

```
1  //#####
2  //Letter.h
3  //#####
4  #pragma once
5  //*****
6  // File name:   Letter.h
7  // Author:      Taylor Isaac
8  // Date:        5/2/2021
9  // Class:       CS 250
10 // Assignment:  046Polymorphism_Classes
11 // Purpose:     Demonstrate the Letter class
12 //*****
13 #include <string>
14 #include <iostream>
15 #include "Parcels.h"
16
17 using namespace std;
18
19 class Letter : public Parcels {
20 public:
21     Letter ();
22     virtual bool read (istream&);
23     virtual void print (ostream&);
24
25     virtual double getCost ();
26     virtual int getDaysForDelivery () ;
27     virtual double getInsuranceExpense (double) const;
28     virtual double getRushExpense (double) const;
29
30 private:
31     double mInsuranceFlatRate;
32     double mRushCostMultiplier;
33 };
34
35
36 //#####
37 //Overnight.h
38 //#####
39 #pragma once
40 //*****
41 // File name:   Overnight.h
42 // Author:      Taylor Isaac
43 // Date:        5/2/2021
44 // Class:       CS 250
45 // Assignment:  06Polymorphism_Classes
46 // Purpose:     Demonstrate the Overnight class
47 //*****
48 #include <string>
49 #include <iostream>
```

```
50 #include "Parcels.h"
51 using namespace std;
52
53 class Overnight : public Parcels {
54 public:
55     Overnight ();
56     int getVol () const { return mVol; }
57
58     virtual bool read (istream&);
59     virtual void print (ostream&);
60
61     virtual double getCost ();
62     virtual int getDaysForDelivery ();
63     virtual double getInsuranceExpense (double) const;
64     virtual double getRushExpense (double) const;
65
66 private:
67     int mVol;
68     double mInsuranceCostMultiplier;
69     double mRushCostMultiplier;
70 };
71
72
73 //#####
74 //Parcels.h
75 //#####
76 #pragma once
77 //*****
78 // File name:   Parcels.h
79 // Author:      Taylor Isaac
80 // Date:        5/2/2021
81 // Class:       CS 250
82 // Assignment:  06Polymorphism_Classes
83 // Purpose:     Demonstrate the Parcels class
84 //*****
85
86 #include <string>
87 #include <iostream>
88
89 using namespace std;
90
91 class Parcels {
92 public:
93     Parcels();
94     virtual bool read (istream&);
95     virtual void print (ostream&);
96     static bool getValidUserTID (int index, Parcels* apcArrayOfParcels[],
97                                 int userChoice);
98     bool getInsuranceTruth ();
```

```
99     bool getRushTruth ();
100     int getWeightOz () const { return mWeightOz; }
101     int getDistance () const { return mDistanceToTravel; }
102
103     virtual double getCost () = 0;
104     virtual int getDaysForDelivery () = 0;
105     virtual double getInsuranceExpense (double) const = 0;
106     virtual double getRushExpense (double) const = 0;
107
108     void addInsurance ();
109     void addRush ();
110
111 private:
112     int mTrackingNumb;
113     string mToAddress;
114     string mFromAddress;
115     int mWeightOz;
116     int mDistanceToTravel;
117     int mMilesPerDayCanTravel;
118     int mDayMinimumOfTravel;
119     bool mbInsured;
120     bool mbRush;
121 };
122
123
124 //#####
125 //Postcard.h
126 //#####
127 #pragma once
128 //*****
129 // File name:   Postcard.h
130 // Author:      Taylor Isaac
131 // Date:        5/2/2021
132 // Class:       CS 250
133 // Assignment:  06Polymorphism_Classes
134 // Purpose:     Demonstrate the Postcard class
135 //*****
136
137 #include <string>
138 #include <iostream>
139 #include "Parcels.h"
140
141 using namespace std;
142
143 class Postcard : public Parcels {
144 public:
145     Postcard ();
146     virtual bool read (istream&);
147     virtual void print (ostream&);
```

```
148
149     virtual double getCost ();
150     virtual int getDaysForDelivery ();
