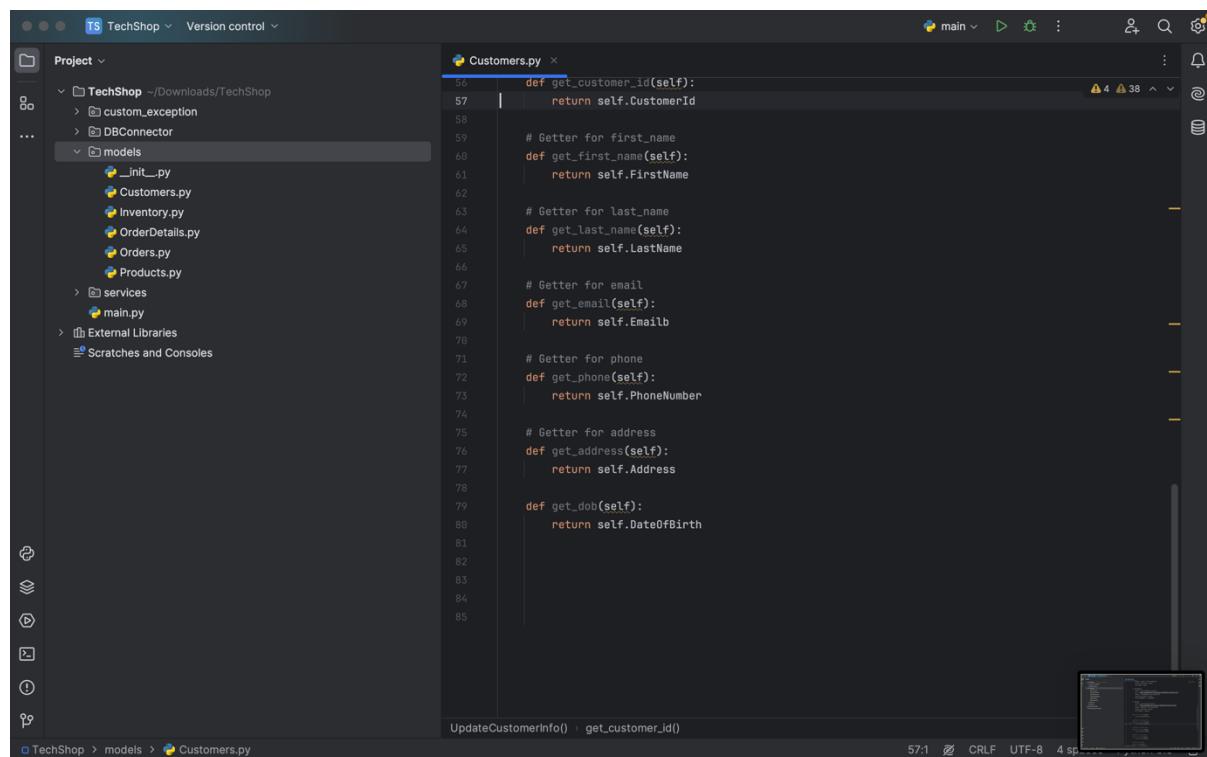
Sridhar S

Customer.py



```
34     params = (Email, self.CustomerId)
35     cursor.execute(sql, params)
36     self.Email = Email
37
38     if PhoneNumber:
39         cursor = self.connection.cursor()
40         sql = 'UPDATE Customers SET PhoneNumber = %s WHERE CustomerId = %s'
41         params = (PhoneNumber, self.CustomerId)
42         cursor.execute(sql, params)
43         self.PhoneNumber = PhoneNumber
44
45     if Address:
46         cursor = self.connection.cursor()
47         sql = 'UPDATE Customers SET Address = %s WHERE CustomerId = %s'
48         params = (Address, self.CustomerId)
49         cursor.execute(sql, params)
50         self.Address = Address
51
52     def get_num_orders(self):
53         return self.NoOfOrders
54
55     # Getter for customer_id
56     def get_customer_id(self):
57         return self.CustomerId
58
59     # Getter for first_name
60     def get_first_name(self):
61         return self.FirstName
62
63     # Getter for last_name
64     def get_last_name(self):
65         return self.LastName
66
67     # Getter for email
68     def get_email(self):
69         return self.Email
70
71     # Getter for phone
72     def get_phone(self):
73         return self.PhoneNumber
74
75     # Getter for address
76     def get_address(self):
77         return self.Address
78
79     def get_dob(self):
80         return self.DateOfBirth
81
82
83
84
85
```

UpdateCustomerInfo() > get_customer_id()



```
56     def get_customer_id(self):
57         return self.CustomerId
58
59     # Getter for first_name
60     def get_first_name(self):
61         return self.FirstName
62
63     # Getter for last_name
64     def get_last_name(self):
65         return self.LastName
66
67     # Getter for email
68     def get_email(self):
69         return self.Email
70
71     # Getter for phone
72     def get_phone(self):
73         return self.PhoneNumber
74
75     # Getter for address
76     def get_address(self):
77         return self.Address
78
79     def get_dob(self):
80         return self.DateOfBirth
81
82
83
84
85
```

UpdateCustomerInfo() > get_customer_id()

```

from DBConnector.DBConnectivity import getDBConn

class Customers:
    def __init__(self, CustomerId, FirstName, LastName, Email, DateOfBirth, PhoneNumber):
        self.connection = getDBConn()
        self.CustomerId = CustomerId
        self.FirstName = FirstName
        self.LastName = LastName
        self.Email = Email
        self.DateOfBirth = DateOfBirth
        self.PhoneNumber = PhoneNumber

    def CalculateTotalOrders(self):
        cursor = self.connection.cursor()
        sql = 'SELECT %s,COUNT(*) AS TotalOrders FROM Orders WHERE CustomerId = %s'
        params = (self.CustomerId, self.CustomerId)
        cursor.execute(sql, params)
        temp = list(cursor.fetchone())
        return temp[1]

    def getCustomerDetails(self):
        details = f"CustomerId : {self.CustomerId}"
        details += f"Name: {self.FirstName} {self.LastName}"
        details += f"Email: {self.Email}"
        details += f"DateOfBirth: {self.DateOfBirth}"
        details += f"PhoneNumber: {self.PhoneNumber}"
        return details

    def UpdateCustomerInfo(self, Email, Address, PhoneNumber):
        if Email:
            cursor = self.connection.cursor()
            sql = 'UPDATE Customers SET Email = %s WHERE CustomerId = %s'
            params = (Email, self.CustomerId)
            UpdateCustomerInfo() > get_customer_id()

5 usages

```

Products.py

```

from DBConnector.DBConnectivity import getDBConn

class Products:
    def __init__(self, ProductId, ProductName, Description, Price):
        self.connection = getDBConn()
        self.ProductId = ProductId
        self.ProductName = ProductName
        self.Description = Description
        self.Price = Price

    def get_ProductId(self):
        return self.ProductId

    def set_ProductId(self, ProductId):
        sql = 'UPDATE Products SET ProductId = %s WHERE ProductId = %s'
        para = (ProductId, self.ProductId)
        cursor = self.connection.cursor()
        cursor.execute(sql, para)
        self.ProductId = ProductId

    def get_ProductName(self):
        return self.ProductName

    def set_ProductName(self, ProductName):
        sql = 'UPDATE Products SET ProductName = %s WHERE ProductId = %s'
        para = (ProductName, self.ProductId)
        cursor = self.connection.cursor()
        cursor.execute(sql, para)
        self.ProductName = ProductName

    def get_Description(self):
        return self.Description

18 usages

