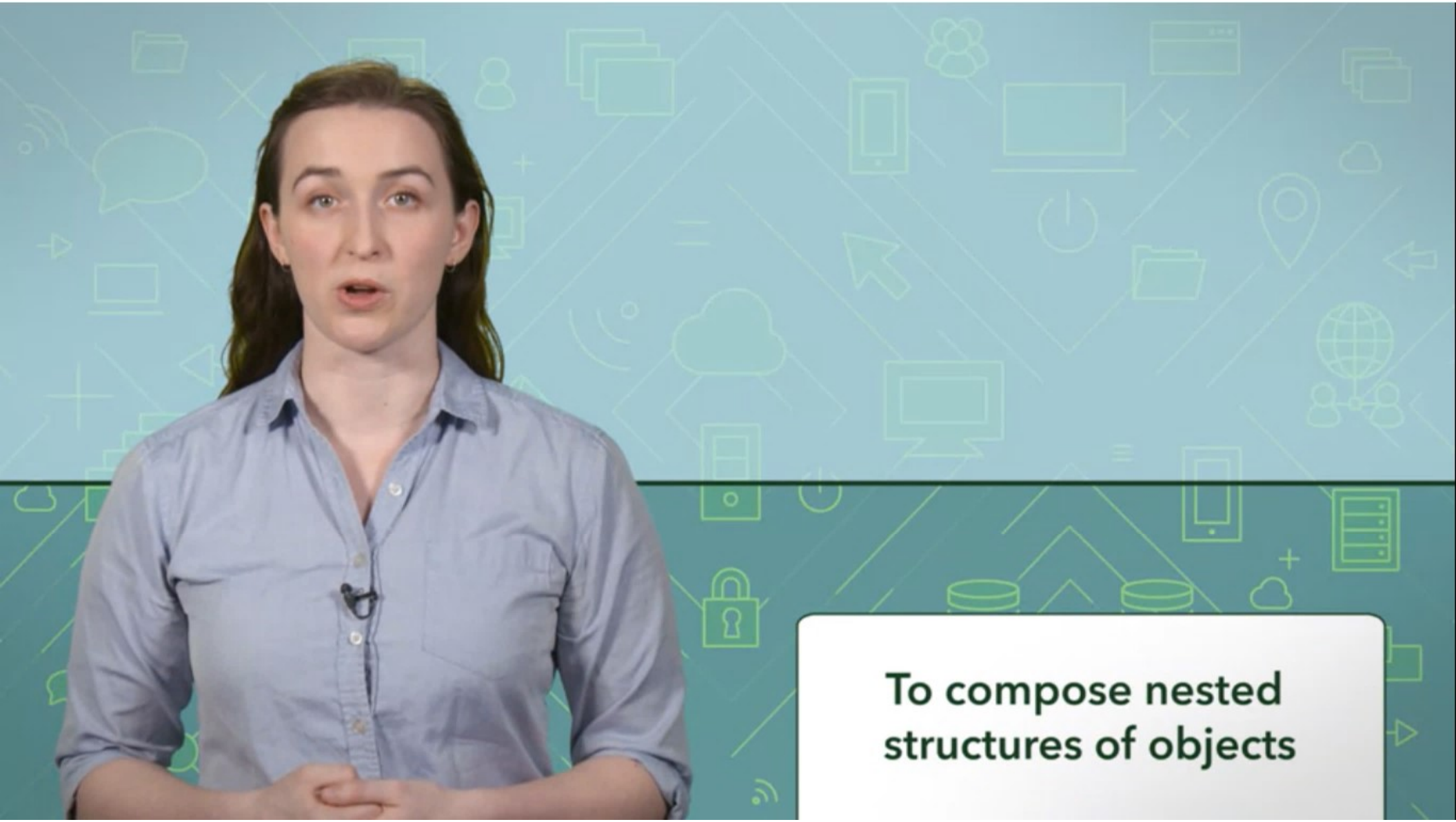
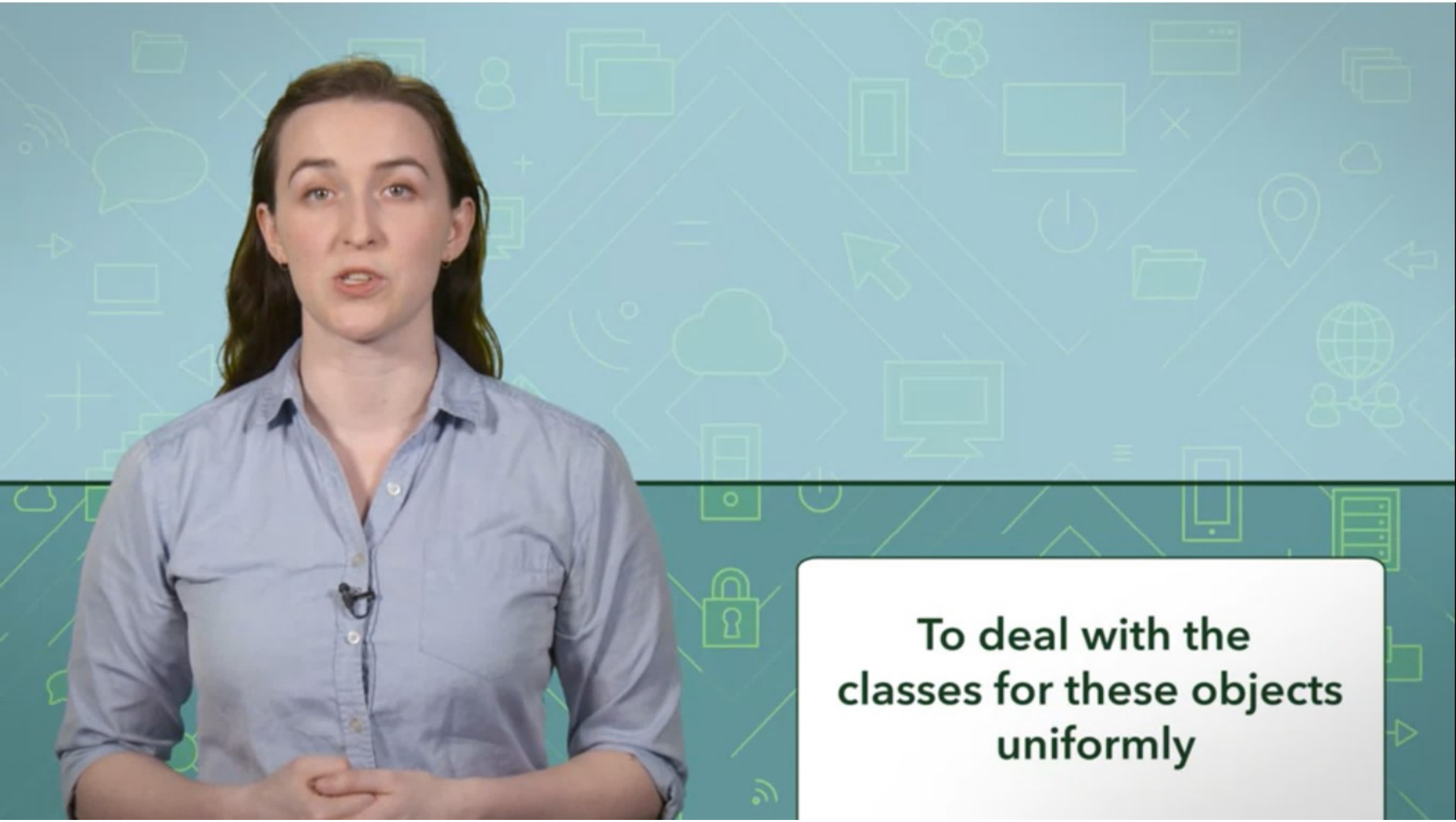