151     virtual double getInsuranceExpense (double) const;
152     virtual double getRushExpense (double) const;
153
154 private:
155     string mMessage;
156     double mInsuranceFlatRate;
157     double mRushCost;
158 };
159
160
161 //#####
162 //Letter.cpp
163 //#####
164 //*****
165 // File name: Letter.cpp
166 // Author: Taylor Isaac
167 // Date: 5/2/2021
168 // Class: CS 250
169 // Assignment: 06Polymorphism_Classes
170 // Purpose: Demonstrate Letter and its pertinence to inheritance
171 //*****
172
173 #include "Parcels.h"
174 #include "Letter.h"
175 #include <iostream>
176 #include <string>
177 #include <iomanip>
178
179 //*****
180 // Constructor: Letter
181 //
182 // Description: Provides initialization for the appropriate data member
183 //              variables of a letter--this should have pertinence to
184 //              inheritance since this class is a derived class
185 //
186 // Parameters: None
187 //
188 // Returned: none
189 //*****
190
191 Letter::Letter() : Parcels(), mInsuranceFlatRate(0.45),
192                  mRushCostMultiplier(1.1) {
193 }
194
195 //*****
196 // Function: read
```

```
197 //
198 // Description: utilizes the properties of inheritance, (from its parent
199 //              function) reading in data
200 //
201 // Parameters:  rcIn - designated input option
202 //
203 // Returned:    determines whether or not data was read in
204 //*****
205
206 bool Letter::read(istream& rcIn) {
207     bool bTheTruth = false;
208     if (Parcels::read(rcIn)) {
209         bTheTruth = true;
210     }
211     return bTheTruth;
212 }
213
214 //*****
215 // Function:    print
216 //
217 // Description: prints an object's correct associated information
218 //
219 // Parameters:  rcOut - one may specifit whether to the console or to an
220 //              output file
221 //
222 // Returned:    none
223 //*****
224
225 void Letter::print(ostream& rcOut) {
226     Parcels::print(rcOut);
227     rcOut << "\n";
228 }
229
230 //*****
231 // Function:    getCost
232 //
233 // Description: gets the total running costs if particular attributes are
234 //              valid
235 //
236 // Parameters:  none
237 //
238 // Returned:    the total running cost
239 //*****
240
241 double Letter::getCost() {
242     double runningCost = (getWeightOz() * mInsuranceFlatRate);
243     if (Parcels::getInsuranceTruth()) {
244         runningCost += mInsuranceFlatRate;
245     }
```

```
246
247     if (Parcels::getRushTruth()) {
248         runningCost *= mRushCostMultiplier;
249     }
250     cout << fixed << setprecision(2);
251     return runningCost;
252 }
253
254 //*****
255 // Function:    getDaysForDelivery
256 //
257 // Description: gets the total running number of days if particular
258 //               attributes are valid
259 //
260 // Parameters:  none
261 //
262 // Returned:    the total running number of days required for delivery
263 //*****
264
265 int Letter::getDaysForDelivery() {
266     const int MILES_PER_DAY_CAN_TRAVEL = 100;
267     const int ZERO_THRESHOLD = 0;
268     const int ONE_DAY_THRESHOLD = 1;
269     int milesToTravel = Parcels::getDistance();
270     int daysForDelivery = 0;
271
272     if (MILES_PER_DAY_CAN_TRAVEL >= milesToTravel) {
273         daysForDelivery = 1;
274     }
275     else {
276         while (milesToTravel > ZERO_THRESHOLD) {
277             (milesToTravel -= MILES_PER_DAY_CAN_TRAVEL);
278             daysForDelivery++;
279         }
280     }
281     if (Parcels::getRushTruth()) {
282         if (daysForDelivery > ONE_DAY_THRESHOLD) {
283             daysForDelivery -= 1;
284         }
285     }
286     return daysForDelivery;
287 }
288
289 //*****
290 // Function:    getInsuranceExpense
291 //