```

The screenshot shows the VS Code interface with the 'TechShop' project open. The left sidebar displays the project structure under 'Project'. The main editor area shows the 'Products.py' file. The code implements methods for updating product information based on price or description, and checking if a product is in stock. The status bar at the bottom indicates the file is saved, has 15 changes, and is in Python 3.8 mode.

```
53     details += f"Product ID: {self.ProductId}\n"
54     details += f"Product Name: {self.ProductName}\n"
55     details += f"Description: {self.Description}\n"
56     details += f"Price: ${self.Price: .2f}\n"
57     return details
58
59     2 usages (1 dynamic)
60 def update_product_info(self, Price=None, Description=None):
61     if Price:
62         cursor = self.connection.cursor()
63         sql = 'UPDATE Products SET Price = %s WHERE ProductID = %s'
64         para = (Price, self.ProductId)
65         cursor.execute(sql, para)
66         self.Price = Price
67     if Description:
68         cursor = self.connection.cursor()
69         sql = 'UPDATE Products SET Description = %s WHERE ProductID = %s'
70         para = (Description, self.ProductId)
71         cursor.execute(sql, para)
72         self.Description = Description
73
74     def is_product_in_stock(self):
75         cursor = self.connection.cursor()
76         sql = 'SELECT Inventory.QuantityInStock FROM Products JOIN Inventory ON Products.ProductID = Inventory.ProductID WHERE ProductID = %s'
77         para = (self.ProductId,)
78         cursor.execute(sql, para)
79         x = list(cursor.fetchone())[0]
80         return x > 0
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
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1396
1397
1398
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1496
1497
1498
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1596
1597
1598
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1996
1997
1998
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2096
2097
2098
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2177
2178
2179
2179
2180
2181
218
```

Orders.py:

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar titled 'Project' showing the directory structure of the 'TechShop' project. The 'models' folder contains several files: __init__.py, Customers.py, Inventory.py, OrderDetails.py, Orders.py (which is currently selected), Products.py, and main.py. The right pane displays the content of the Orders.py file.

```
1 from datetime import datetime
2 from models.Customers import Customers
3 from DBConnector.DBConnectivity import getDBConn
4
5 class Orders:
6     def __init__(self, order_id, customer: Customers, order_date, total_amount, order_status="Pending"):
7         self.connection = getDBConn()
8         self.OrderId = order_id
9         self.Customer = customer
10        self.OrderDate = order_date
11        self.TotalAmount = total_amount
12        self.OrderStatus = order_status
13
14    # Getter for order_id
15    def get_order_id(self):
16        return self.OrderId
17
18    # Getter for customer
19    def get_customer(self):
20        return self.Customer
21
22    # Getter for order_date
23    def get_order_date(self):
24        return self.OrderDate
25
26    # Setter for order_date
27    def set_order_date(self, order_date):
28        cursor = self.connection.cursor()
29        sql = 'UPDATE Orders SET OrderDate = %s WHERE OrderID = %s'
30        para = (order_date, self.Orderid)
31
```

At the bottom of the editor, status bars indicate the file is 1:1, uses CRLF line endings, is in UTF-8 encoding, has 4 spaces per indentation, and is written in Python 3.8.

This screenshot shows the same code editor interface as the previous one, but the code in the Orders.py file has been modified. It includes additional methods for setting and calculating total amounts and order statuses.

```
31    cursor.execute(sql, para)
32    self.OrderDate = order_date
33
34    # Getter for total_amount
35    def get_total_amount(self):
36        return self.Totalamount
37
38    # Setter for total_amount
39    def set_total_amount(self, total_amount):
40        cursor = self.connection.cursor()
41        sql = 'UPDATE Orders SET TotalAmount = %s WHERE OrderID = %s'
42        para = (total_amount, self.Orderid)
43        cursor.execute(sql, para)
44        self.TotalAmount = total_amount
45
46    # Getter for order_status
47    def get_order_status(self):
48        return self.Orderstatus
49
50    # Setter for order_status
51    def set_order_status(self, order_status):
52        cursor = self.connection.cursor()
53        sql = 'UPDATE Orders SET OrderStatus= %s WHERE OrderID = %s'
54        para = (order_status, self.Orderid)
55        cursor.execute(sql, para)
56        self.OrderStatus = order_status
57
58    # calculate_total_amount
59    def calculate_total_amount(self):
60        cursor = self.connection.cursor()
61        sql = 'SELECT SUM(OrderDetails.Quantity*orders.TotalAmount) AS TotalAmount FROM OrderDetails JOIN
62        para = (self.OrderId)
```

At the bottom of the editor, status bars indicate the file is 1:1, uses CRLF line endings, is in UTF-8 encoding, has 4 spaces per indentation, and is written in Python 3.8.

The screenshot shows a Python IDE interface with the following details:

- Project:** TechShop
- File:** Orders.py
- Code Content:**

```
62     cursor.execute(sql, para)
63     TotalAmount = list(cursor.fetchone())[0]
64     return TotalAmount
65
66     2 usages (1 dynamic)
67     def get_order_details(self):
68         details += f"Order ID: {self.OrderId}\n"
69         details += f"Customer: {self.Customer.get_customer_details()}\n"
70         details += f"Order Date: {self.OrderDate}\n"
71         details += f"Total Amount: ${self.calculate_total_amount():.2f}\n"
72         details += f"Order Status: {self.OrderStatus}\n"
73         return details
74
75     1 usage (1 dynamic)
76     def update_order_status(self, new_status):
77         cursor = self.connection.cursor()
78         sql = 'UPDATE Orders SET OrderStatus= %s WHERE OrderID = %s'
79         para = (new_status, self.OrderId)
80         cursor.execute(sql, para)
81         self.OrderStatus = new_status
82
83     1 usage (1 dynamic)
84     def cancel_order(self):
85         cursor = self.connection.cursor()
86         sql = 'UPDATE Orders SET OrderStatus= %s WHERE OrderID = %s'
87         para = ('Cancelled', self.OrderId)
88         cursor.execute(sql, para)
89         self.OrderStatus = "Cancelled"
90         print('Order successfully cancelled')
```

- Bottom Status Bar:** 1:1, CRLF, UTF-8, 4 spaces, Python 3.8

OrderDetails.py

The screenshot shows a Python IDE interface with the following details:

- Project:** TechShop
- File:** OrderDetails.py
- Code Content:**

```
1  from datetime import datetime
2  from DBConnector.DBConnectivity import getDBConn
3  from models.Customers import Customers
4  from models.Orders import Orders
5  from models.Products import Products
6
7  class OrderDetails:
8      def __init__(self, OrderDetailId, Order: Orders, Product: Products, Quantity):
9          self.connection = getDBConn()
10         self.OrderDetailId = OrderDetailId
11         self.Order = Order
12         self.Product = Product
13         self.Quantity = Quantity
14
15     # Getter for OrderDetailId
16     def get_OrderDetailId(self):
17         return self.OrderDetailId
18
19     # Getter for order
20     def get_Order(self):
21         return self.Orders
22
23     # Getter for Product
24     def get_Product(self):
25         return self.Product
26
27     # Setter for Product
28     def set_Product(self, Product: Products):
29         cursor = self.connection.cursor()
30         sql = 'UPDATE orderdetails SET ProductID = %s WHERE OrderDetailID = %s'
31         para = (Product.get_product_id(), self.OrderDetailId)
32         cursor.execute(sql, para)
33         self.Product = Product
34
35 OrderDetails > __init__()
```