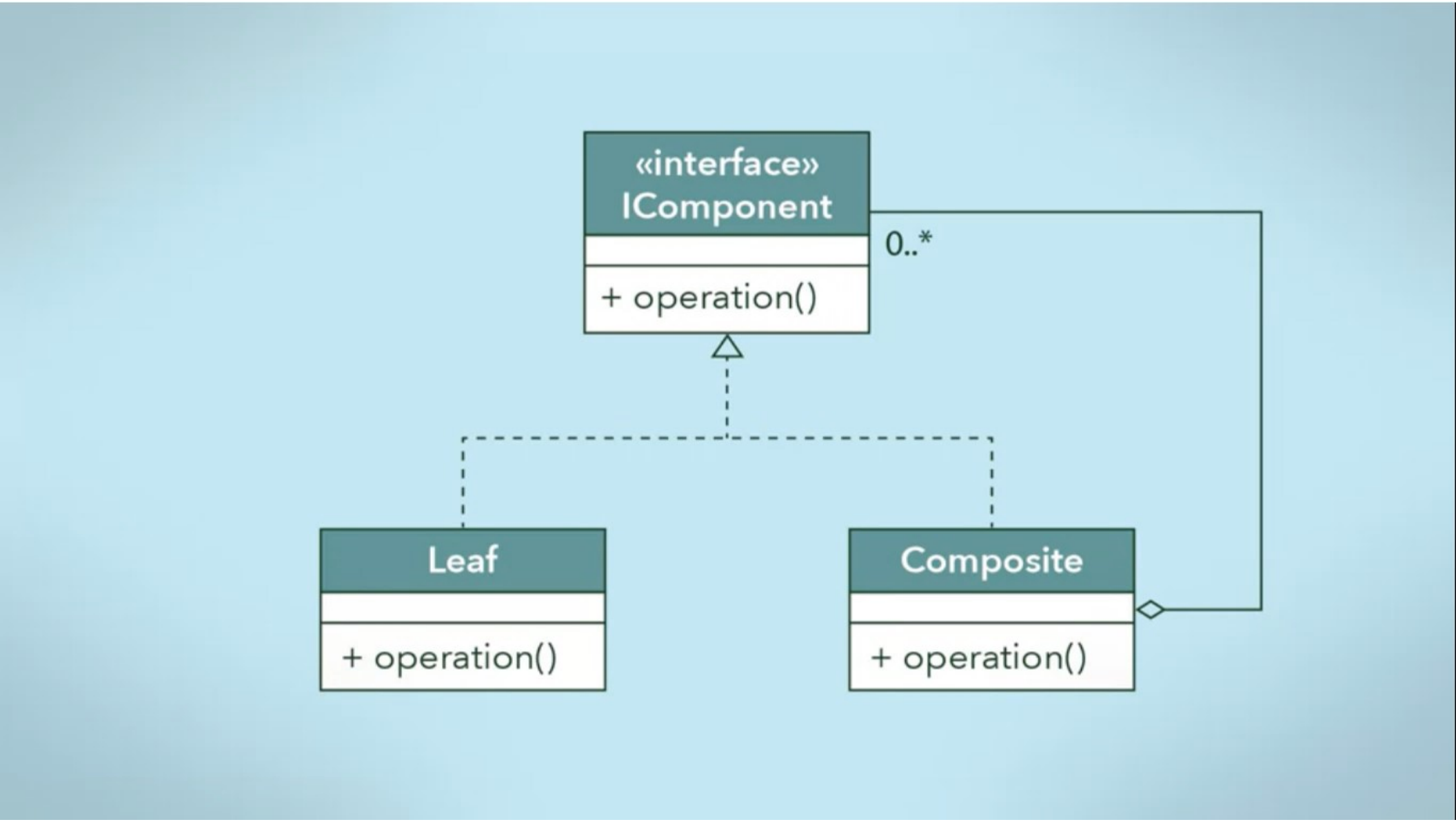
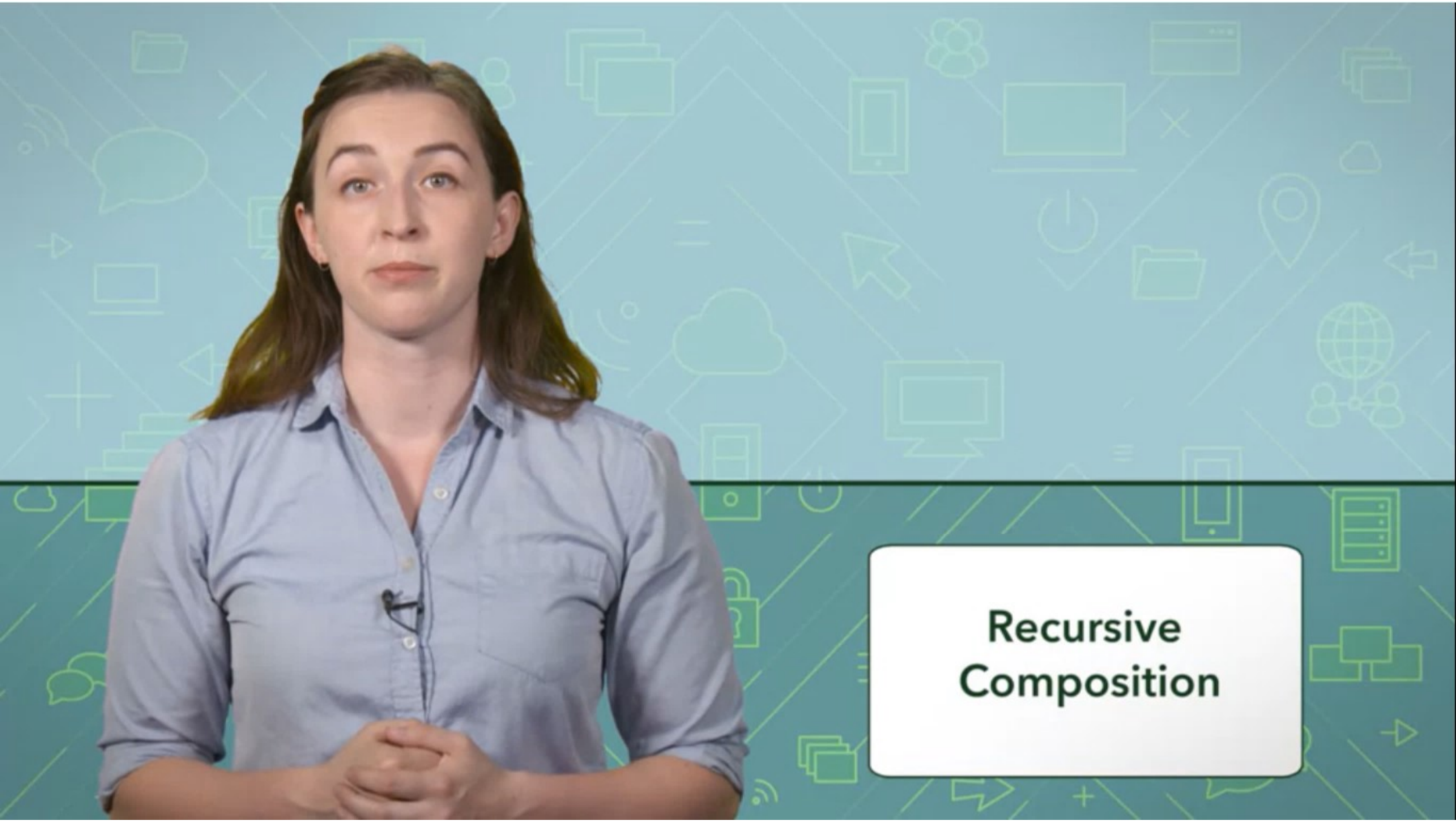

