```java
/*
 * Programmer:  Chad Philip Johnson
 * Date Created:  Thursday, November 06th, 2012
 * Date of Last Modification:  Tuesday, December 11th, 2012
 *
 * Description:
 * Game.class contains the main program subroutines used by MyGame.class.  It
 * depends on CinReader.class to handle user input, Player.class to set up a
 * player character, World.class to set up the game world, and Room.class to
 * produce a location in the game world for the player's character to appear.
 */

import java.util.*;
import java.io.Serializable;

/**
 * Game.class contains the main program subroutines used by MyGame.class.  It
 * depends on CinReader.class to handle user input, Player.class to set up a
 * player character, World.class to set up the game world, and Room.class to
 * produce a location in the game world for the player's character to appear.
 *
 * @author Chad Philip Johnson
 * @version 1.0
 */

public class Game implements Serializable   {

    // Final room number
    static final int FINAL_ROOM     = 5;

    CinReader driverKeyboard;
    Player somePlayer;
    World someWorld;

    Room currentRoom;

    char chrUserInput;
    boolean blnKeyToTreasure;

    /**
      * Overloaded constructor:
      * Instantiate CinReader.class and World.class.  Assign player object to a local field.
      */

    public Game( Player somePlayer )    {

        this.driverKeyboard = new CinReader();
        this.somePlayer     = somePlayer;
        this.someWorld      = new World();

```

```
51          // Win condition to complete the game
52          this.blnKeyToTreasure   = false;
53
54      }
55
56      /**
57       * Main menu
58       */
59
60      public void go()    {
61
62          // Put player in the start room if a previously saved game has not been loaded
63          if( currentRoom == null )   {   currentRoom     = someWorld.getStartRoom(); }
64
65          while( true )   {
66
67              System.out.print    ( "\n\n\n\n'" + somePlayer.getPlayerName() + "' has " + somePlayer.getGoldHeld() + " gold
                pieces.\n" );
68              System.out.print    ( currentRoom );
69
70              // Main menu
71              System.out.print    ( "\nAvailable actions:\n" );
72              System.out.print    ( "\tGo (N)orth/(S)outh/(W)est/(E)ast\n" );
73              System.out.print    ( "\t(P)ick up or (D)rop an item\n" );
74              System.out.print    ( "\t(C)ollect gold\n" );
75              System.out.print    ( "\t(T)alk\n" );
76              System.out.print    ( "\t(A)ttack\n" );
77              System.out.print    ( "\tView (I)nventory\n" );
78              System.out.print    ( "\tChan(g)e player name\n" );
79              System.out.print    ( "\tSa(v)e or (Q)uit the game\n" );
80              System.out.print    ( "What would you like to do?  " );
81              chrUserInput    = String.valueOf( driverKeyboard.readChar() ).toLowerCase().charAt( 0 );
82
83              switch( chrUserInput )  {
84
85                  // Go North, South, West or East
86                  case 'n':
87                  case 's':
88                  case 'w':
89                  case 'e':
90                      if( !moveCharacter( chrUserInput ) )    {
91
92                          System.out.print    ( "\nYou can't go in that direction.\n" );
93
94                      } else {    System.out.print    ( "\nYou are leaving this room.\n" );   }
95
96                      pressEnterToContinue();
97                      break;
98
99                  // Pickup an item
```

```
100                 case 'p':
101                     if( !pickupItem() ) {   System.out.print    ( "\nThere aren't any items here.\n");  }
102
103                     pressEnterToContinue();
104                     break;
105
106                 // Drop an item
107                 case 'd':
108                     if( !dropItem() )   {   System.out.print    ( "\nYou currently do not have any items.\n" ); }
109
110                     pressEnterToContinue();
111                     break;
112
113                 // Collect gold in this room
114                 case 'c':
115                     int intGoldCollected    = currentRoom.retrieveGoldPieces();
116
117                     if( intGoldCollected > 0 )  {
118
119                         somePlayer.addGold( intGoldCollected );
120                         System.out.printf   ( "\nYou collected %d gold pieces.\n", intGoldCollected );
121
122                     } else {    System.out.print    ( "\nThere aren't any gold pieces here.\n" );   }
123
124                     pressEnterToContinue();
125                     break;
126
