



```
1 package src.com.gamingroom;
2
3 /**
4  * A simple class to hold information about a player
5  * <p>
6  * Notice the overloaded constructor that requires
7  * an id and name to be passed when creating.
8  * Also note that no mutators (setters) defined so
9  * these values cannot be changed once a player is
10 * created.
11 * </p>
12 * @author coce@snhu.edu
13 *
14 */
15 public class Player {
16     long id;
17     String name;
18
19     /*
20      * Constructor with an identifier and name
21      */
22     public Player(long id, String name) {
23         this.id = id;
24         this.name = name;
25     }
26
27     /**
28      * @return the id
29      */
30     public long getId() {
31         return id;
32     }
33
34     /**
35      * @return the name
36      */
37     public String getName() {
38         return name;
39     }
40
41     @Override
42     public String toString() {
43         return "Player [id=" + id + ", name=" + name + "];"
44     }
45 }
46
```

```
1 package src.com.gamingroom;
2
3 /**
4  * A simple class to hold information about a game
5  *
6  * <p>
7  * Notice the overloaded constructor that requires
8  * an id and name to be passed when creating.
9  * Also note that no mutators (setters) defined so
10 * these values cannot be changed once a game is
11 * created.
12 * </p>
13 *
14 * @author coce@snhu.edu
15 *
16 */
17 public class Game {
18     long id;
19     String name;
20
21     /**
22      * Hide the default constructor to prevent creating empty instances.
23      */
24     private Game() {
25     }
26
27     /**
28      * Constructor with an identifier and name
29      */
30     public Game(long id, String name) {
31         //deleted this()
32         this.id = id;
33         this.name = name;
34     }
35
36     /**
37      * @return the id
38      */
39     public long getId() {
40         return id;
41     }
42
43     /**
44      * @set the id
45      */
46     //added setId
47     public void setId(long id) {
48         this.id = id;
49     }
50 }
```

```
51  /**
52   * @return the name
53   */
54  public String getName() {
55      return name;
56  }
57
58  /**
59   * @set the name
60   */
61  public void SetName(String name) {
62      this.name = name;
63  }
64
65  @Override
66  public String toString() {
67
68      return "Game [id=" + id + ", name=" + name + "];
69  }
70
71 }
72
```

```
1 package src.com.gamingroom;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.Iterator;      //Added
6
7 /**
8  * A singleton service for the game engine
9  *
10 * @author coce@snhu.edu
11 */
12 public class GameService {
13
14     /**
15      * A list of the active games
16      */
17     private static List<Game> games = new ArrayList<Game>();
18
19     /**
20      * Holds the next game identifier
21      */
22     private static long nextGameId = 1;
23
24     // FIXME: Add missing pieces to turn this class a singleton
25     public static GameService GS_Singleton = null; //declare and set to null
26
27     public static GameService getInstance() {
28         if (GS_Singleton == null) {
29             GS_Singleton = new GameService(); //if null then singleton is NEW
30         }
31         return GS_Singleton;
32     }
33
34     /**
35      * Construct a new game instance
36      *
37      * @param name the unique name of the game
38      * @return the game instance (new or existing)
39      */
40     public Game addGame(String name) {
41
42         // a local game instance
43         Game game = null;
44
45         // FIXME: Use iterator to look for existing game with same name
46         // if found, simply return the existing instance
47         Iterator iter = games.iterator();
48         while (iter.hasNext()) { //while games has next set 1st game to
            iter.next
```

```
49     Game gameOne = (Game)iter.next();
50     if (name.equalsIgnoreCase(gameOne.getName())) {
51         game = gameOne;
52     }
53 }
54 // if not found, make a new game instance and add to list of games
55
56 if (game == null) {
57     game = new Game(nextGameId++, name);
58     games.add(game);
59 }
60
61 // return the new/existing game instance to the caller
62 return game;
63 }
64
65 /**
66  * Returns the game instance at the specified index.
67  * <p>
68  * Scope is package/local for testing purposes.