292 // Description: returns the calculated insurance expense for the unique
293 //               parcel
294 //
```

```
295 // Parameters:  none
296 //
297 // Returned:    the insurance's flat rate for the unique parcel
298 //*****
299
300 double Letter::getInsuranceExpense(double currCost) const {
301     return mInsuranceFlatRate;
302 }
303
304 //*****
305 // Function:    getRushExpense
306 //
307 // Description: returns the calculated rush expense for the unique
308 //              parcel
309 //
310 // Parameters:  currCost - the current cost of the unique type of parcel to
311 //
312 // Returned:    the rushing expense
313 //*****
314
315 double Letter::getRushExpense(double currCost) const {
316     double actualRushExpense = 0;
317     actualRushExpense = (currCost * mRushCostMultiplier);
318     return actualRushExpense;
319 }
320
321
322 //*****
323 //Overnight.cpp
324 //*****
325 //*****
326 // File name:  Parcels.cpp
327 // Author:     Taylor Isaac
328 // Date:       5/2/2021
329 // Class:      CS 250
330 // Assignment: 06Polymorphism_Classes
331 // Purpose:    Demonstrate Inheritance
332 //*****
333
334 #include "Parcels.h"
335 #include "Overnight.h"
336 #include <iostream>
337 #include <string>
338 #include <iomanip>
339
340 //*****
341 // Constructor: Overnight
342 //
343 // Description: Provides initialization for the appropriate data member
```

```
344 //          variables of an overnight package. This should set up
345 //          inheritance for this class's derived classes
346 //
347 // Parameters:  None
348 //
349 // Returned:   none
350 //*****
351
352 Overnight::Overnight() : Parcels(), mVol(-1), mInsuranceCostMultiplier(0.25),
353                          mRushCostMultiplier(0.75) {
354
355 }
356
357 //*****
358 // Function:   read
359 //
360 // Description: using inherited functionality from its parent class to read
361 //              in more unique data tailored to its own derived class
362 //
363 // Parameters:  rcIn - designated input option
364 //
365 // Returned:   the truth of whether or not data was successful in reading in
366 //*****
367
368 bool Overnight::read(istream& rcIn) {
369     bool bTheTruth = false;
370     Parcels::read(rcIn);
371     if (rcIn >> mVol) {
372         bTheTruth = true;
373     }
374     return bTheTruth;
375 }
376
377 //*****
378 // Function:   print
379 //
380 // Description: prints an object's correct associated information
381 //
382 // Parameters:  rcOut - one may specifit whether to the console or to an
383 //              output file
384 //
385 // Returned:   none
386 //*****
387
388 void Overnight::print(ostream& rcOut) {
389     Parcels::print(rcOut);
390     rcOut << "\tOVERNIGHT!\n";
391 }
392
```



```
393 //*****
394 // Function:    getCost
395 //
396 // Description: gets the total running costs if particular attributes are
397 //              valid
398 //
399 // Parameters:  none
400 //
401 // Returned:    the total running cost
402 //*****
403
404 double Overnight::getCost() {
405     const double INSURANCE_COST_MULTIPLIER = 1.25;
406     const double RUSH_MULTIPLIER = 1.75;
407     const int UPPER_BOUND_COST = 100;
408     double runningCost = 0;
409     if (getVol() > UPPER_BOUND_COST) {
410         runningCost = 20;
411     }
412     if (getVol() <= UPPER_BOUND_COST) {
413         runningCost = 12;
414     }
415     if (Parcels::getInsuranceTruth()) {
416         runningCost *= INSURANCE_COST_MULTIPLIER;
417     }
418     if (Parcels::getRushTruth()) {
419         runningCost *= RUSH_MULTIPLIER;
420     }
421     cout << fixed << setprecision(2);
422     return runningCost;
423 }
424
425 //*****
426 // Function:    getDaysForDelivery
427 //
428 // Description: gets the total running number of days if particular
