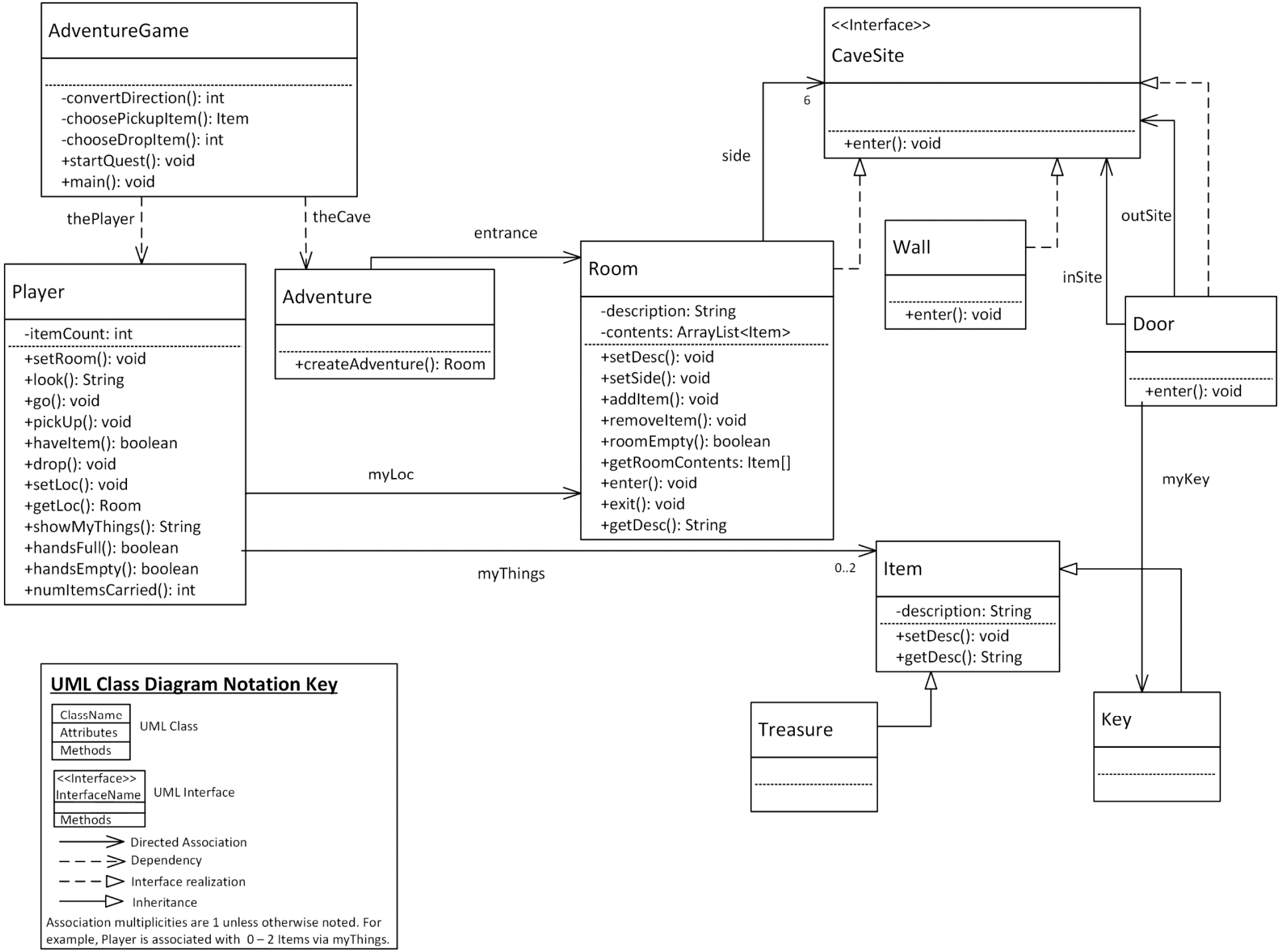
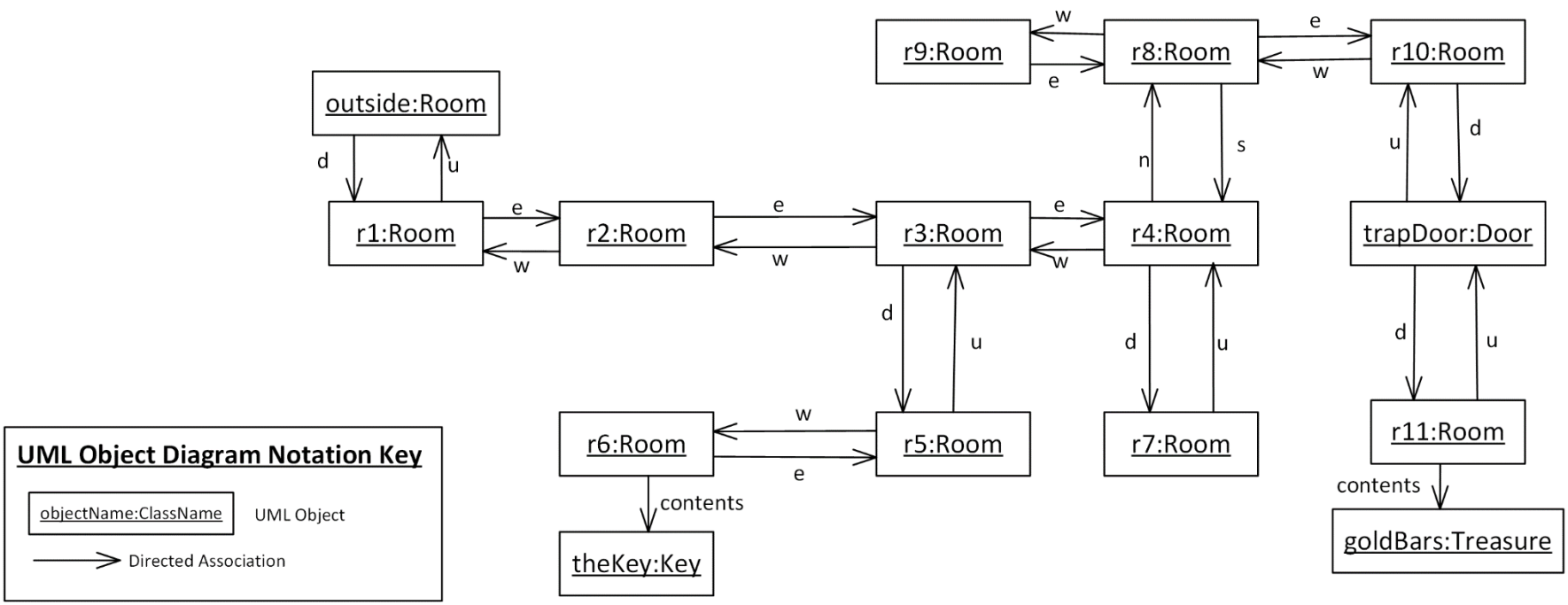

AdventureGame
-convertDirection(): int
-choosePickupItem(): Item
-chooseDropItem(): int
+startQuest(): void
+main(): void
thePlayer
theCave
<<Interface>>
CaveSite
+enter(): void
6
side
outSite
inSite
Player
-itemCount: int
+setRoom(): void
+look(): String
+go(): void
+pickUp(): void
+haveItem(): boolean
+drop(): void
+setLoc(): void
+getLoc(): Room
+showMyThings(): String
+handsFull(): boolean
+handsEmpty(): boolean
+numItemsCarried(): int
Adventure
+createAdventure(): Room
entrance
Room
-description: String
-contents: ArrayList<Item>
+setDesc(): void
+setSide(): void
+addItem(): void
+removeItem(): void
+roomEmpty(): boolean
+getRoomContents: Item[]
+enter(): void
+exit(): void
+getDesc(): String
Wall
+enter(): void
Door
+enter(): void
myLoc
myThings
0..2
myKey
Item
-description: String
+setDesc(): void
+getDesc(): String
Treasure
Key
UML Class Diagram Notation Key
ClassName
Attributes
Methods
UML Class
<<Interface>>
InterfaceName
Methods
UML Interface
Directed Association
Dependency
Interface realization
Inheritance
Association multiplicities are 1 unless otherwise noted. For example, Player is associated with 0 – 2 Items via myThings.