- Bottom Status Bar:** 11:43, CRLF, UTF-8, 4 spaces, Python 3.8

```

    46     sql = 'UPDATE OrderDetails SET Quantity = %s WHERE OrderDetailID = %s'
    47     para = (quantity, self.OrderDetailID)
    48     cursor.execute(sql, para)
    49     self.Quantity = quantity
    50
    51     def calculate_subtotal(self):
    52         return self.Product.get_price() * self.Quantity
    53
    54     def get_order_detail_info(self):
    55         details = f'Order Detail ID: {self.OrderDetailID}\n'
    56         details += f'Order: {self.Orders.get_order_details()}\n'
    57         details += f'Product: {self.Product.get_product_id()}\n'
    58         details += f'Quantity: {self.Quantity}\n'
    59         details += f'Subtotal: ${self.calculate_subtotal():.2f}\n'
    60
    61     def update_quantity(self, new_quantity):
    62         cursor = self.connection.cursor()
    63         sql = 'UPDATE OrderDetails SET Quantity = %s WHERE OrderDetailID = %s'
    64         para = (new_quantity, self.OrderDetailID)
    65         cursor.execute(sql, para)
    66         self.Quantity = new_quantity
    67
    68     def add_discount(self, discount_percentage):
    69         discount_factor = 1 - (discount_percentage / 100)
    70         subtotal = self.calculate_subtotal()
    71         discounted_subtotal = subtotal * discount_factor
    72         print(f'Discounted Subtotal: {discounted_subtotal}')
    73
    74

```

TechShop > models > OrderDetails.py 11:43 CRLF UTF-8 4 spaces Python 3.8

```

    62     def update_quantity(self, new_quantity):
    63         cursor = self.connection.cursor()
    64         sql = 'UPDATE OrderDetails SET Quantity = %s WHERE OrderDetailID = %s'
    65         para = (new_quantity, self.OrderDetailID)
    66         cursor.execute(sql, para)
    67         self.Quantity = new_quantity
    68
    69     def add_discount(self, discount_percentage):

```

TechShop > models > OrderDetails.py 11:43 CRLF UTF-8 4 spaces Python 3.8

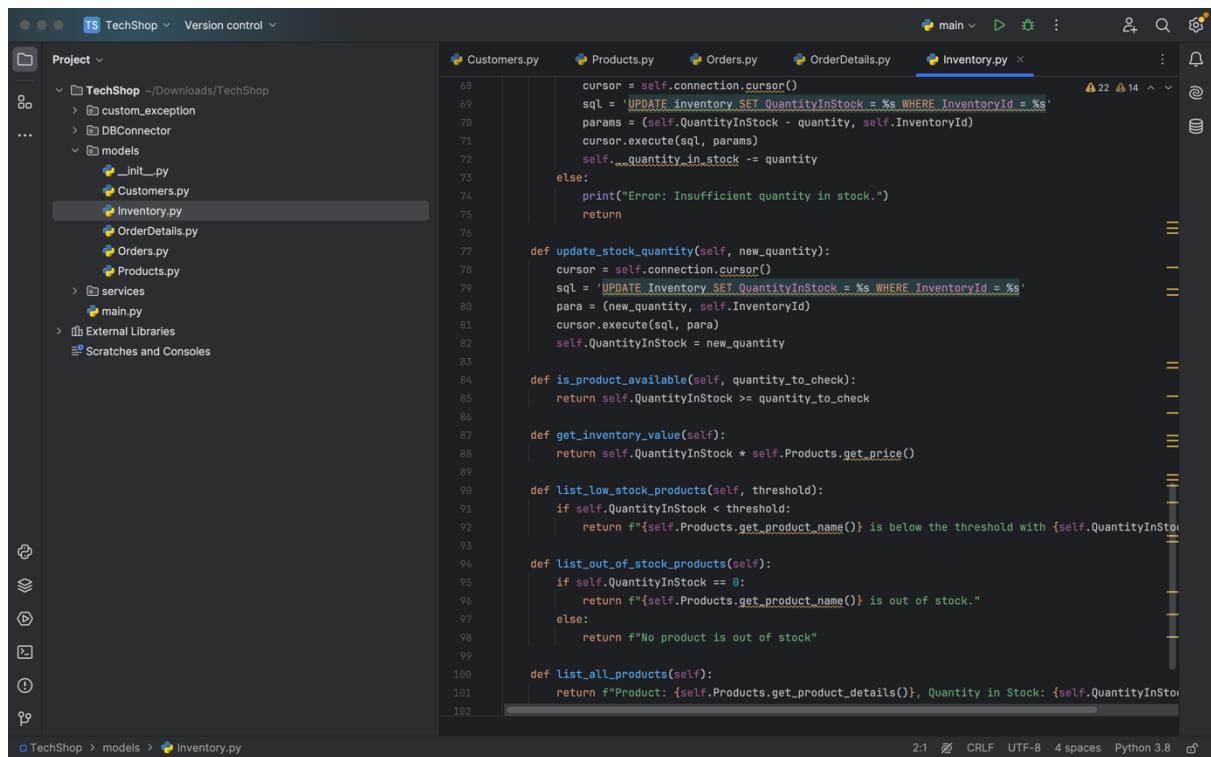
Inventory.py

```

    34     cursor = self.connection.cursor()
    35     sql = 'UPDATE Inventory SET QuantityInStock = %s WHERE InventoryId = %s'
    36     params = (QuantityInStock, self.InventoryId)
    37     cursor.execute(sql, params)
    38     self.QuantityInStock = self.QuantityInStock
    39
    40     # Getter for last_stock_update
    41     def get_last_stock_update(self):
    42         return self.LastStockUpdate
    43
    44     # Setter for last_stock_update
    45     def set_last_stock_update(self, LastStockUpdate):
    46         cursor = self.connection.cursor()
    47         sql = 'UPDATE Inventory SET LastStockUpdate = %s WHERE InventoryId = %s'
    48         para = (LastStockUpdate, self.InventoryId)
    49         cursor.execute(sql, para)
    50         self._last_stock_update = LastStockUpdate
    51
    52     def get_product(self):
    53         return self.Products
    54
    55     def get_quantity_in_stock(self):
    56         return self.QuantityInStock
    57
    58     def add_to_inventory(self, quantity):
    59         cursor = self.connection.cursor()
    60         sql = 'UPDATE Inventory SET QuantityInStock = %s WHERE InventoryId = %s'
    61         params = (self._quantity_in_stock + quantity, self.InventoryId)
    62         cursor.execute(sql, params)
    63         self.QuantityInStock += quantity
    64
    65     def remove_from_inventory(self, quantity):
    66         if self._quantity_in_stock >= quantity:
    67             cursor = self.connection.cursor()
    68             sql = 'UPDATE Inventory SET QuantityInStock = %s WHERE InventoryId = %s'