429 //              attributes are valid
430 //
431 // Parameters:  none
432 //
433 // Returned:    the total running number of days required for delivery
434 //*****
435
436 int Overnight::getDaysForDelivery() {
437     const int MAX_MILES_PER_DAY_CAN_TRAVEL = 1000;
438     int milesToTravel = Parcels::getDistance();
439     int daysForDelivery = 0;
440
441     if (MAX_MILES_PER_DAY_CAN_TRAVEL >= milesToTravel) {
```

```
442     daysForDelivery = 1;
443 }
444 else {
445     daysForDelivery += 2;
446 }
447 if (Parcels::getRushTruth()) {
448     daysForDelivery = 1;
449 }
450 return daysForDelivery;
451 }
452
453 //*****
454 // Function:    getInsuranceExpense
455 //
456 // Description: returns the calculated insurance expense for the unique
457 //               parcel
458 //
459 // Parameters:  none
460 //
461 // Returned:    the insurance's overall rate for the unique parcel
462 //*****
463
464 double Overnight::getInsuranceExpense(double currCost) const {
465     double actualInsuranceExpense = 0;
466     actualInsuranceExpense = (currCost * mInsuranceCostMultiplier);
467     return actualInsuranceExpense;
468 }
469
470 //*****
471 // Function:    getRushExpense
472 //
473 // Description: returns the calculated rush expense for the unique
474 //               parcel
475 //
476 // Parameters:  currCost - the current cost of the unique type of parcel to
477 //
478 // Returned:    the rushing expense
479 //*****
480
481 double Overnight::getRushExpense(double currCost) const {
482     double actualRushExpense = 0;
483     actualRushExpense = (currCost * mRushCostMultiplier);
484     return actualRushExpense;
485 }
486
487
488 //#####
489 //Parcels.cpp
490 //#####
```

```
491 //*****
492 // File name:  Parcels.cpp
493 // Author:     Taylor Isaac
494 // Date:       5/2/2021
495 // Class:      CS 250
496 // Assignment: 06Polymorphism_Classes
497 // Purpose:    Demonstrate Inheritance using a Parcel
498 //*****
499
500 #include "Parcels.h"
501 #include <iostream>
502 #include <string>
503
504 //*****
505 // Constructor: Parcels
506 //
507 // Description: Provides initialization for the appropriate data member
508 //              variables of a parcel. This should set up inheritance for
509 //              this class's derived classes
510 //
511 // Parameters:  None
512 //
513 // Returned:    none
514 //*****
515
516 Parcels::Parcels() {
517     mTrackingNumb = -1;
518     mToAddress = "";
519     mFromAddress = "";
520     mWeightOz = -1;
521     mDistanceToTravel = -1;
522     mMilesPerDayCanTravel = -1;
523     mDayMinimumOfTravel = -1;
524     mbInsured = false;
525     mbRush = false;
526 }
527
528 //*****
529 // Function:    read
530 //
531 // Description: reads in specified data (the derived classes will use this
532 //              exact function, but add more special things to read in
533 //
534 // Parameters:  rcIn - designated input option
535 //
536 // Returned:    determination of whether or not data was read in
537 //*****
538
539 bool Parcels::read(istream& rcIn) {
```

```
540     bool bTheTruth = false;
541     if (rcIn >> mTrackingNumb >> mToAddress >> mFromAddress >> mWeightOz
542         >> mDistanceToTravel) {
543         bTheTruth = true;
544     }
545
546     return bTheTruth;
547 }
548
549 //*****
550 // Function:    print
551 //
552 // Description: prints an object's correct associated information
553 //
554 // Parameters:  ostream - one may specifit whether to the console or to an
555 //              output file
556 //
557 // Returned:    none
558 //*****
559
560 void Parcels::print(ostream& rcOut) {
561     rcOut << "TID: " << mTrackingNumb << "\tFrom: " << mFromAddress
562         << "\tTo: " << mToAddress;
563     if (mbInsured) {