Design Patterns
Module 1

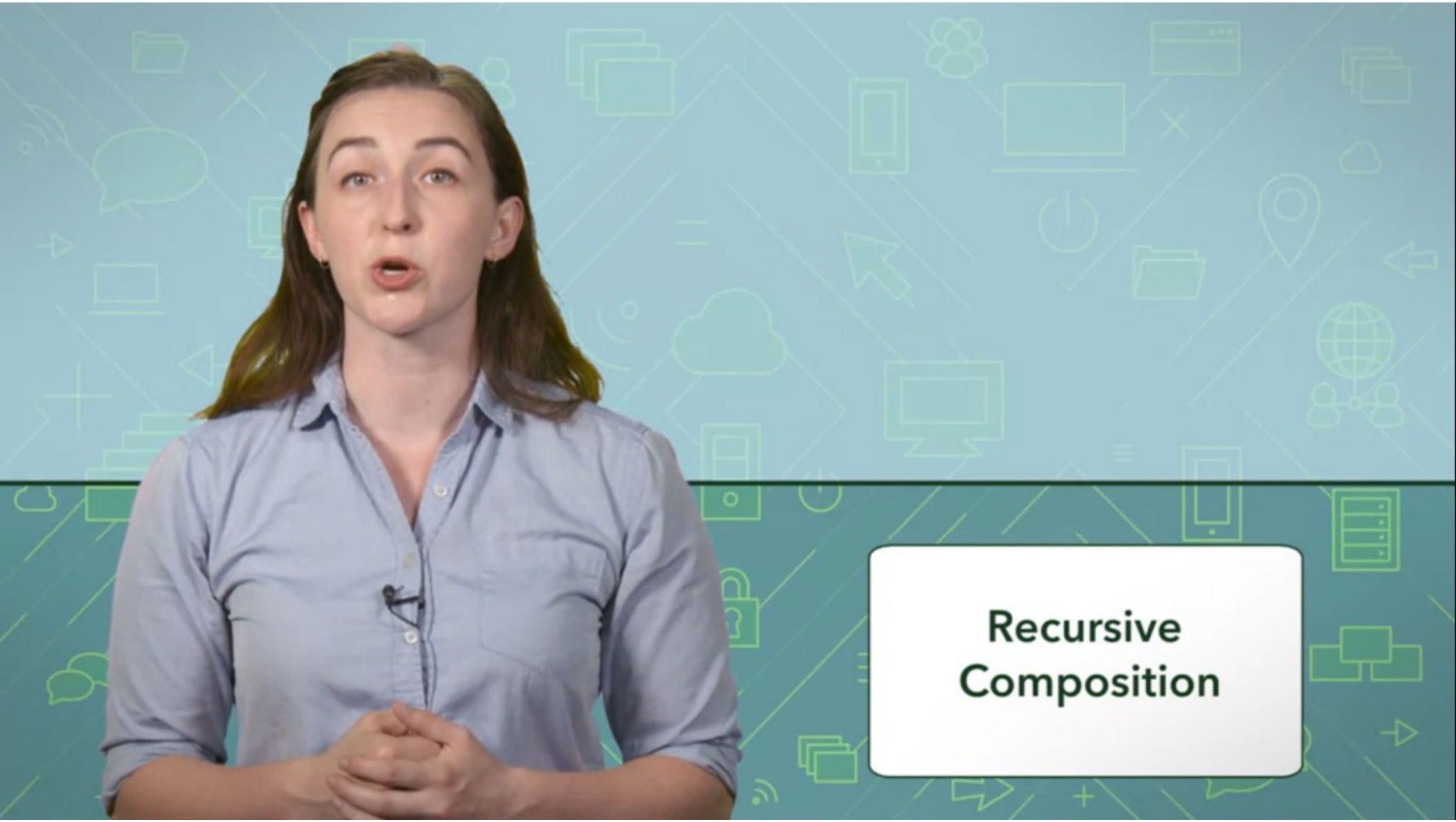
Composite Pattern

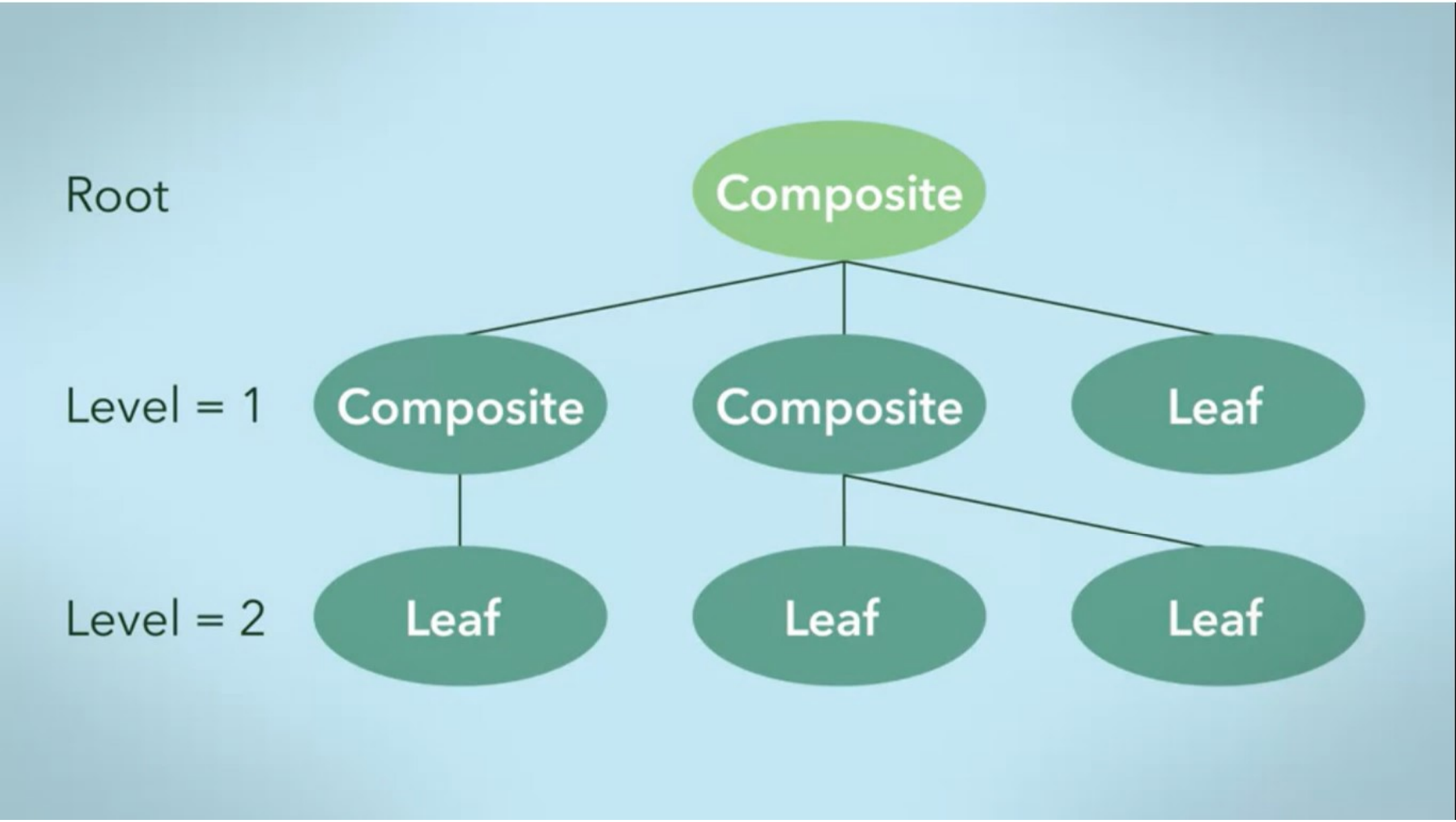