127                 // Talk to a character in this room
128                 case 't':
129                     if( winConditions() )   {
130
131                         // Get special messages from characters
132                         talkToCharacter( true );
133                         // Unlock the final room
134                         currentRoom.setDoor( 3, 5 );
135
136                     } else if( !talkToCharacter( false ) )  {   System.out.print    ( "\nThere isn't anyone to talk to here.\n"
                        );  }
137
138                     pressEnterToContinue();
139                     break;
140
141                 // Attack a character in this room
142                 case 'a':
143                     if( !attackCharacter() )    {   System.out.print    ( "\nThere isn't anyone here to attack.\n" );   }
144
145                     pressEnterToContinue();
146                     break;
147
148                 // View inventory
```

```
149                 case 'i':
150                     if( !showItems() )  {
151 
152                         System.out.print    ( "\nYou currently do not have any items.\n" );
153                         pressEnterToContinue();
154 
155                     } else {    pressEnterToContinue(); }
156 
157                     break;
158 
159                 // Change player name
160                 case 'g':
161                     System.out.print    ( "\nWhat would you like to change your name to?  " );
162                     somePlayer.setPlayerName( driverKeyboard.readString() );
163 
164                     System.out.printf   ( "\nYour character now goes by the name of %s.\n", somePlayer.getPlayerName() );
165 
166                     if( somePlayer.getPlayerName().toLowerCase().equals( "bob" ) )  {
167 
168                         blnKeyToTreasure    = true;
169 
170                     } else {    blnKeyToTreasure    = false;    }
171 
172                     pressEnterToContinue();
173 
174                     break;
175 
176                 // Save the game
177                 case 'v':
178                     System.out.print    ( "\nYou have chosen to save your game.\n" );
179                     System.out.print    ( "What would you like to name the file?  ");
180                     Storage.saveGame( this, driverKeyboard.readString() );
181                     break;
182 
183                 // Quit the game
184                 case 'q':
185                     System.out.print    ( "\nThanks for playing TextDungeon!\n\n" );
186                     break;
187 
188                 // Incorrect entries cause menu loop to start again
189                 default:
190                     System.out.print    ( "You entered '" + chrUserInput + "' which is not a valid option.\n\n" );
191                     pressEnterToContinue();
192                     continue;
193 
194             }
195 
196             // End game and print final message after player enters the last room
197             if( currentRoom.equals( someWorld.getRoom( FINAL_ROOM ) ) ) {
198 
```

```
                System.out.print    ( "\n\n\n\n" + currentRoom + "\n\nGame Over\n\n" );
                break;

            }

            // Exit loop to quit game
            if( chrUserInput == 'q' )   {   break;  };

        }

    }

    /**
     * Moves character in the appropriate direction by translating the characters n/s/w/e into their numerical counterparts.
     *
     * @param chrDirection Letter representing the direction the player wishes to go.
     * @return boolean Boolean value representing whether the character input was valid.
     */

    private boolean moveCharacter( char chrDirection )  {

        switch( chrDirection )  {

            case 'n':
                return changeRooms( 0 );

            case 's':
                return changeRooms( 1 );

            case 'w':
                return changeRooms( 2 );

            case 'e':
                return changeRooms( 3 );

        }

        return false;

    }

    /**
     * Changes the players position by moving to another room.
     *
     * @param intIndex Numerical value representing the newly selected room after a successful movement command.
     * @return boolean Boolean value representing whether or not the movement was successful.
     */

    private boolean changeRooms( int intIndex ) {

```

```
249         // Get the values of rooms connected to the current room
250         int[] intAvailableRooms = currentRoom.getDoor();
251 
252         // Only true when the selected direction returns a value for another room (rooms have values greater than or equal to
            zero).