564         rcOut << "\tINSURED\t";
565     }
566     if (mbRush) {
567         rcOut << "\tRUSH";
568     }
569 }
570
571
572 //*****
573 // Function:    getUserTID
574 //
575 // Description: gets the user input's desired TID. This function checks
576 //              to see if any of the TID's in the array of pointers matches
577 //              up with what the user enters in
578 //
579 // Parameters:  index          - actual size of array of pointers
580 //
581 //              apcArrayOfParcels - the array of pointers passed in
582 //
583 //              userTID         - passes in the user's inputted initial TID
584 //
585 // Returned:    returns a boolean variable if valid TID
586 //*****
587
588 bool Parcels::getValidUserTID(int index, Parcels* apcArrayOfParcels[],
```

```
589         int userTID) {
590     bool validUserTID = false;
591     for (int start = 0; start < index; start++) {
592         if (apcArrayOfParcels[start] != nullptr &&
593             apcArrayOfParcels[start]->mTrackingNumb == userTID) {
594             validUserTID = true;
595         }
596     }
597     return validUserTID;
598 }
599
600 //*****
601 // Function:    getInsuranceTruth
602 //
603 // Description: gets the returned value of the current state of being insured
604 //               or not
605 //
606 // Parameters:  none
607 //
608 // Returned:    returns a boolean variable of insurance
609 //*****
610
611 bool Parcels::getInsuranceTruth() {
612     return mbInsured;
613 }
614
615 //*****
616 // Function:    getRushTruth
617 //
618 // Description: gets the returned value of whether or not there is rushed
619 //               delivery
620 //
621 // Parameters:  none
622 //
623 // Returned:    returns a boolean variable of whether or not its rushed
624 //*****
625
626 bool Parcels::getRushTruth() {
627     return mbRush;
628 }
629
630 //*****
631 // Function:    addInsurance
632 //
633 // Description: sets true the state of being insured for the unique type of
634 //               parcel
635 //
636 // Parameters:  none
637 //
```

```
638 // Returned:    none
639 //*****
640
641 void Parcels::addInsurance() {
642     mbInsured = true;
643 }
644
645 //*****
646 // Function:    addRush
647 //
648 // Description: sets true the unique parcel's state of being rushed
649 //
650 // Parameters:  none
651 //
652 // Returned:    none
653 //*****
654
655 void Parcels::addRush() {
656     mbRush = true;
657 }
658
659
660 //*****
661 //Postcard.cpp
662 //*****
663 //*****
664 // File name:  Postcard.cpp
665 // Author:     Taylor Isaac
666 // Date:       5/2/2021
667 // Class:      CS 250
668 // Assignment: 06Polymorphism_Classes
669 // Purpose:    Demonstrate Inheritance
670 //*****
671
672 #include "Parcels.h"
673 #include "Letter.h"
674 #include <iostream>
675 #include "Postcard.h"
676 #include <string>
677 #include <iomanip>
678
679 //*****
680 // Constructor: Postcard
681 //
682 // Description: Provides initialization for the appropriate data member
683 //              variables of a postcard. This should utilize inheritance for
684 //              this class's derived classes
685 //
686 // Parameters:  None
```

```

687 //
688 // Returned:    none
689 //*****
690
691 Postcard::Postcard() : Parcels(), mMessage(""), mInsuranceFlatRate(0.15),
692                       mRushCost(0.25) {
693 }
694
695 //*****
696 // Function:    read
697 //
698 // Description: using inherited functionality from its parent class to read
699 //              in more unique data tailored to its own derived class
700 //
701 // Parameters:  rcIn - designated input option
702 //
703 // Returned:    determination of whether or not data was read in
704 //*****
705
706 bool Postcard::read(istream& rcIn) {