To compose nested
structures of objects

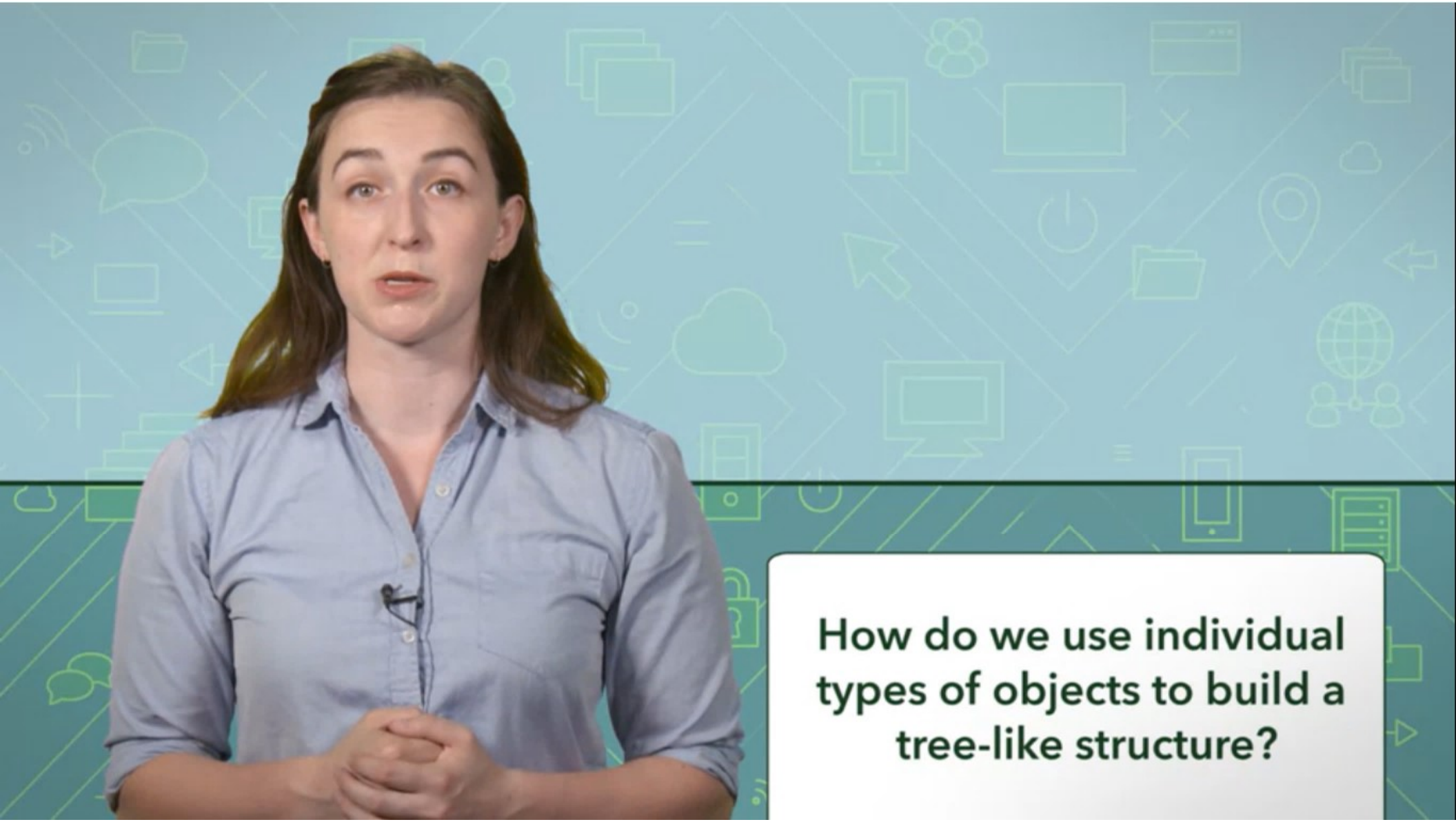

To deal with the
classes for these objects
uniformly