69  * </p>
70  * @param index index position in the list to return
71  * @return requested game instance
72  */
73 Game getGame(int index) {
74     return games.get(index);
75 }
76
77 /**
78  * Returns the game instance with the specified id.
79  *
80  * @param id unique identifier of game to search for
81  * @return requested game instance
82  */
83 public Game getGame(long id) {
84
85     // a local game instance
86     Game game = null;
87
88     // FIXME: Use iterator to look for existing game with same id
89     Iterator iter = games.iterator();
90     while (iter.hasNext()) { //while games has next
91         Game gameOne = (Game) iter.next();
92         // if found, simply assign that instance to the local variable
93         if (gameOne.getId() == id) {
94             game = gameOne;
95         }
96     }
97     return game;
98 }
```

```
99
100 /**
101  * Returns the game instance with the specified name.
102  *
103  * @param name unique name of game to search for
104  * @return requested game instance
105  */
106 public Game getGame(String name) {
107     // a local game instance
108     Game game = null;
109
110     // FIXME: Use iterator to look for existing game with same name
111     Iterator iter = games.iterator();
112     while (iter.hasNext()) {
113         Game gameOne = (Game) iter.next();
114         if (name.equalsIgnoreCase(gameOne.getName())) {
115             game = gameOne;
116         }
117     }
118     // if found, simply assign that instance to the local variable
119     return game;
120 }
121
122
123
124 /**
125  * Returns the number of games currently active
126  *
127  * @return the number of games currently active
128  */
129 public int getGameCount() {
130     return games.size();
131 }
132 }
133
```

```
1 package src.com.gamingroom;
2
3 /**
4  * A simple class to hold information about a team
5  * <p>
6  * Notice the overloaded constructor that requires
7  * an id and name to be passed when creating.
8  * Also note that no mutators (setters) defined so
9  * these values cannot be changed once a team is
10 * created.
11 * </p>
12 * @author coce@snhu.edu
13 *
14 */
15 public class Team {
16     long id;
17     String name;
18
19     /*
20      * Constructor with an identifier and name
21      */
22     public Team(long id, String name) {
23         this.id = id;
24         this.name = name;
25     }
26
27     /**
28      * @return the id
29      */
30     public long getId() {
31         return id;
32     }
33
34     /**
35      * @return the name
36      */
37     public String getName() {
38         return name;
39     }
40
41     @Override
42     public String toString() {
43         return "Team [id=" + id + ", name=" + name + "]";
44     }
45 }
46
```



```
1 package src.com.gamingroom;
2
3 /**
4  * Application start-up program
5  *
6  * @author coce@snhu.edu
7  */
8 public class ProgramDriver {
9
10     /**
11      * The one-and-only main() method
12      *
13      * @param args command line arguments
14      */
15     public static void main(String[] args) {
16
17         // FIXME: obtain reference to the singleton instance
18         GameService service = GameService.getInstance(); // replace null with ???
19
20         System.out.println("\nAbout to test initializing game data...");
21
22         // initialize with some game data
23         Game game1 = service.addGame("Game #1");
24         System.out.println(game1);
25         Game game2 = service.addGame("Game #2");
26         System.out.println(game2);
27
28         // use another class to prove there is only one instance
29         SingletonTester tester = new SingletonTester();
30         tester.testSingleton();
31     }
32 }
33
```

```
1 package src.com.gamingroom;
2
3 /**
4  * A class to test a singleton's behavior
5  *
6  * @author coce@snhu.edu
7  */
8 public class SingletonTester {
9
10     public void testSingleton() {
11
12         System.out.println("\nAbout to test the singleton...");
13
14         // FIXME: obtain local reference to the singleton instance
15         GameService service = GameService.getInstance(); // replace null with
16         ???
17
18         // a simple for loop to print the games
19         for (int i = 0; i < service.getGameCount(); i++) {
20             System.out.println(service.getGame(i));
21         }
22     }
23 }
24 }
25
```