707     bool bTheTruth = false;
708     Parcels::read(rcIn);
709     if (rcIn >> mMessage) {
710         bTheTruth = true;
711     }
712     return bTheTruth;
713 }
714
715 //*****
716 // Function:    print
717 //
718 // Description: prints an object's correct associated information
719 //
720 // Parameters:  rcOut - one may specifit whether to the console or to an
721 //              output file
722 //
723 // Returned:    none
724 //*****
725
726 void Postcard::print(ostream& rcOut) {
727     Parcels::print(rcOut);
728     rcOut << "\t" << mMessage << "\n";
729 }
730
731 //*****
732 // Function:    getCost
733 //
734 // Description: gets the total running costs if particular attributes are
735 //              valid

```

```
736 //
737 // Parameters:  none
738 //
739 // Returned:    the total running cost for the postcard
740 //*****
741
742 double Postcard::getCost() {
743     double runningCost = 0.15;
744
745     if (Parcels::getRushTruth()) {
746         runningCost += mRushCost;
747     }
748     if (Parcels::getInsuranceTruth()) {
749         runningCost += mInsuranceFlatRate;
750     }
751     cout << fixed << setprecision(2);
752     return runningCost;
753 }
754
755 //*****
756 // Function:    getDaysForDelivery
757 //
758 // Description: gets the total running number of days if particular
759 //              attributes are valid
760 //
761 // Parameters:  none
762 //
763 // Returned:    the total running number of days required for delivery of a
764 //              postcard
765 //*****
766
767 int Postcard::getDaysForDelivery() {
768     const int MILES_PER_DAY_CAN_TRAVEL = 10;
769     const int ZERO_THRESHOLD = 0;
770     const int ONE_DAY_THRESHOLD = 1;
771     int milesToTravel = Parcels::getDistance();
772     int daysForDelivery = 0;
773
774     if (MILES_PER_DAY_CAN_TRAVEL >= milesToTravel) {
775         daysForDelivery = 1;
776     }
777     else {
778         while (milesToTravel > ZERO_THRESHOLD) {
779             (milesToTravel -= MILES_PER_DAY_CAN_TRAVEL);
780             daysForDelivery++;
781         }
782     }
783     if (Parcels::getRushTruth()) {
784         if (daysForDelivery > ONE_DAY_THRESHOLD) {
```



```
785     daysForDelivery -= 1;
786 }
787 }
788 return daysForDelivery;
789 }
790
791 //*****
792 // Function:    getInsuranceExpense
793 //
794 // Description: returns the calculated insurance expense for the unique
795 //              parcel
796 //
797 // Parameters:  none
798 //
799 // Returned:    the insurance's flat rate for the unique parcel
800 //*****
801
802 double Postcard::getInsuranceExpense(double currCost) const {
803     return mInsuranceFlatRate;
804 }
805
806 //*****
807 // Function:    getRushExpense
808 //
809 // Description: returns the calculated rush expense for the unique
810 //              parcel
811 //
812 // Parameters:  currCost - the current cost of the unique type of parcel to
813 //
814 // Returned:    the rushing expense
815 //*****
816
817 double Postcard::getRushExpense(double currCost) const {
818     return mRushCost;
819 }
820
821
822 //#####
823 //main.cpp
824 //#####
825 //*****
826 // File name:    main.cpp
827 // Author:       Taylor Isaac
828 // Date:         5/2/2021
829 // Class:        CS 250
830 // Assignment:   06Polymorphism
831 // Purpose:      Demonstrate the parcels class and its derived classes
832 //*****
833
```

```
834 #include "Parcels.h"
835 #include "Letter.h"
836 #include "Overnight.h"
837 #include "Postcard.h"
838 #include <iostream>
839 #include <fstream>
840 #include <iomanip>
841 #include <string>
842 #include <vld.h>
843 const int OPTION_ONE = 1;
844 const int OPTION_TWO = 2;
845 const int OPTION_THREE = 3;
846 const int OPTION_FOUR = 4;