outside:Room
r1:Room
r2:Room
r3:Room
r4:Room
r5:Room
r6:Room
r7:Room
r8:Room
r9:Room
r10:Room
r11:Room
trapDoor:Door
theKey:Key
goldBars:Treasure
contents
contents
d
u
e
w
n
s
UML Object Diagram Notation Key
objectName:ClassName
UML Object
Directed Association

```
1  package AdventureGame;
2
3  /**  Adventure Game  Program Code
4       Copyright (c) 1999 James M. Bieman
5
6       To compile: javac cs314.a2.AdventureGame.java
7       To run:     java cs314.a2.AdventureGame
8       The main routine is AdventureGame.main
9
10      The AdventureGame is a Java implementation of the old text based
11      adventure game from long ago.  The design was adapted from
12      one in Gamma, Helm, Johnson, Vlissides (The Gang of Four),
13      "Design Patterns: Elements of Reusable Object-Oriented Software",
14      Addison-Wesley, 1997.
15
16      To really be consistent with the old game we would need a
17      much larger cave system with a hundred or so rooms, and a
18      more "understanding" user interface.
19
20      The old game just put you near the cave, displayed the "view"
21      as text, and offered no instructions.  If you gave a command that
22      it understood, you could proceed.  If your command could not
23      be interpreted, nothing would happen.  Rooms were never identified
24      precisely; your only clues came from the descriptions.  You would
25      have to remember or create your own map of the cave system to
26      find your way around.  Sometimes you could not return exactly
27      the way you came.  An exit to the east may not enter the west
28      side of the "adjacent room"; the passage might curve.
29
30      Perhaps, this implementation can evolve to be closer to
31      the original game, or even go beyond it.
32
33      Jim Bieman
34      September 1999.
35
36
37 /**  Adventure Game  Program Code
38 Copyright (c) 1999 James M. Bieman
39 Updated August 2010
40 - Code is put into package cs314.a2 to match current CS314 coding standards.
41 - Obsolete Vector is replaced with ArrayList with type parameters.
42 - Deletion of some unused variables.
43
44 To compile: javac cs314.a2.AdventureGame.java
45 To run:     java cs314.a2.AdventureGame
46
47 The main routine is AdventureGame.main
48
49            **/
50
51 import java.io.*;
52
53
54 public class AdventureGame {
55     private Adventure theCave;
56     private Player thePlayer;
57
58   /** Our system-wide internal representation of directions
```

```
59      is integers.  Here, we convert input string directions
60      into integers.  Internally, we use integers 0-9 as
61      directions throughout the program.  This is a bit of
62      a cludge, but is simpler for now than creating a Direction
63      class.  I use this cludge because Java in 1999 did not have
64      an enumerated data type.  */
65  private int convertDirection(String input){
66   char d = input.charAt(0);
67   int theDirection = 9999;
68   switch(d){
69       case 'n': case 'N': theDirection = 0;break;
70       case 's': case 'S': theDirection = 1;break;
71       case 'e': case 'E': theDirection = 2;break;
72       case 'w': case 'W': theDirection = 3;break;
73       case 'u': case 'U': theDirection = 4;break;
74       case 'd': case 'D': theDirection = 5;break;
75    }
76    return theDirection;
77  }
78
79  /** choosePickupItem determines the specific item
80      that a player wants to pick up.   */
81  private Item choosePickupItem(Player p,  BufferedReader keyB)
82                  throws IOException{
83   Item[] contentsArray = (p.getLoc()).getRoomContents();
84   String inputString = "prepare";
85   int theChoice = -1;
86
87   do {
88       System.out.println("The room has:");
89       for (int i = 0; i < contentsArray.length ; i++)
90       System.out.println((i+1) + ": "