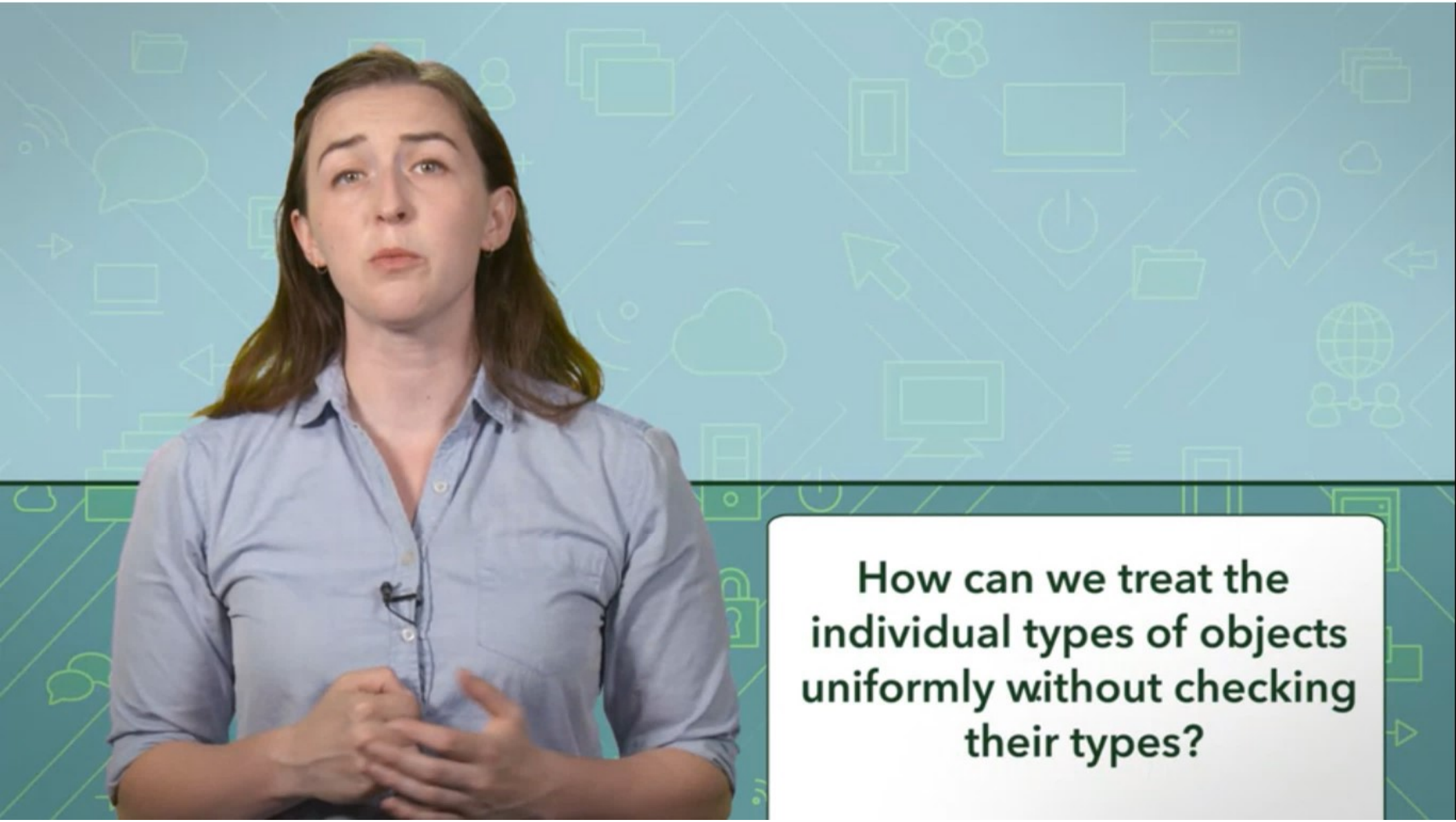

«interface»
IComponent
+ operation()
0..*
Leaf
+ operation()
Composite
+ operation()