```

TechShop > models > Inventory.py 2:1 CRLF UTF-8 4 spaces Python 3.8



```
cursor = self.connection.cursor()
sql = 'UPDATE inventory SET QuantityInStock = %s WHERE InventoryId = %s'
params = (self.QuantityInStock - quantity, self.InventoryId)
cursor.execute(sql, params)
self._quantity_in_stock -= quantity
else:
    print("Error: Insufficient quantity in stock.")
    return

def update_stock_quantity(self, new_quantity):
    cursor = self.connection.cursor()
    sql = 'UPDATE inventory SET QuantityInStock = %s WHERE InventoryId = %s'
    para = (new_quantity, self.InventoryId)
    cursor.execute(sql, para)
    self.QuantityInStock = new_quantity

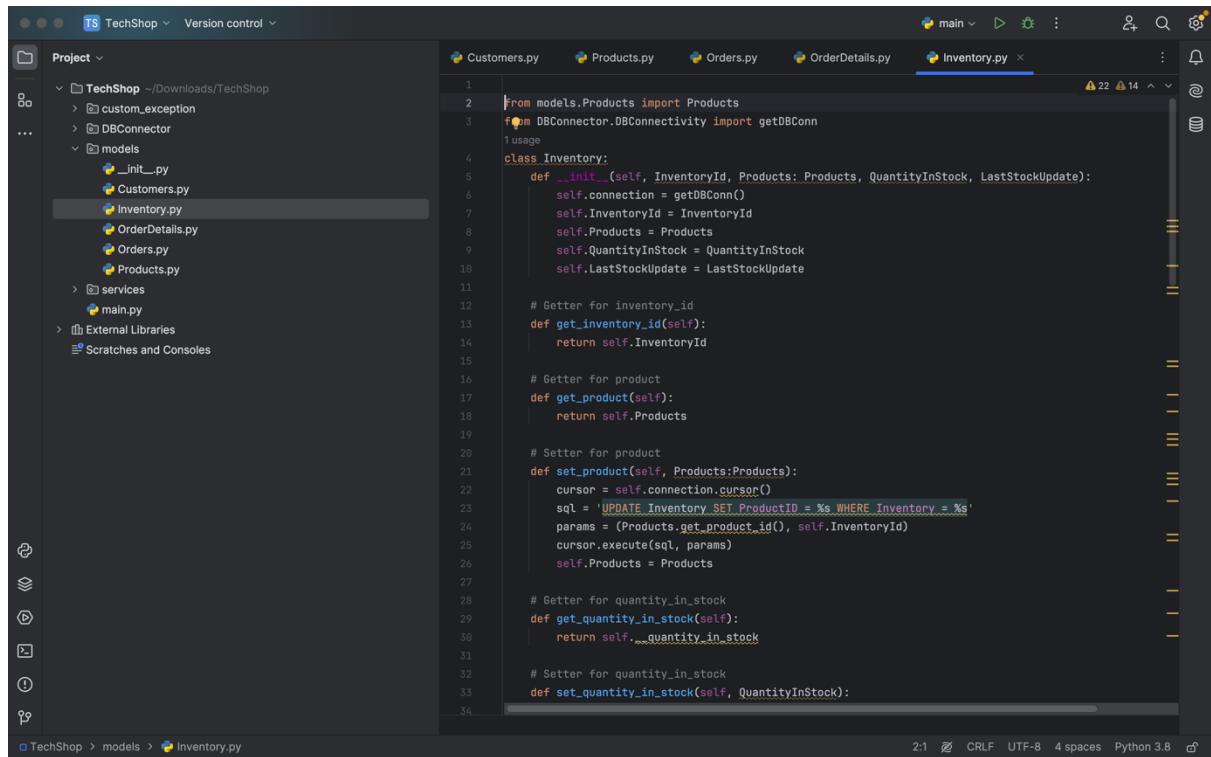
def is_product_available(self, quantity_to_check):
    return self.QuantityInStock >= quantity_to_check

def get_inventory_value(self):
    return self.QuantityInStock * self.Products.get_price()

def list_low_stock_products(self, threshold):
    if self.QuantityInStock < threshold:
        return f'{self.Products.get_product_name()} is below the threshold with {self.QuantityInStock} units in stock.'
    else:
        return f'No products are below the threshold of {threshold} units in stock.'

def list_out_of_stock_products(self):
    if self.QuantityInStock == 0:
        return f'{self.Products.get_product_name()} is out of stock.'
    else:
        return f'No products are out of stock'

def list_all_products(self):
    return f'Product: {self.Products.get_product_details()}, Quantity in Stock: {self.QuantityInStock}'
```



```
from models.Products import Products
from DBConnector.DBConnectivity import getDBConn
usage

class Inventory:
    def __init__(self, InventoryId, Products: Products, QuantityInStock, LastStockUpdate):
        self.connection = getDBConn()
        self.InventoryId = InventoryId
        self.Products = Products
        self.QuantityInStock = QuantityInStock
        self.LastStockUpdate = LastStockUpdate

    # Getter for inventory_id
    def get_inventory_id(self):
        return self.InventoryId

    # Getter for product
    def get_product(self):
        return self.Products

    # Setter for product
    def set_product(self, Products:Products):
        cursor = self.connection.cursor()
        sql = 'UPDATE inventory SET ProductID = %s WHERE InventoryId = %s'
        params = (Products.get_product_id(), self.InventoryId)
        cursor.execute(sql, params)
        self.Products = Products

    # Getter for quantity_in_stock
    def get_quantity_in_stock(self):
        return self._quantity_in_stock

    # Setter for quantity_in_stock
    def set_quantity_in_stock(self, QuantityInStock):
```

DbConnectivity.py

The screenshot shows a Python code editor interface with a dark theme. On the left is a project navigation sidebar titled 'Project' showing the directory structure of a project named 'TechShop'. The 'DBConnector' folder contains files like __init__.py and DBConnectivity.py, which is currently selected. The main editor area displays the contents of DBConnectivity.py. The code defines a class getDBConn with a static method getDBConn that connects to a SQL Server database using pyodbc. It includes example usage code demonstrating how to use the connection object.

```
2 usages
class getDBConn:
    1 usage
    @staticmethod
    def getDBConn():
        server = r'LAPTOP-J4AMUSB\SQLEXPRESS'
        database = 'TechShop'

        conn_str = f'DRIVER={{SQL Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'

        try:
            conn = pyodbc.connect(conn_str)
            return conn
        except Exception as e:
            print(f"Error connecting: {str(e)}")

    # Example usage:
    conn = getDBConn.getDBConn()
    if conn:
        print("Connection established successfully.")
        cursor = conn.cursor()

        cursor.execute("SELECT * FROM Customers")
        rows = cursor.fetchall()
        for row in rows:
            print(row)
        # Use the connection object for further operations
    else:
        print("Failed to establish connection.")
```

CustomException.py

The screenshot shows a Python code editor interface with a dark theme. On the left is a project navigation sidebar titled 'Project' showing the directory structure of a project named 'TechShop'. The 'custom_exception' folder contains files like __init__.py and CustomException.py, which is currently selected. The main editor area displays the contents of CustomException.py. The code defines three exception classes: CustomerNotFoundException, ProductNotFoundException, and OrderNotFoundException, all derived from the built-in Exception class.

```
3 usages
class CustomerNotFoundException(Exception):
    pass

3 usages
class ProductNotFoundException(Exception):
    pass

3 usages
class OrderNotFoundException(Exception):
    pass
```

CustomerManager.py

The screenshot shows a code editor interface with a project navigation sidebar on the left and the code editor area on the right. The project sidebar lists the following structure:

- TechShop
- custom_exception
- DBConnector
- models
- services
- External Libraries
- Scratches and Consoles

The code editor area displays the `CustomerManager.py` file. The code handles email validation and customer retrieval from a database. It includes methods for validating email addresses and retrieving customers by ID.

```
32     regex = r'^[\\w+\\.\\w+\\-\\w+\\.\\w+]+@[\\w+\\-\\w+\\-\\w+\\-\\w+\\-\\w+\\.\\w+\\-\\w+\\-\\w+\\.\\w+\\-\\w+\\.\\w+]$'
33     if not re.fullmatch(regex, Email):
34         raise ValueError("Invalid Email Address.")
35
36     1 usage
37     def is_email_duplicate(self, Email):
38         cursor = self.connection.cursor()
39         sql = 'SELECT * FROM Customers WHERE Email = %s'
40         para = (Email,)
41         cursor.execute(sql, para)
42         x = len(list(cursor.fetchall()))
43         if (x > 0):
44             return True
45         return False
46
47     3 usages (1 dynamic)
48     def get_customer_by_id(self, customer_id):
49         try:
50             my_cursor = self.connection.cursor()
51             sql = '''SELECT * FROM Customers WHERE customer_id = %s'''
52             para = (customer_id,)
53             my_cursor.execute(sql, para)
54             x = my_cursor.fetchone()
55             if x is None:
56                 raise CustomerNotFoundException('Invalid Customer ID')
57             else:
58                 return Customers(*x)
59         except CustomerNotFoundException as cnfe:
60             print('An error occurred ', cnfe)
61         except Exception as e:
62             print('An error occurred ', e)
```

The screenshot shows a code editor interface with a project navigation sidebar on the left and the code editor area on the right. The project sidebar lists the same structure as the first screenshot.