91                  + contentsArray[i].getDesc());
92       System.out.print("Enter the number of the item to grab: ");
93       inputString = keyB.readLine();
94       System.out.println('\n');
95       if (inputString.equals("")) inputString = " ";
96       try  {
97        theChoice = Integer.parseInt(inputString);
98        } catch (NumberFormatException e) {
99           System.out.println("Invalid input.");
100          theChoice = -1;
101       }
102      if (theChoice < 0 || theChoice > contentsArray.length)
103     System.out.print("That item is not in the room.");
104      } while (theChoice < 0 || theChoice > contentsArray.length);
105
106  return contentsArray[theChoice-1];
107
108 }
109
110 /** chooseDropItem  determines the specific item
111     that a player wants to drop */
112 private int chooseDropItem(Player p,  BufferedReader keyB)
113    throws IOException{
114    String inputString = "prepare";
115    int theChoice = -1;
116
```

```
117     do {
118         System.out.println("You are carrying: " +
119                p.showMyThings() + '\n');
120         System.out.print("Enter the number of the item to drop: " );
121         inputString = keyB.readLine();
122         try  {theChoice = Integer.parseInt(inputString);}
123     catch (NumberFormatException e) {
124           System.out.println("Invalid input.");
125           theChoice = -1;
126           }
127         if (theChoice < 0 || theChoice > p.numItemsCarried())
128        System.out.print("Invalid choice.");
129        } while (theChoice < 0 || theChoice > p.numItemsCarried());
130
131    return theChoice;
132  }
133
134  public void startQuest() throws IOException{
135   Player thePlayer = new Player();
136   Adventure theCave = new Adventure();
137   Room startRm = theCave.createAdventure();
138   thePlayer.setRoom(startRm);
139
140   /** Create the keyboard to control the game; we only need one */
141    BufferedReader keyboard
142      = new BufferedReader(new InputStreamReader(System.in));
143    String inputString = "prepare";
144
145    /* The main query user, get command, interpret, execute cycle. */
146    while (inputString.charAt(0)!='q') {
147       System.out.println(thePlayer.look());
148       System.out.println("You are carrying: " +
149              thePlayer.showMyThings() + '\n');
150        /* get next move */
151    int direction = 9;
152
153          System.out.println("Which way (n,s,e,w,u,d)," +
154                 " or grab (g) or toss (t) an item," +
155                 " or quit (q)?" + '\n');
156          inputString = keyboard.readLine();
157      System.out.println('\n');
158      if (inputString.equals("")) inputString = " ";
159      char key = inputString.charAt(0);
160      switch (key){
161       // Go
162         case 'n': case 'N': case 's': case 'S':
163         case 'e': case 'E': case 'w': case 'W':
164         case 'u': case 'U': case 'd': case 'D':
165               direction = convertDirection(inputString);
166           thePlayer.go(direction);
167               break;
168           // Grab Item
169         case 'g': case 'G':
170           if (thePlayer.handsFull())
171          System.out.println("Your hands are full.");
172               else if ((thePlayer.getLoc()).roomEmpty())
173                       System.out.println("The room is empty.");
174           else {
```

```
175                     Item itemToGrab =
176                     choosePickupItem(thePlayer,keyboard);
177                     thePlayer.pickUp(itemToGrab);
178                     (thePlayer.getLoc()).removeItem(itemToGrab);
179             }
180           break;
181       // Drop Item
182          case 't': case 'T':
183          if (thePlayer.handsEmpty())
184         System.out.println("You have nothing to drop.");