847 const int OPTION_FIVE = 5;
848 const string INPUT_FILE = "parcels.txt";
849 int printOptionsMenuAndGetChoice();
850 void openFileForRead(ifstream& rcInfile);
851 void closeFileForRead(ifstream& rcInfile);
852 void printAllParcels(int index, Parcels* apcParcelsObjs[]);
853 int obtainInitialTID();
854 using namespace std;
855
856 //*****
857 // Function:    main
858 //
859 // Description: contains the necessary components, variables, and functions
860 //              to execute tasks and operate on parcel related activity
861 //
862 // Parameters:  None
863 //
864 // Returned:    EXIT_SUCCESS
865 //*****
866
867 int main() {
868     const int MAX_NUMB_ARRAY = 25;
869     const string PROMPT_CHOICE = "Choice> ";
870     const char POSTCARD_SYMBOL = 'P';
871     const char LETTER_SYMBOL = 'L';
872     const char OVERNIGHT_SYMBOL = 'O';
873     const int NO_PARCELS = 0;
874     char parcelEat = '.';
875     int index = 0;
876     int initialID = -1;
877     int userChoice = -2;
878     bool bIsValidTID = false;
879     double eatCost = 0;
880     Parcels* apcParcelsObjs[MAX_NUMB_ARRAY] = { nullptr };
881     ifstream cInFile;
882
```

```
883  openFileForRead(cInFile);
884  cout << "Mail Simulator!\n";
885
886  while (cInFile >> parcelEat) {
887
888      switch (parcelEat) {
889          case POSTCARD_SYMBOL: apcParcelsObjs[index] = new Postcard();
890              apcParcelsObjs[index]->read(cInFile);
891              ++index;
892              break;
893          case LETTER_SYMBOL: apcParcelsObjs[index] = new Letter();
894              apcParcelsObjs[index]->read(cInFile);
895              ++index;
896              break;
897          case OVERNIGHT_SYMBOL: apcParcelsObjs[index] = new Overnight();
898              apcParcelsObjs[index]->read(cInFile);
899              ++index;
900              break;
901          default: break;
902      }
903  }
904  if (NO_PARCELS == index) {
905      cout << "File is empty.\n";
906      exit(EXIT_FAILURE);
907  }
908  while (OPTION_FIVE != userChoice) {
909      userChoice = printOptionsMenuAndGetChoice();
910      if (OPTION_ONE == userChoice) {
911          printAllParcels(index, apcParcelsObjs);
912      }
913
914      else if (OPTION_FOUR == userChoice) {
915
916          initialID = obtainInitialTID();
917          if (apcParcelsObjs[initialID - 1] != nullptr) {
918              bIsValidTID = Parcels::getValidUserTID(index, apcParcelsObjs,
919                  initialID);
920              if (bIsValidTID) {
921                  cout << "Delivered!\n";
922                  cout << apcParcelsObjs[initialID - 1]->getDaysForDelivery();
923                  cout << " Day, " << "$"
924                      << apcParcelsObjs[initialID - 1]->getCost();
925                  cout << "\n";
926                  apcParcelsObjs[initialID - 1]->print(cout);
927                  delete apcParcelsObjs[initialID - 1];
928                  apcParcelsObjs[initialID - 1] = { nullptr };
929              }
930          }
931      }
```

```
932     else if (OPTION_TWO == userChoice) {
933         initialID = obtainInitialTID();
934         if (apcParcelsObjs[initialID - 1] != nullptr) {
935             bIsValidTID = Parcels::getValidUserTID(index, apcParcelsObjs,
936                 initialID);
937             if (bIsValidTID) {
938                 cout << "Added Insurance for $";
939                 eatCost = (apcParcelsObjs[initialID - 1]->getCost());
940
941                 cout << apcParcelsObjs[initialID - 1]->
942                     getInsuranceExpense(eatCost);
943
944                 cout << "\n";
945                 apcParcelsObjs[initialID - 1]->addInsurance();
946                 apcParcelsObjs[initialID - 1]->print(cout);