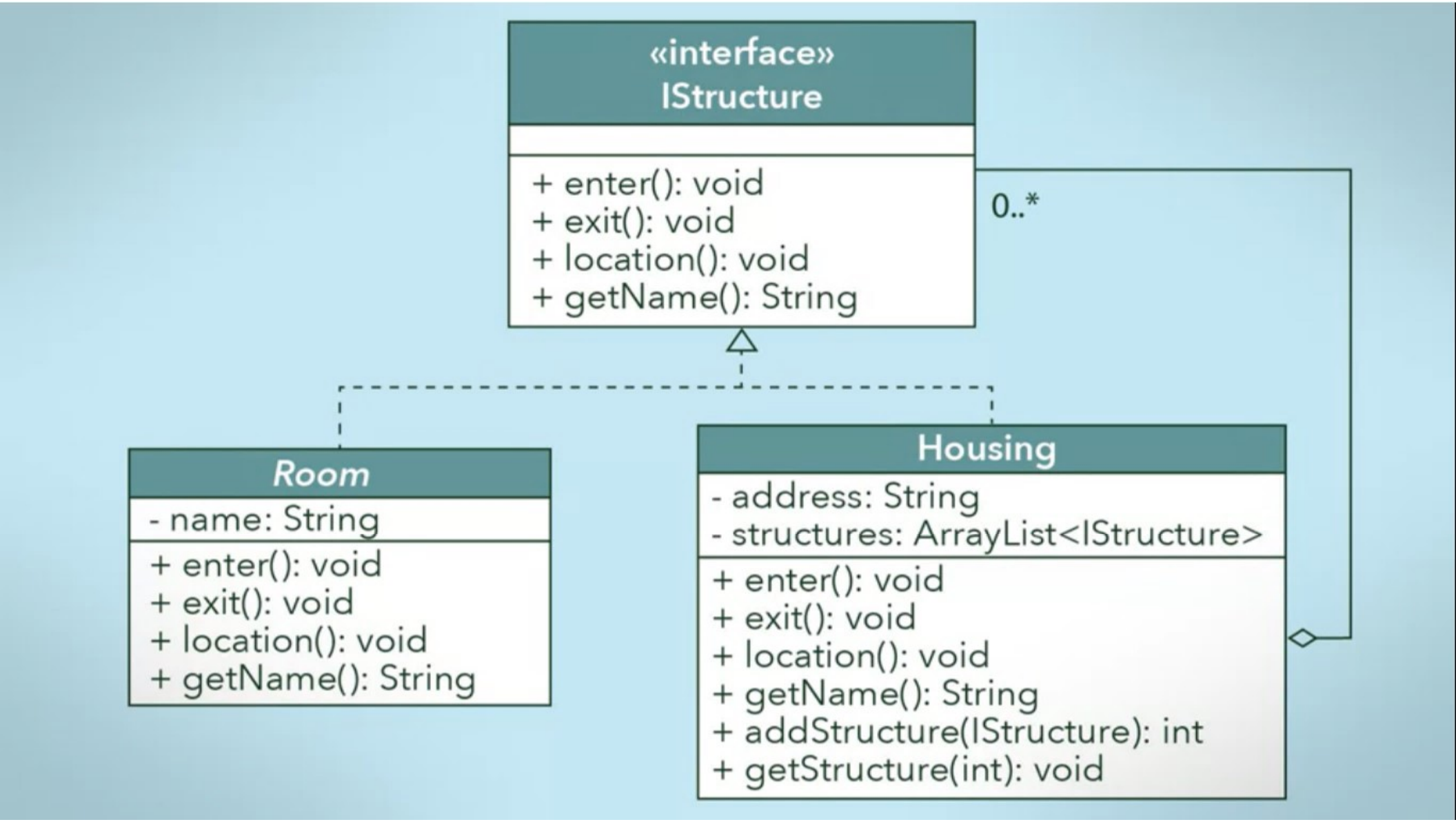

Recursive
Composition

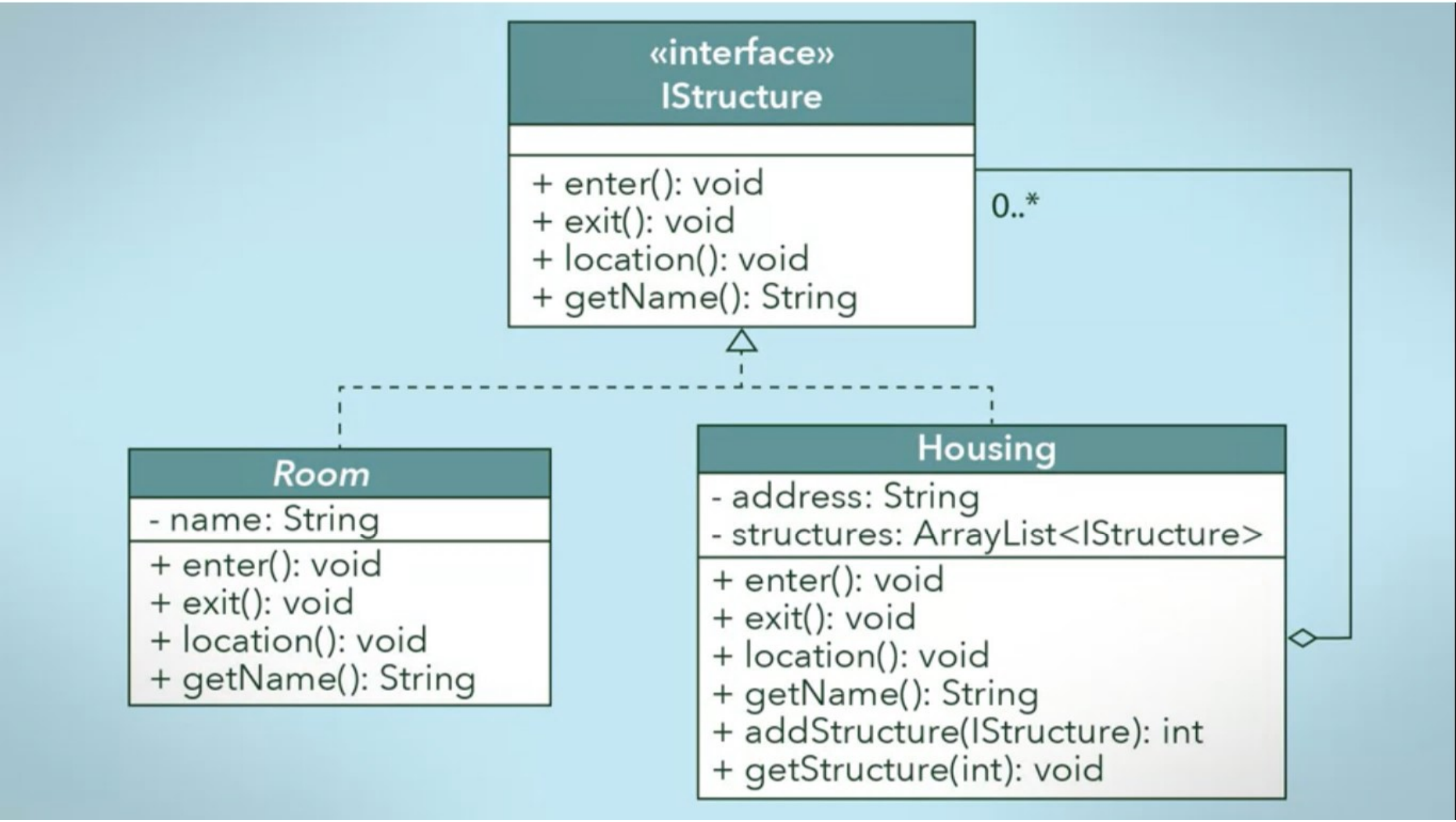

Recursive
Composition


Root
Composite
Level = 1
Composite
Composite
Leaf
Level = 2
Leaf
Leaf
Leaf


How do we use individual types of objects to build a tree-like structure?


How can we treat the individual types of objects uniformly without checking their types?


«interface»
IStructure
+ enter(): void
+ exit(): void
+ location(): void
+ getName(): String
0..*
Room
- name: String
+ enter(): void
+ exit(): void
+ location(): void
+ getName(): String
Housing
- address: String
- structures: ArrayList<IStructure>
+ enter(): void
+ exit(): void
+ location(): void
+ getName(): String
+ addStructure(IStructure): int
+ getStructure(int): void


«interface»
IStructure
+ enter(): void
+ exit(): void
+ location(): void
+ getName(): String
0..*
Room
- name: String
+ enter(): void
+ exit(): void
+ location(): void
+ getName(): String
Housing
- address: String
- structures: ArrayList<IStructure>
+ enter(): void
+ exit(): void
+ location(): void
+ getName(): String
+ addStructure(IStructure): int
+ getStructure(int): void