185              else {
186            int itemToToss =
187             chooseDropItem(thePlayer,keyboard);
188                thePlayer.drop(itemToToss);
189           }
190           }
191    }
192
193  }
194
195public static void main(String args[])
196 throws IOException{
197 System.out.println("Welcome to the Adventure Game,\n"
198    + "which is inspired by an old game called the Colossal Cave Adventure.\n"
199    + "Java implementation Copyright (c) 1999 - 2012 by James M. Bieman\n" );
200 AdventureGame theGame = new AdventureGame();
201 theGame.startQuest();
202 }
203
204}
205
206
```

```java
package AdventureGame;

/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main

**/


public class Player {

  private Room myLoc;

  private Item[] myThings = new Item[2];

  private int itemCount = 0;

  public void setRoom(Room r){
   myLoc = r;
   }

  public String look() {
   return myLoc.getDesc();
   }

  public void go(int direction){
     myLoc.exit(direction,this);
  }

  public void pickUp(Item i){
   if (itemCount < 2) {
      myThings[itemCount] = i;
      itemCount++;
      myLoc.removeItem(i);
      }
  }

  public boolean haveItem(Item itemToFind){
     for (int n = 0; n < itemCount ; n++)
       if (myThings[n] == itemToFind) return true;
     return false;
  }

  public void drop(int itemNum){
   if (itemNum > 0 & itemNum <= itemCount){
      switch(itemNum){
      case 1: { myLoc.addItem(myThings[0]);
            myThings[0]=myThings[1];
            itemCount--;
            break;
          }
      case 2: { myLoc.addItem(myThings[1]);
        itemCount--;
        break;
```

```
59          }
60       }
61    }
62    }
63
64   public void setLoc(Room r){myLoc = r;}
65
66   public Room getLoc(){return myLoc;}
67
68   public String showMyThings(){
69    String outString = "";
70    for (int n = 0; n < itemCount ; n++)
71      outString = outString + Integer.toString(n+1) + ": "
72        + myThings[n].getDesc() + " ";
73    return outString;
74   }
75
76   public boolean handsFull(){return itemCount==2;}
77
78   public boolean handsEmpty(){return itemCount==0;}
79
80   public int numItemsCarried(){return itemCount;}
81
82}
83
84
```

```java
package AdventureGame;


/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main


                    **/

// interface CaveSite

public interface CaveSite{
  void enter(Player p);
}


```

```java
package AdventureGame;


/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main

**/

/**  Adventure Game  Program Code
Copyright (c) 1999-2012 James M. Bieman
The Adventure game is based on the "Colossal Cave Adventure" originally
designed by Will Crowther and implemented by Will Crowther
and Don Wood in Fortran in 1975 and 1976.

This micro-version is a variant of the original cave system and is implemented in Java
with just a few rooms and with a much more limited vocabulary.

Updated August 2010, January 2012
- Code is put into package cs314.a2 to match current CS314 coding standards.
Updated January 2012
- Renamed as the "Adventure Game"

To compile: javac cs314.a2.AdventureGame.java
To run:     java cs314.a2.AdventureGame

The main routine is AdventureGame.main

                **/

/** class Adventure: Primary method, createCave, creates the cave system.
        It eventually be replaced with a more flexible mechanism
        to support input and output from devices other than
        an ASCII terminal.

        Room descriptions are followed by a room identifier,
        to ease debugging and testing.  These would be removed
        to help confuse the user, which is our ultimate aim.

        I haven't added all of the room descriptions.  They
        will be done later.

        In this version all I/O is through standard I/O;
        I/O is to and from the console.