The code editor area displays the `CustomerManager.py` file. This version of the code adds functionality for registering new customers. It includes validation for customer data and handling of database operations like inserting new records.

```
1 import re
2 from custom_exception.CustomException import CustomerNotFoundException
3 from models.Customers import Customers
4 from DBConnector.DBConnectivity import getDBConn
5
6 class CustomerManager:
7     def __init__(self):
8         self.connection = getDBConn()
9
10    1 usage
11    def register_customer(self, FirstName, LastName, Email, DateOfBirth, Phone, NumOrders, Address):
12        try:
13            # Validating input data
14            self.validate_customer_data(Email)
15            # Checking if the Email already exists in the database
16            if self.is_email_duplicate(Email):
17                raise ValueError("Email Address is already registered.")
18
19            cursor = self.connection.cursor()
20            sql = 'SELECT * FROM Customers'
21            cursor.execute(sql)
22            x = list(cursor.fetchall())
23            sql2 = 'INSERT INTO Customers(CustomerId, FirstName, LastName, Email, DateOfBirth, Phone, NumOrders, Address)'
24            para = (len(x)+1, FirstName, LastName, Email, DateOfBirth, Phone, NumOrders, Address)
25            cursor.execute(sql2, para)
26            self.connection.commit()
27            self.connection.close()
28            print("Customer registration successful.")
29        except Exception as e:
30            print(f"Error registering customer: {e}")
31
32    1 usage
33    def validate_customer_data(self, Email):
```

InvenotryManager.py

The screenshot shows a code editor interface with a project navigation sidebar on the left and the code editor area on the right. The project sidebar lists several packages and files under 'TechShop' and 'services'. The current file is 'InventoryManager.py'. The code in the editor is as follows:

```
32     sql = 'UPDATE Inventory SET QuantityInStock = %s WHERE ProductID = %s'
33     para = (Quantity, ProductID)
34     mycursor.execute(sql, para)
35     self.connection.commit()
36 
37     else:
38         print(f"Error updating inventory: Product ID {ProductID} not found")
39     except Exception as e:
40         print(f"Error updating inventory: {e}")
41 
42     def get_inventory_value(self):
43         mycursor = self.connection.cursor()
44         sql = '''SELECT SUM(Inventory.QuantityInStock*Products.Price)
45                 AS TotalInventoryValue FROM Inventory
46                 JOIN Products ON Inventory.ProductID = Products.ProductID'''
47         mycursor.execute(sql)
48         x = list(mycursor.fetchone())[0]
49         return x
50 
51     1 usage
52     def list_low_stock_products(self, threshold):
53         mycursor = self.connection.cursor()
54         sql = '''SELECT Products.*, Inventory.QuantityInStock FROM Inventory
55                 JOIN Products ON Inventory.ProductID = Products.ProductID
56                 WHERE Inventory.QuantityInStock < %s'''
57         para = (threshold,)
58         mycursor.execute(sql, para)
59         print("All Low Stock Products in Inventory:")
60         print("ProductID | ProductName | Description | Price | QuantityInStock \n")
61         for item in mycursor.fetchall():
62             print(item)
63 
64     1 usage
65     def list_out_of_stock_products(self):
66         mycursor = self.connection.cursor()
```

The screenshot shows a code editor interface with a project navigation sidebar on the left and the code editor area on the right. The project sidebar lists several packages and files under 'TechShop' and 'services'. The current file is 'InventoryManager.py'. The code in the editor is as follows:

```
62     def list_out_of_stock_products(self):
63         mycursor = self.connection.cursor()
64         sql = '''SELECT Products.*, Inventory.QuantityInStock FROM Inventory JOIN Products ON Inventory.ProductID = Products.ProductID
65                 WHERE Inventory.QuantityInStock=0'''
66         mycursor.execute(sql)
67         print("All Out of Stock Products in Inventory:")
68         print("ProductID | ProductName | Description | Price | QuantityInStock \n")
69         for item in mycursor.fetchall():
70             print(item)
71 
72     2 usages
73     def list_all_products(self):
74         mycursor = self.connection.cursor()
75         sql = '''SELECT Products.*, Inventory.QuantityInStock FROM Inventory
76                 JOIN Products ON Inventory.ProductID = Products.ProductID'''
77         mycursor.execute(sql)
78         print("All Products in Inventory:")
79         print("ProductID | ProductName | Description | Price | QuantityInStock \n")
80         for item in mycursor.fetchall():
81             print(item)
82 
83     def get_product_by_id(self, ProductID) -> Products:
84         mycursor = self.connection.cursor()
85         sql = 'SELECT * FROM Products WHERE ProductID = %s'
86         para = (ProductID,)
87         mycursor.execute(sql, para)
88         x = list(mycursor.fetchone())
89         return Products(x[0], x[1], x[2], x[3])
```

TechShop Version control

Project

- TechShop ~/Downloads/TechShop
 - custom_exception
 - __init__.py
 - CustomException.py
 - DBConnector
 - models
 - __init__.py
 - Customers.py
 - Inventory.py
 - OrderDetails.py
 - Orders.py
 - Products.py
 - services
 - __init__.py
 - CustomerManager.py
 - InventoryManager.py
 - OrderManager.py
 - ProductManager.py
 - main.py
- External Libraries
- Scratches and Consoles

InventoryManager.py

```
1 from models.Inventory import Inventory
2 from models.Products import Products
3 from services.ProductManager import ProductManager
4 from DBConnector.DBConnectivity import getDBConn
5
6 class InventoryManager:
7     def __init__(self):
8         self.connection = getDBConn()
9
10    2 usages
11    def add_to_inventory(self, ProductId, Quantity):
12        try:
13            mycursor = self.connection.cursor()
14            p1 = ProductManager()
15            if p1.product_exists(ProductId):
16                sql = "UPDATE Inventory SET QuantityInStock = %s WHERE ProductID = %s"
17                para = (Quantity, ProductId)
18                mycursor.execute(sql, para)
19                self.connection.commit()
20            else:
21                print(f'Invalid ProductID')
22                return
23
24                print(f"Inventory updated successfully. Product ID: {ProductId}, Quantity: {Quantity}")
25            except Exception as e:
26                print(f"Error updating inventory: {e}")
27
28    1 usage
29    def remove_from_inventory(self, ProductId, Quantity):
30        try:
31            mycursor = self.connection.cursor()
32            p1 = ProductManager()
33            if p1.product_exists(ProductId):
```

1:1 CRLF UTF-8 4 spaces Python 3.8

OrderManager.py

TechShop Version control

Project

- TechShop ~/Downloads/TechShop
 - custom_exception
 - __init__.py
 - CustomException.py
 - DBConnector
 - models
 - __init__.py
 - Customers.py
 - Inventory.py
 - OrderDetails.py
 - Orders.py
 - Products.py
 - services
 - __init__.py
 - CustomerManager.py
 - InventoryManager.py
 - OrderManager.py
 - ProductManager.py
 - main.py
- External Libraries
- Scratches and Consoles

OrderManager.py

```
58     sql = 'SELECT * FROM Orders ORDER BY OrderDate ASC'
59     mycursor.execute(sql)
60     t = list(mycursor.fetchall())
61     x = []
62     for i in t:
63         x.append(list(i))
64     if not ascending:
65         x.reverse()
66     return x
67
68 def list_all_orders(self):
69     return getAllOrders()
70
71     3 usages
72     def get_order_by_id(self, order_id):
73         try:
74             my_cursor = self.connection.cursor()
75             sql = '''SELECT * FROM Orders WHERE orderID = %s'''
76             para = (order_id,)
77             my_cursor.execute(sql, para)
78             x = my_cursor.fetchone()
79             if x is None:
80                 raise OrderNotFoundException('Invalid Order ID')
81             else:
82                 return Orders(*x)
83         except OrderNotFoundException as onfe:
84             print('An error occurred ',onfe)
85         except Exception as e:
86             print('An error occurred ',e)
```