253         if( intAvailableRooms[intIndex] >= 0 )  {
254 
255             // Get the value for the connected room and then load that room into the currentRoom variable
256             currentRoom     = someWorld.getRoom( intAvailableRooms[intIndex] );
257             return true;
258 
259         } else {
260 
261             return false;
262 
263         }
264 
265     }
266 
267     /**
268      * Display a list of the player's current inventory
269      *
270      * @return boolean Boolean value that returns true when the player is carrying items.
271      */
272 
273     private boolean showItems() {
274 
275         // Items currently held by the player
276         ArrayList<Item> itemsHeld   = somePlayer.getItem();
277 
278         if( !itemsHeld.isEmpty() )  {
279 
280             System.out.print    ( "\n\n'" + somePlayer.getPlayerName() + "' has " + somePlayer.getGoldHeld() + " gold and the
                following items:\n" );
281 
282             int intCount        = 0;
283             // Prints a list of carried items
284             for( Item item : itemsHeld )    {
285 
286                 intCount++;
287                 System.out.print    ( "\t[" + Integer.toString( intCount ) + "] " + item + "\n" );
288 
289             }
290 
291             return true;
292 
293         }
294 
295         return false;
296 
```

```
297         }
298
299         /**
300          * Pickup an item lying in the player's current location.
301          *
302          * @return boolean Boolean value representing whether or not an item is available to be picked up.
303          */
304
305         private boolean pickupItem()    {
306
307             // Items available in the current room
308             ArrayList<Item> itemsInRoom = currentRoom.getItem();
309
310             if( !itemsInRoom.isEmpty() )    {
311
312                 System.out.print    ( "\nThese items are in the room:\n" );
313
314                 Item tempItem       = null;
315                 int intUserInput    = 0;
316                 int intCount        = 0;
317                 // Prints a list of items in the current room
318                 for( Item item : itemsInRoom )  {
319
320                     intCount++;
321                     System.out.print    ( "\t[" + Integer.toString( intCount ) + "] " + item + "\n" );
322
323                 }
324
325                 // Select an item to pickup
326                 System.out.print    ( "\nWhich item would you like to pick up?  " );
327                 intUserInput    = driverKeyboard.readInt( 1, intCount ) - 1;
328
329                 tempItem        = currentRoom.removeItem( intUserInput );
330                 somePlayer.addItem( tempItem );
331
332                 System.out.print    ( "\nYou have retrieved the item:\n\t" + tempItem + "\n" );
333
334                 return true;
335
336             }
337
338             return false;
339
340         }
341
342         /**
343          * Drops a carried item in the current room.
344          *
345          * @return boolean Boolean value representing whether or not the player has an item that can be dropped.
346          */
```

```
347
348     private boolean dropItem()  {
349
350         // Items carried by the player that can be dropped in the current room
351         ArrayList<Item> itemsHeld   = somePlayer.getItem();
352
353         if( !itemsHeld.isEmpty() )  {
354
355             System.out.print    ( "\nYou are currently carrying the following items:\n" );
356
357             Item tempItem       = null;
358             int intUserInput    = 0;
359             int intCount        = 0;
360             // Prints a list of items that can be dropped
361             for( Item item : itemsHeld )    {
362
363                 intCount++;
364                 System.out.print    ( "\t[" + Integer.toString( intCount ) + "] " + item + "\n" );
365
366             }
367
368             // Select an item to drop
369             System.out.print    ( "\nWhich item would you like to drop?  " );
370             intUserInput    = driverKeyboard.readInt( 1, intCount ) - 1;
371
372             tempItem        = somePlayer.removeItem( intUserInput );
373             currentRoom.addItem( tempItem );
374
375             System.out.print    ( "\nYou have dropped the item:\n\t" + tempItem + "\n" );
376
377             return true;
378
379         }
380
381         return false;
382
383     }
384
385     /**
386      * Allows player to talk to a character by selecting from a list of characters within the current room.