*/

public class Adventure {

  private Room entrance;

  public Room createAdventure(){
    // The outside:
      Room  outside = new Room();
```

```
59      outside.setDesc(
60        "You are standing outside, on the edge of a cliff;\n" +
61    " A creek runs alongside the cliff.\n" +
62    "a cave opens straight down (outside).");
63
64   // Room 1:
65      Room r1 = new Room();
66      r1.setDesc(
67     "The darkness is pierced by a bright light overhead.\n"
68     + "There is a narrow, dark passage to the east (r1)." );
69
70   // Connect the outside to Room 1:
71      outside.setSide(5,r1);
72      r1.setSide(4,outside);
73      entrance = outside;
74
75   // Room 2:
76      Room r2 = new Room();
77      r2.setDesc(
78    "You are in a gloomy oval shaped room with grey walls.\n" +
79     "There is a dim light to the west, and a narrow\n" +
80     "dark hole to the east only about 18 inches high (r2).");
81
82  // Room 3:
83     Room r3 = new Room();
84     r3.setDesc("You really need your flashlight here. \n"+
85        "There is a wide passage that quickly narrows\n"
86        +"to the west, a bright opening to the east,\n"
87        + "and a deep hole that appears to have no bottom\n"
88        + "in the middle of the room (r3).");
89
90  // Connect Rooms 1, 2, & 3:
91     r1.setSide(2,r2);
92     r2.setSide(3,r1);
93     r2.setSide(2,r3);
94     r3.setSide(3,r2);
95
96  // Room 4:
97     Room r4 = new Room();
98     r4.setDesc("There is what looks like a giant grizzly bear\n"
99        + "skull in a corner.  A passage leads to the west,\n"
100       + "another one to the north, and a slippery route\n"
101       + "goes down steeply. You can hear the shrieks of bats (r4).");
102
103 // Room 5:
104    Room r5 = new Room();
105    r5.setDesc("There is a dim light from above and the shrieks\n"
106               + "are clearly coming from a passageway to the east (r5).");
107
108 // Room 6:
109    Room r6 = new Room();
110    r6.setDesc("The ceiling is full of bats.\n"
111                + "You should put your hat on your head (r6).");
112
113 // Room 7:
114    Room r7 = new Room();
115    r7.setDesc("This room is very damp. There are puddles on the floor\n" +
116           "and a steady dripping from above (r7).");
```

```
117
118  // Connect rooms 3, 4, 5, 6, & 7.
119     r3.setSide(2,r4);
120     r3.setSide(5,r5);
121     r4.setSide(3,r3);
122     r4.setSide(5,r7);
123     r5.setSide(4,r3);
124     r5.setSide(2,r6);
125     r6.setSide(3,r5);
126     r7.setSide(4,r4);
127
128  // Room 8:
129     Room r8 = new Room();
130     r8.setDesc("A lizard scampers past you, or is it a snake?\n" +
131            "a narrow passage runs to the east and an evin narrower one\n" +
132            "runs to the west (r8).");
133
134  // Room 9:
135     Room r9 = new Room();
136     r9.setDesc("Room r9.");
137
138  // Room 10:
139     Room r10 = new Room();
140     r10.setDesc("It looks like someone has been here.\n" +
141            "There is a pile of candy wrappers on the floor,\n" +
142            "and maybe something else. \n" +
143            "Wait, there is a trap door on the floor,\n" +
144            "but it is locked (r10).");
145
146 // Room 11:
147    Room r11 = new Room();
148    r11.setDesc("This room is very dark. You can just barely see (r11).");
149    Treasure theTreasure = new Treasure();
150    theTreasure.setDesc("A bag filled with gold bars.");
151    r11.addItem(theTreasure);
152
153 // Lets connect them:
154    r4.setSide(0,r8);
155    r8.setSide(1,r4);
156    r8.setSide(3,r9);
157    r8.setSide(2,r10);
158    r9.setSide(2,r8);
159    r10.setSide(3,r8);
160
161 // Create a key and put it in r6:
162    Key theKey = new Key();
163    theKey.setDesc("A shiny gold key.");
164    r6.addItem(theKey);
165
166 // We add a door between r10 and r11:
167    Door theDoor = new Door(r10,r11,theKey);
168    r10.setSide(5,theDoor);
169    r11.setSide(4,theDoor);
170
171 // Now return the entrance:
172    entrance = outside;
173    return entrance;
174
```

```
175   }
176}
177
178
```

```java
package AdventureGame;

/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main

**/

// class Door

public class Door implements CaveSite {
  /** In this implementation doors are always locked.