OrderManager > get_order_by_id() > try

73:41 CRLF UTF-8 4 spaces Python 3.8

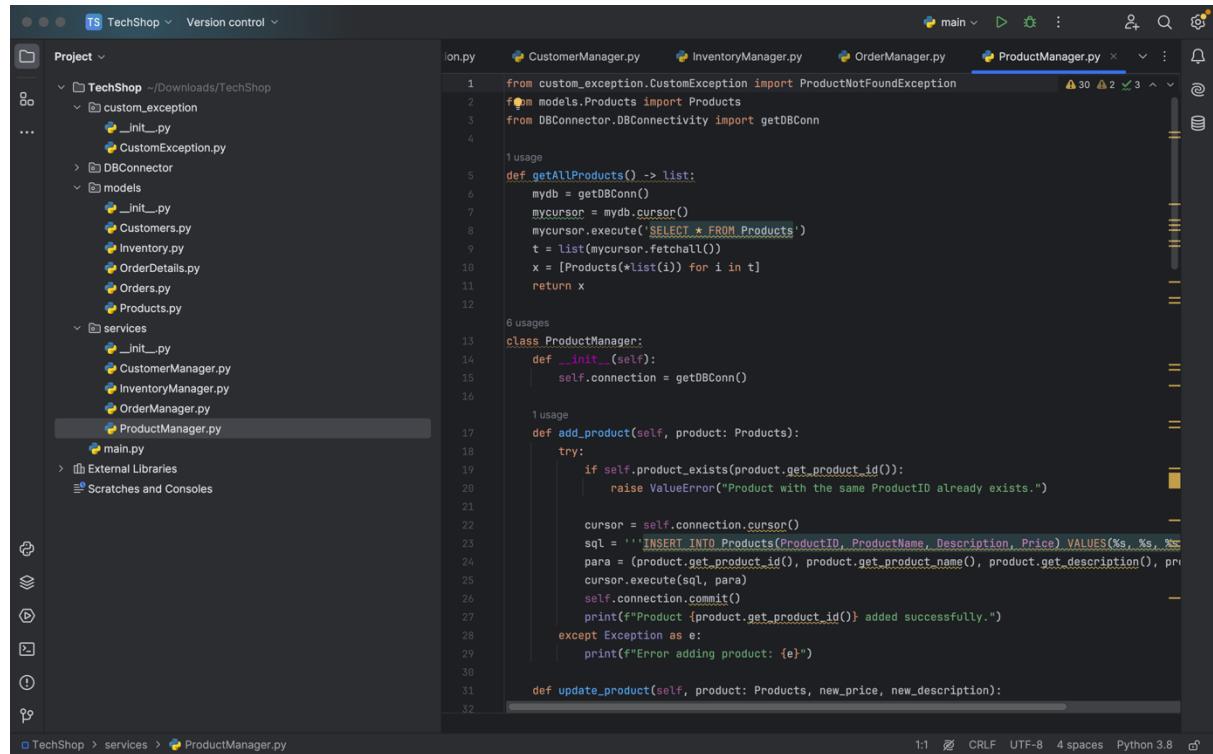
```
1 usage (1 dynamic)
def update_order_status(self, order_id, new_status):
    try:
        mycursor = self.connection.cursor()
        sql = 'UPDATE Order SET OrderStatus = %s WHERE OrderID = %s'
        para = (new_status, order_id)
        mycursor.execute(sql, para)
        self.connection.commit()
    except Exception as e:
        print(f"Error updating order status: {e}")

def remove_canceled_orders(self):
    try:
        mycursor = self.connection.cursor()
        sql1 = '''DELETE FROM OrderDetails WHERE OrderID IN
                  (SELECT OrderID FROM Orders WHERE OrderStatus IN ('Cancelled', 'cancelled'))'''
        sql2 = '''DELETE FROM Orders WHERE OrderStatus IN ('Cancelled', 'cancelled')'''
        mycursor.execute(sql1)
        mycursor.execute(sql2)
        print("Orders removed successfully")
    except Exception as e:
        print(f"Error removing canceled orders: {e}")

def sort_orders_by_date(self, ascending=True):
    mydb = getDBConn()
    mycursor = mydb.cursor()
    sql = 'SELECT * FROM Orders ORDER BY OrderDate ASC'
    mycursor.execute(sql)
    t = list(mycursor.fetchall())
    x = []
    for i in t:
        x.append(list(i))
    if not ascending:
        x.reverse()
    return x
```

```
1 from datetime import datetime
2
3 from custom_exception.CustomException import OrderNotFoundException
4 from models.Orders import Orders
5 from models.Products import Products
6 from DBConnector.DBConnectivity import getDBConn
7
8 1 usage
9 def getAllOrders() -> list:
10     mydb = getDBConn()
11     mycursor = mydb.cursor()
12     sql = 'SELECT * FROM Orders'
13     mycursor.execute(sql)
14     t = list(mycursor.fetchall())
15     x = []
16     for i in t: x.append(list(i))
17     return x
18
19 2 usages
20 class OrderManager:
21     def __init__(self):
22         self.connection = getDBConn()
23
24     1 usage
25     def add_order(self, order: Orders):
26         try:
27             mycursor = self.connection.cursor()
28             sql = 'INSERT INTO Orders(OrderID, CustomerID, OrderDate, TotalAmount, OrderStatus) VALUES (%s, %s, %s, %s, %s)'
29             para = (order.get_order_id(), order.get_customer().get_customer_id(), order.get_order_date(),
30                     order.get_total_amount(), order.get_order_status())
31             mycursor.execute(sql, para)
32             self.connection.commit()
33             print(f"Order {order.get_order_id()} added successfully.")
34         except Exception as e:
35             print(f"Error adding order: {e}")
```

ProductManager.py



VS Code interface showing the ProductManager.py file in the TechShop project. The code implements a ProductManager class with methods for adding, updating, and removing products from a database.

```
from custom_exception.CustomException import ProductNotFoundException
from models.Products import Products
from DBConnector.DBConnectivity import getDBConn

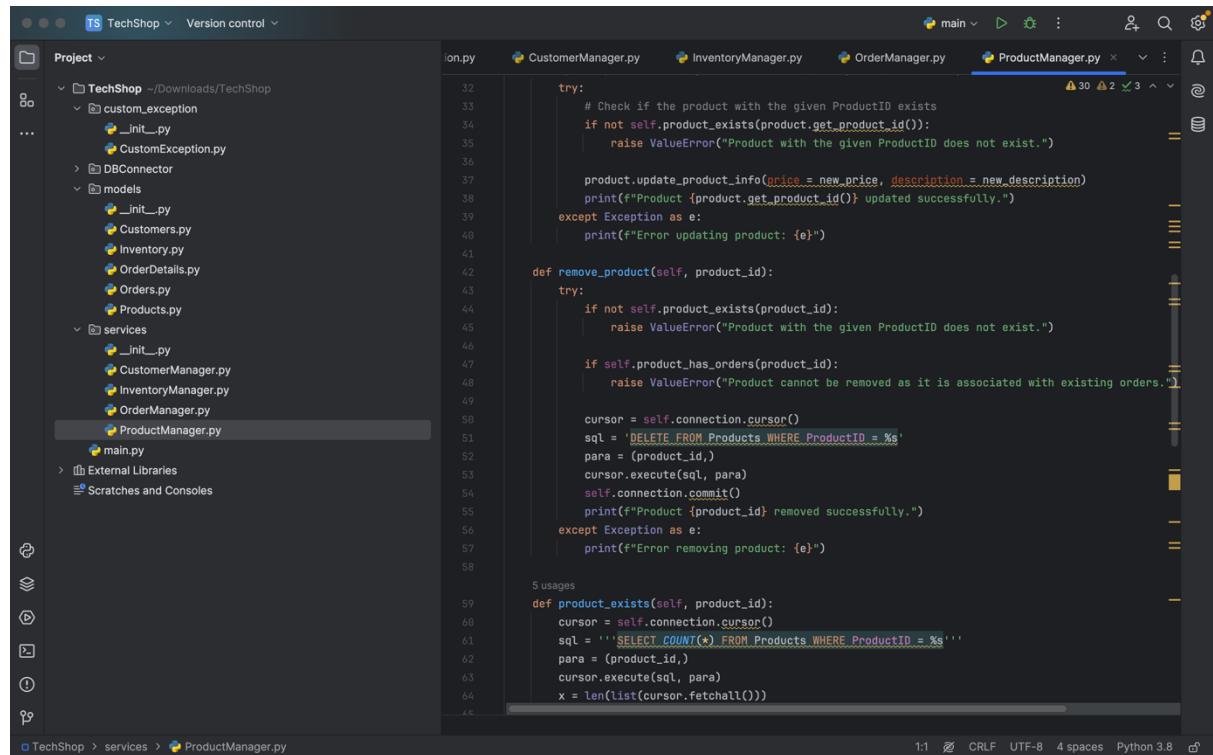
usage
def getAllProducts() -> list:
    mydb = getDBConn()
    mycursor = mydb.cursor()
    mycursor.execute('SELECT * FROM Products')
    t = list(mycursor.fetchall())
    x = [Products(*list(i)) for i in t]
    return x

6 usages
class ProductManager:
    def __init__(self):
        self.connection = getDBConn()

    1 usage
    def add_product(self, product: Products):
        try:
            if self.product_exists(product.get_product_id()):
                raise ValueError("Product with the same ProductID already exists.")

            cursor = self.connection.cursor()
            sql = '''INSERT INTO Products(ProductID, ProductName, Description, Price) VALUES(%s, %s, %s, %s)'''
            para = (product.get_product_id(), product.get_product_name(), product.get_description(), product.get_price())
            cursor.execute(sql, para)
            self.connection.commit()
            print(f"Product {product.get_product_id()} added successfully.")
        except Exception as e:
            print(f"Error adding product: {e}")

    def update_product(self, product: Products, new_price, new_description):
        try:
```