Step 1: Design the interface that defines the overall type

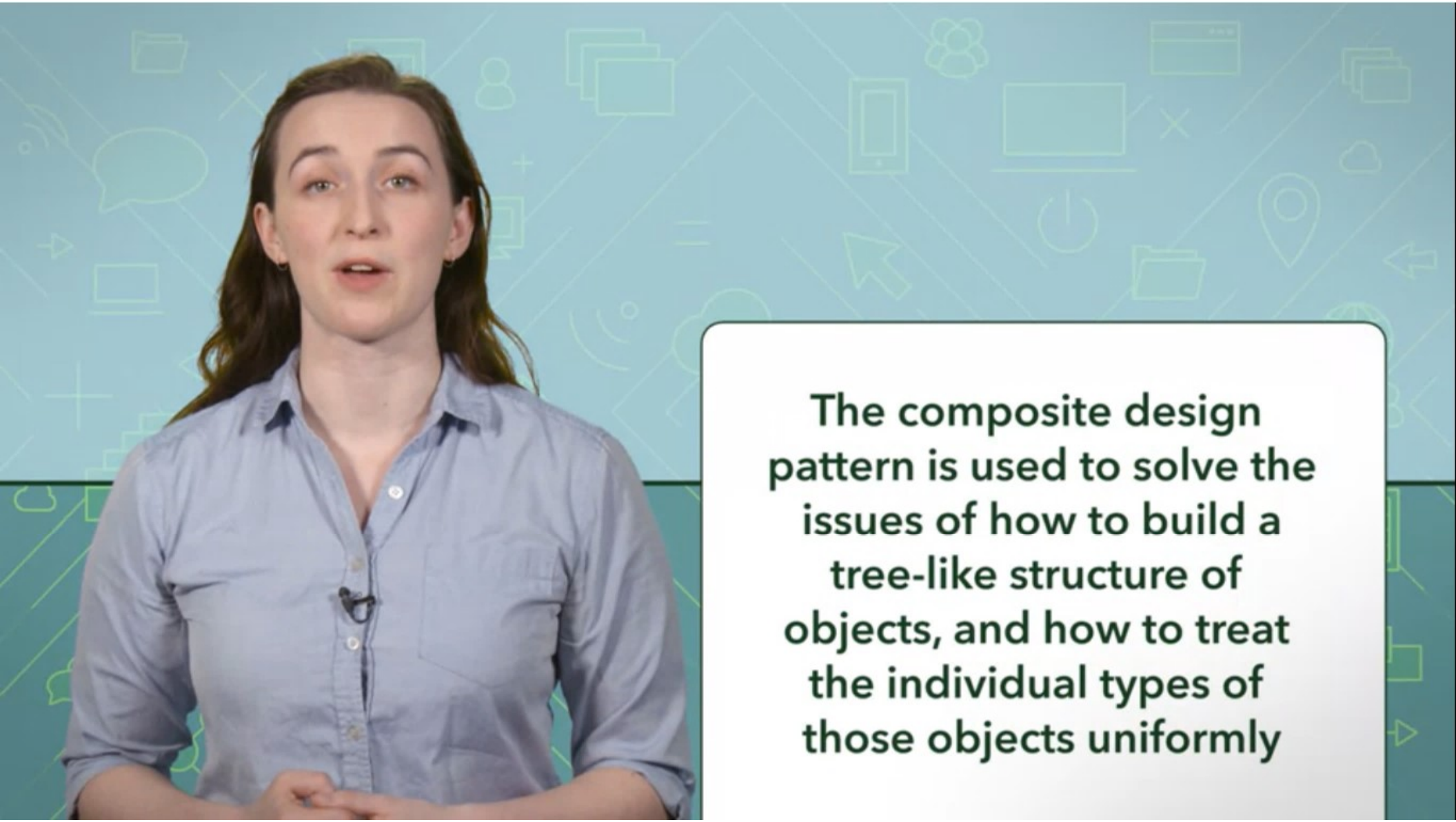
```
public interface IStructure {
     public void enter();
     public void exit();
     public void location();
     public String getName();
}
```

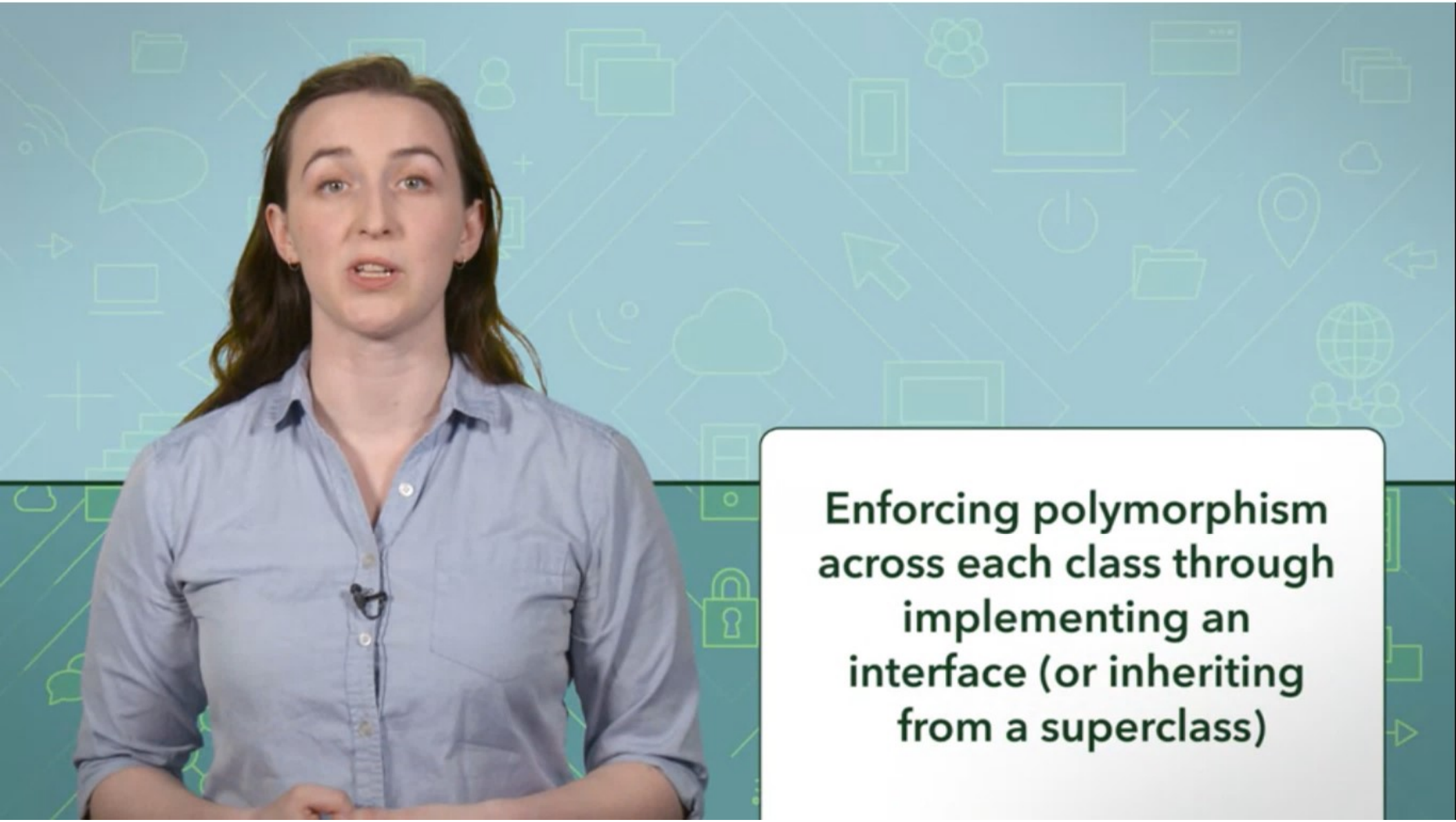
```
public class Housing implements IStructure {
     private ArrayList<IStructure> structures;
     private String address;

     public Housing (String address) {
          this.structures = new ArrayList<IStructure>();
          this.address = address;
     }

     public String getName() {
          return this.address;
     }

     public int addStructure(IStructure component) {
          this.structures.add(component);
          return this.structures.size() - 1;
     }

     public IStructure getStructure(int componentNumber) {
          return this.structures.get(componentNumber);
     }

     public void location() {
          System.out.println("You are currently in " + this.getName() +
               ". It has ");
          for (IStructure struct : this.structures)
               System.out.println(struct.getName());
     }

     /* Print out when you enter and exit the building */
     public void enter() { … }
     public void exit() { … }
}
```