947             }
948         }
949     }
950     else if (OPTION_THREE == userChoice) {
951         initialID = obtainInitialTID();
952         if (apcParcelsObjs[initialID - 1] != nullptr) {
953             bIsValidTID = Parcels::getValidUserTID(index, apcParcelsObjs,
954                 initialID);
955             if (bIsValidTID) {
956                 cout << "Added Rush for $";
957                 eatCost = (apcParcelsObjs[initialID - 1]->getCost());
958                 cout << apcParcelsObjs[initialID - 1]->getRushExpense(eatCost);
959                 cout << "\n";
960                 apcParcelsObjs[initialID - 1]->addRush();
961                 apcParcelsObjs[initialID - 1]->print(cout);
962             }
963         }
964     }
965 }
966 for (int start = 0; start < index; start++) {
967     delete (apcParcelsObjs[start]);
968 }
969 closeFileForRead(cInFile);
970 return EXIT_SUCCESS;
971 }
972
973 //*****
974 // Function:    printOptionsMenuAndGetChoice
975 //
976 // Description: Outputs desired description or criterium with the option of
977 //               allowing the user to input a choice
978 //
979 // Parameters:  None
980 //
```

```
981 // Returned:    the returned valid choice as a number displayed from menu
982 //*****
983
984 int printOptionsMenuAndGetChoice() {
985     int userChoice = -1;
986     cout << "\n1. Print All\n2. Add Insurance\n3. Add Rush\n4. Deliver\n5. ";
987     cout << "Quit\n\n";
988     do {
989         cout << "Choice> ";
990         cin >> userChoice;
991     } while (!(OPTION_ONE == userChoice || OPTION_TWO == userChoice ||
992             OPTION_THREE == userChoice || OPTION_FOUR == userChoice ||
993             OPTION_FIVE == userChoice));
994     return userChoice;
995 }
996
997 //*****
998 // Function:    openFileForRead
999 //
1000 // Description: opens file, checking for proper opening
1001 //
1002 // Parameters:  rcInfile - specified input file
1003 //
1004 // Returned:    None
1005 //*****
1006
1007 void openFileForRead(ifstream& rcInfile) {
1008     rcInfile.open(INPUT_FILE);
1009     if (!rcInfile.is_open()) {
1010         cout << "Error opening file.\n";
1011         exit(EXIT_FAILURE);
1012     }
1013 }
1014
1015 //*****
1016 // Function:    closeFileForRead
1017 //
1018 // Description: closes a file opened for reading
1019 //
1020 // Parameters:  rInfile - the object used to close the file to be read
1021 //
1022 // Returned:    None
1023 //*****
1024
1025 void closeFileForRead(ifstream& rcInfile) {
1026     rcInfile.close();
1027 }
1028
1029 //*****
```

```
1030 // Function:    printAllParcels
1031 //
1032 // Description: when user chooses option 1, it prints all associated parcels
1033 //
1034 // Parameters:  index - the correct amount of positions associated with the
1035 //               array of pointers
1036 //
1037 //               *apcArrayOfParcels[] - allows an array of pointers of class
1038 //               Parcels to be passed in
1039 //
1040 // Returned:    None
1041 //*****
1042
1043 void printAllParcels(int index, Parcels* apcArrayOfParcels[]) {
1044     cout << "\n";
1045     for (int start = 0; start < index; start++) {
1046         if (apcArrayOfParcels[start] != nullptr) {
1047             apcArrayOfParcels[start]->print(cout);
1048         }
1049     }
1050 }
1051
1052 //*****
1053 // Function:    obtainInitialTID
1054 //
1055 // Description: gets initial TID from user. We do not know if it's valid yet
1056 //
1057 // Parameters:  None
1058 //
1059 // Returned:    the user's inputted initial tracking ID number
1060 //*****
1061
1062 int obtainInitialTID() {
1063     int initialTID = -1;
1064     cout << "\nTID> ";
1065     cin >> initialTID;
1066     return initialTID;
1067 }
```