Continuation of the ProductManager.py file in the TechShop project. It includes methods for updating product info and removing products, along with a check for product existence.

```
        # Check if the product with the given ProductID exists
        if not self.product_exists(product.get_product_id()):
            raise ValueError("Product with the given ProductID does not exist.")

        product.update_product_info(price = new_price, description = new_description)
        print(f"Product {product.get_product_id()} updated successfully.")
    except Exception as e:
        print(f"Error updating product: {e}")

    def remove_product(self, product_id):
        try:
            if not self.product_exists(product_id):
                raise ValueError("Product with the given ProductID does not exist.")

            if self.product_has_orders(product_id):
                raise ValueError("Product cannot be removed as it is associated with existing orders.")

            cursor = self.connection.cursor()
            sql = 'DELETE FROM Products WHERE ProductID = %s'
            para = (product_id,)
            cursor.execute(sql, para)
            self.connection.commit()
            print(f"Product {product_id} removed successfully.")
        except Exception as e:
            print(f"Error removing product: {e}")

    def product_exists(self, product_id):
        cursor = self.connection.cursor()
        sql = '''SELECT COUNT(*) FROM Products WHERE ProductID = %s'''
        para = (product_id,)
        cursor.execute(sql, para)
        x = len(list(cursor.fetchall()))
```

The screenshot shows a code editor interface with a dark theme. On the left is a project navigation pane titled 'Project' showing a directory structure for 'TechShop'. The main area displays the 'ProductManager.py' file. The code implements a class with methods for searching products by keyword and getting a product by its ID. It uses a cursor to execute SQL queries.

```
79     return getAllProducts()
80
81
82     def search_product(self, search_keyword):
83         mycursor = self.connection.cursor()
84         sql = '''SELECT * FROM Products WHERE UPPER(ProductName) LIKE UPPER(%s) OR UPPER(Description) LIKE %s'''
85         para = ('%' + search_keyword + '%', '%' + search_keyword + '%')
86         mycursor.execute(sql, para)
87         t = list(mycursor.fetchall())
88         x = [list(i) for i in t]
89         return x
90
91     2 usages
92     def get_product_by_id(self, product_id):
93         try:
94             my_cursor = self.connection.cursor()
95             sql = ...
96             SELECT * FROM Products WHERE ProductID = %s
97             ...
98             para = (product_id,)
99             my_cursor.execute(sql, para)
100            x = my_cursor.fetchone()
101            if x is None:
102                raise ProductNotFoundException('Invalid product ID')
103            else:
104                return Products(*x)
105        except ProductNotFoundException as pnf:
106            print('An error occurred: ', pnf)
107        except Exception as e:
108            print('An error occurred: ', e)
```

This screenshot shows the same code editor interface, but the code block is shifted down by one line. The 'ProductManager.py' file is still open, and the code remains identical to the first screenshot, implementing a class with methods for searching products by keyword and getting a product by its ID. The code uses a cursor to execute SQL queries.

```
65     return x > 0
66
67     1 usage
68     def product_has_orders(self, product_id):
69         cursor = self.connection.cursor()
70         sql = '''SELECT COUNT(OrderDetails.Order_Number) AS Order_Number FROM Products
71             JOIN OrderDetails
72             ON Products.ProductID = OrderDetails.ProductID
73             WHERE products.ProductID = %s GROUP BY products.ProductID'''
```

Main.py

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** TechShop
- Version control:** main
- Files:**
 - customerManager.py:** Imports Orders, Products, CustomerManager, ProductManager, OrderManager, and InventoryManager. Contains a main menu function with options 1 through 5.
 - main.py:** The currently selected file, containing three nested menu functions: customer management menu, product management menu, and orders management menu, each with four options.
 - InventoryManager.py, OrderManager.py, ProductManager.py:** Also part of the TechShop project.
- External Libraries:** Not present in the screenshot.
- Scratches and Consoles:** Not present in the screenshot.

The screenshot shows a Python development environment with the following details:

- Project:** TechShop
- Version control:** main
- Code Editor:** The main pane displays the `main.py` file, which imports modules from `customerManager`, `InventoryManager`, `OrderManager`, and `ProductManager`. It defines a menu for inventory management and a nested loop for customer management.

```
customerManager.py    InventoryManager.py    OrderManager.py    ProductManager.py    main.py
32     print("1. Place Orders")
33     print("2. View Orders Details")
34     print("3. Update Orders Status")
35     print("4. Cancel Orders")
36     print("5. Back to Main Menu")
37
38     1 usage
39     def inventory_management_menu():
40         print("\nInventory Management Menu:")
41         print("1. Add to Inventory")
42         print("2. Remove from Inventory")
43         print("3. View Inventory Details")
44         print("4. List Low Stock Products")
45         print("5. List Out of Stock Products")
46         print("6. List All Products in Inventory")
47         print("7. Back to Main Menu")
48
49     if __name__ == "__main__":
50         CustomerManager = CustomerManager()
51         product_manager = ProductManager()
52         order_manager = OrderManager()
53         inventory_manager = InventoryManager()
54
55         while True:
56             main_menu()
57             choice = input("Enter your choice (1-5): ")
58
59             if choice == "1":
60                 while True:
61                     customer_management_menu()
62                     customer_choice = input("Enter your choice (1-4): ")
63
64                     if customer_choice == "1":
```

- Left Sidebar:** Shows the project structure with folders like `TechShop`, `custom_exception`, `DBConnector`, `models`, `services`, and files like `main.py`.
- Bottom Status Bar:** Displays the file name `main.py`, line number `1:1`, encoding `CRLF`, character set `UTF-8`, and code style settings `4 spaces` and `Python 3.8`.