387      *
388      * @param blnSpecialDialogue Indicates whether the character should say his or her special dialogue
389      * @return boolean Boolean value that returns true when there is a character available to talk to in the current room.
390      */
391
392     private boolean talkToCharacter( boolean blnSpecialDialogue )   {
393
394         // Characters in the current room
395         ArrayList<Character> charactersInRoom   = currentRoom.getCharacter();
396
```

```
397         if( !charactersInRoom.isEmpty() )   {
398 
399             System.out.print    ( "\n\nThe following characters are in this room:\n" );
400 
401             int intUserInput    = 0;
402             int intCount        = 0;
403             // Prints a list of characters in the current room
404             for( Character character : charactersInRoom )   {
405 
406                 intCount++;
407                 System.out.print    ( "\t[" + Integer.toString( intCount ) + "] " + character + "\n" );
408 
409             }
410 
411             // Select a character with which to speak
412             System.out.print    ( "\nWith whom would you like to speak?  " );
413             intUserInput    = driverKeyboard.readInt( 1, intCount ) - 1;
414 
415             System.out.print    ( charactersInRoom.get( intUserInput ).getCharacterName() + " has the following to say to
                you:\n\"" );
416 
417             // Show special dialogue if blnSpecialDialogue flag is 'true'
418             if( blnSpecialDialogue) {
419 
420                 System.out.print    ( charactersInRoom.get( intUserInput ).getSpecialDialogue() + "\"\n\n" );
421 
422             } else {    System.out.print    ( charactersInRoom.get( intUserInput ).getTalkDialogue() + "\"\n\n" );  }
423 
424             return true;
425 
426         }
427 
428         return false;
429 
430     }
431 
432     /**
433      * Allows the player to attack a character within the current room.
434      *
435      * @return boolean Boolean value that returns true when there is a character that can be attacked in the current room.
436      */
437 
438     private boolean attackCharacter()   {
439 
440         // Characters in the current room
441         ArrayList<Character> charactersInRoom   = currentRoom.getCharacter();
442 
443         if( !charactersInRoom.isEmpty() )   {
444 
445             System.out.print    ( "\n\nThe following characters are in this room:\n" );
```

```
446
447             int intUserInput    = 0;
448             int intCount        = 0;
449             // Prints a list of characters in the current room
450             for( Character character : charactersInRoom )   {
451
452                 intCount++;
453                 System.out.print    ( "\t[" + Integer.toString( intCount ) + "] " + character + "\n" );
454
455             }
456
457             // Select a character to attack
458             System.out.print    ( "\nWho would you like to attack?  " );
459             intUserInput    = driverKeyboard.readInt( 1, intCount ) - 1;
460
461             // Print character's response to the attack
462             System.out.print    ( charactersInRoom.get( intUserInput ).getCharacterName() + " says:\n\"" );
463             System.out.print    ( charactersInRoom.get( intUserInput ).getAttackDialogue() + "\"\n\n" );
464
465             return true;
466
467         }
468
469         return false;
470
471     }
472
473     /**
474      * Checks to see if game victory conditions are satisfied
475      *
476      * @return boolean Boolean value that returns true when victory conditions are satisfied.
477      */
478
479     private boolean winConditions() {
480
481         // Player has key to treasure (has changed his character's name to "Bob" to fool the Cross-Eyed Giant)
482         if( blnKeyToTreasure )  {
483
484             for( Character character : currentRoom.getCharacter() ) {
485
486                 // Player is in the same room as the Cross-Eyed Giant
487                 if( character.getCharacterName().equals( "Cross-eyed Giant" ) ) {   return true;    }
488
489             }
490
491         }
492
493         return false;
494
495     }
```

```
496
497         /**
498          * Console prompt for the player to press the enter button to continue playing
499          */
500
501         private void pressEnterToContinue() {
502
503             System.out.print( "Press enter to continue..." );
504             driverKeyboard.readString();
505
506         }
507
508
509     }
```