      A player must have the correct key to get through
      a door.  Doors automatically lock after a player
      passes through. */

  private Key myKey;

  /** The door's location. */
  private CaveSite outSite;
  private CaveSite inSite;

  /** We can construct a door at the site. */
  Door(CaveSite out, CaveSite in, Key k){
    outSite = out;
    inSite = in;
    myKey = k;
  }

 /** A player will need the correct key to enter. */
 public void enter(Player p){
 if (p.haveItem(myKey)) {
    System.out.println("Your key works! The door creaks open,");
    System.out.println("and slams behind you after you pass through.");
    if (p.getLoc() == outSite) inSite.enter(p);
    else if (p.getLoc() == inSite) outSite.enter(p);
 }
 else {System.out.println("You don't have the key for this door!");
       System.out.println("Sorry.");
      }
 }

}


```

```java
package AdventureGame;

/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main

     Update August 2010: refactored Vector contents into ArrayList<Item> contents.
      This gets rid of the use of obsolete Vector and makes it type safe.

**/

// class Room


import java.util.ArrayList;
import java.util.ListIterator;
import java.util.Enumeration;
import java.util.Vector;


public class Room implements CaveSite {

  private String description;

  private CaveSite[] side = new CaveSite[6];

  private ArrayList<Item> contents = new ArrayList<Item>();

  Room() {
    side[0] = new Wall();
    side[1] = new Wall();
    side[2] = new Wall();
    side[3] = new Wall();
    side[4] = new Wall();
    side[5] = new Wall();
    }

  public void setDesc(String d){
    description = d;
    }

  public void setSide(int direction, CaveSite m){
   side[direction] = m;
   }

  public void addItem(Item theItem){
   contents.add(theItem);
   }

  public void removeItem(Item theItem){
   contents.remove(theItem);
   }

  public boolean roomEmpty(){
```

```
59      return contents.isEmpty();
60    }
61
62    public Item[] getRoomContents(){
63     Item[] contentsArray = new Item[contents.size()];
64     contentsArray = contents.toArray(contentsArray);
65     return contentsArray;
66    }
67
68
69    public void enter(Player p) {
70     p.setLoc(this);
71    }
72
73    public void exit(int direction, Player p){
74     side[direction].enter(p);
75     }
76
77    public String getDesc(){
78     ListIterator<Item> roomContents = contents.listIterator();
79     String contentString = "";
80     while(roomContents.hasNext())
81       contentString =
82      contentString + (roomContents.next()).getDesc() + " ";
83
84       return description + '\n' + '\n' +
85       "Room Contents: " + contentString + '\n';
86     }
87
88  }
89
90
```

```java
package AdventureGame;

/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main

**/

// class Wall


public class Wall implements CaveSite {

 public void enter(Player p)
         {
   System.out.println("Ouch! That hurts.");
 }

}


```

```java
package AdventureGame;

/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main

**/


// class Item

public class Item {

  private String description;

  public void setDesc(String d){
      description = d;
  }

  public String getDesc(){
         return description;
  }

}
```

```java
package AdventureGame;

/**  Adventure Game  Program Code
     Copyright (c) 1999 James M. Bieman

     To compile: javac AdventureGame.java
     To run:     java AdventureGame

     The main routine is AdventureGame.main

**/


// class Key.

public class Key extends Item {
}


```

```
 1 package AdventureGame;
 2
 3 /**  Adventure Game  Program Code
 4      Copyright (c) 1999 James M. Bieman
 5
 6       To compile: javac AdventureGame.java
 7            To run:     java AdventureGame
 8
 9             The main routine is AdventureGame.main
10
11 **/
12
13
14 // class Treasure
15
16 public class Treasure extends Item {
17 }
18
19
```