The screenshot shows a Python IDE interface with the following details:

- Project:** TechShop
- File:** main.py
- Code Content:**

```
62         if customer_choice == "1":
63             CustomerManager.register_customer(
64                 input("Enter First Name: "),
65                 input("Enter Last Name: "),
66                 input("Enter Email: "),
67                 input("Enter DOB: "),
68                 input("Enter Phone: "),
69                 NumOrders=0,
70                 input("Enter Address: ")
71             )
72         elif customer_choice == "2":
73             temp_customer = CustomerManager.get_customer_by_id(input('Enter Customer ID: '))
74             print(temp_customer.get_customer_details())
75         elif customer_choice == "3":
76             temp_customer = CustomerManager.get_customer_by_id(input('Enter Customer ID: '))
77             temp_customer.update_customer_info(email=input('Enter new email'),
78                                                 phone=input('Enter new phone'),
79                                                 address=input('Enter new address'))
80
81         elif customer_choice == "4":
82             break
83         else:
84             print("Invalid choice. Please enter a number between 1 and 4.")
85
86     elif choice == "2":
87         while True:
88             product_management_menu()
89             product_choice = input("Enter your choice (1-4): ")
90
91             if product_choice == "1":
92                 product_manager.add_product(
93                     Products(input("Enter Product ID: "),
94                             input("Enter Product Name: "),
95                             input("Enter Description: "),
96                             float(input("Enter Price: ")))
97             elif product_choice == "2":
98                 temp_product = ProductManager.get_product_by_id(input('Enter Product ID: '))
99                 print(temp_product.get_product_details())
100            elif product_choice == "3":
101                temp_product = product_manager.get_product_by_id(input('Enter Product ID: '))
102                temp_product.update_product_info(price=input('Enter New Product Price'), description=input('Enter New Product Description'))
103            elif product_choice == "4":
104                break
105            else:
106                print("Invalid choice. Please enter a number between 1 and 4.")
107
108    elif choice == "3":
109        while True:
110            order_manager()
111            order_choice = input("Enter your choice (1-5): ")
112            cm = CustomerManager()
113
114            if order_choice == "1":
115                order_manager.add_order(Orders(
116                    input("Enter Order ID: "),
117                    cm.get_customer_by_id(input('Enter Customer ID: ')),
118                    input("Enter Order Date: "),
119                    input("Enter Total Amount: "),
120                    input("Enter Order Status: ")
121                ))
122            elif order_choice == "2":
123                temp_Data = order_manager.get_order_by_id(input(
124                    'Enter Order ID: '
125                ))
126                print(temp_Data.get_order_details())
127            elif order_choice == "3":
```

- Bottom Status Bar:** 1:1, CRLF, UTF-8, 4 spaces, Python 3.8

The screenshot shows a Python IDE interface with the following details:

- Project:** TechShop
- File:** main.py
- Code Content:**

```
96             input("Enter Description: "),
97             float(input("Enter Price: "))
98         )
99     elif product_choice == "2":
100        temp_product = ProductManager.get_product_by_id(input('Enter Product ID: '))
101        print(temp_product.get_product_details())
102    elif product_choice == "3":
103        temp_product = product_manager.get_product_by_id(input('Enter Product ID: '))
104        temp_product.update_product_info(price=input('Enter New Product Price'), description=input('Enter New Product Description'))
105    elif product_choice == "4":
106        break
107    else:
108        print("Invalid choice. Please enter a number between 1 and 4.")
109
110    elif choice == "3":
111        while True:
112            order_manager()
113            order_choice = input("Enter your choice (1-5): ")
114            cm = CustomerManager()
115
116            if order_choice == "1":
117                order_manager.add_order(Orders(
118                    input("Enter Order ID: "),
119                    cm.get_customer_by_id(input('Enter Customer ID: ')),
120                    input("Enter Order Date: "),
121                    input("Enter Total Amount: "),
122                    input("Enter Order Status: ")
123                ))
124            elif order_choice == "2":
125                temp_Data = order_manager.get_order_by_id(input(
126                    'Enter Order ID: '
127                ))
128                print(temp_Data.get_order_details())
129            elif order_choice == "3":
```

- Bottom Status Bar:** 1:1, CRLF, UTF-8, 4 spaces, Python 3.8

The screenshot shows the PyCharm IDE interface with the following details:

- Project Tree:** The left sidebar displays the project structure under "TechShop". It includes:
 - TechShop** (~Downloads/TechShop)
 - custom_exception**: Contains `__init__.py` and `CustomException.py`.
 - DBConnector**
 - models**: Contains `__init__.py` and several model files: `Customers.py`, `Inventory.py`, `OrderDetails.py`, `Orders.py`, and `Products.py`.
 - services**: Contains `__init__.py` and service manager files: `CustomerManager.py`, `InventoryManager.py`, `OrderManager.py`, and `ProductManager.py`.
 - main.py** (highlighted in yellow).
 - External Libraries**
 - Scratches and Consoles**
- Code Editor:** The right pane shows the content of `main.py`. The code implements a menu system for managing orders, inventory, and products. It uses `order_manager` and `inventory_manager` objects to perform operations like getting orders by ID, updating order status, canceling orders, adding/removing products from inventory, listing products, and listing low/out-of-stock products.
- Status Bar:** At the bottom, it shows the file path as "TechShop > main.py", and the bottom right corner indicates the Python version as "Python 3.8".

The screenshot shows a Python project named "TechShop" in VS Code. The project structure on the left includes "TechShop", "custom_exception", "DBConnector", "models", "services", "External Libraries", and "Scratches and Consoles". The "main.py" file is currently selected and open in the editor. The code in "main.py" is as follows:

```
customerManager.py  InventoryManager.py  OrderManager.py  ProductManager.py  main.py x  ▾ 3 13 7 ▾ 1:1 CRLF UTF-8 4 sp

143 elif choice == "4":
144     while True:
145         inventory_management_menu()
146         inventory_choice = input("Enter your choice (1-7): ")
147
148         if inventory_choice == "1":
149             inventory_manager.add_to_inventory(input('Enter Product ID: '), input('Enter Product Name: '))
150         elif inventory_choice == "2":
151             inventory_manager.remove_from_inventory(input('Enter Product ID: '), input('Enter Quantity: '))
152         elif inventory_choice == "3":
153             inventory_manager.list_all_products()
154         elif inventory_choice == "4":
155             inventory_manager.list_low_stock_products(2)
156         elif inventory_choice == "5":
157             inventory_manager.list_out_of_stock_products()
158         elif inventory_choice == "6":
159             inventory_manager.list_all_products()
160         elif inventory_choice == "7":
161             break
162         else:
163             print("Invalid choice. Please enter a number between 1 and 7.")
164
165         elif choice == "5":
166             print("Exiting TechShop. Goodbye!")
167             break
168
169         else:
170             print("Invalid choice. Please enter a number between 1 and 5.")
```

Outputs:

Welcome to TechShop!

1. Customer Management
2. Product Management
3. Order Management
4. Inventory Management
5. Exit

Enter your choice (1-5): |

Customer Management Menu:

1. Register Customer
2. View Customer Details
3. Update Customer Information
4. Back to Main Menu

Enter your choice (1-4):

```
Enter your choice (1-4): 2
Enter CustomerID: 1
Customer ID: 1
Name: John Doe
DOB: 1990-01-15
Email: doe.john@email.com
Phone: 1234567890
Address: Avenue Street 1, Los Angeles, USA
```

Welcome to TechShop!

1. Customer Management
2. Product Management
3. Order Management
4. Inventory Management
5. Exit

Enter your choice (1-5): 5

Exiting TechShop. Goodbye!

Product Management Menu:

1. Add Product
2. View Product Details
3. Update Product Information
4. Back to Main Menu

Enter your choice (1-4):

Enter your choice (1-5): 3

Order Management Menu:

1. Place Order
2. View Order Details
3. Update Order Status
4. Cancel Order
5. Back to Main Menu

Enter your choice (1-5):