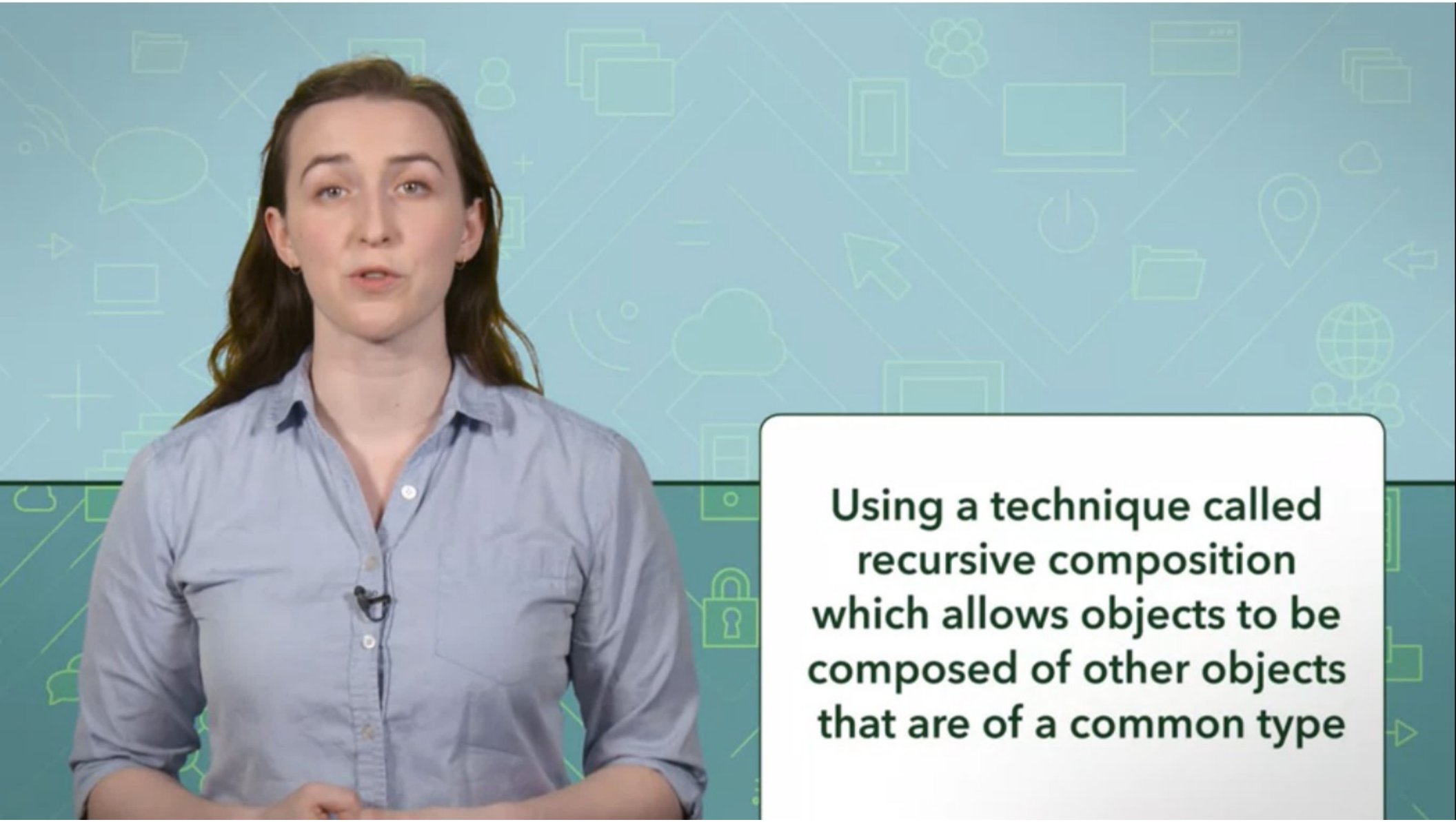
Step 3: Implement the leaf class

```
public abstract class Room implements IStructure {
     public String name;

     public void enter() {
          System.out.println("You have entered the " + this.name);
     }

     public void exit() {
          System.out.println("You have left the " + this.name);
     }

     public void location() {
          System.out.println("You are currently in the " + this.name);
     }

     public String getName() {
          return this.name;
     }
}
```

```
public class Program {

    public static void main(String args[]) {
        Housing building = new Housing("123 Street");
        Housing floor1 = new Housing("123 Street - First Floor");
        int firstFloor = building.addStructure(floor1);

        Room washroom1m = new Room("1F Men's Washroom");
        Room washroom1w = new Room("1F Women's Washroom");
        Room common1 = new Room("1F Common Area");

        int firstMens = floor1.addStructure(washroom1m);
        int firstWomans = floor1.addStructure(washroom1w);
        int firstCommon = floor1.addStructure(common1);

        building.enter(); // Enter the building
        Housing currentfloor = building.getStructure(firstFloor);
        currentFloor.enter(); // Walk into the first floor
        Room currentRoom = currentFloor.getStructure(firstMens);
        currentRoom.enter(); // Walk into the men's room
        currentRoom = currentFloor.getStructure(firstCommon);
        currentRoom.enter(); // Walk into the common area
    }
}
```


The composite design pattern is used to solve the issues of how to build a tree-like structure of objects, and how to treat the individual types of those objects uniformly


Enforcing polymorphism across each class through implementing an interface (or inheriting from a superclass)


Using a technique called
recursive composition
which allows objects to be
composed of other objects
that are of a common type

Practice Peer-graded Assignment: Ungraded Assessment – Composite Pattern

Ready for the assignment?
You will find instructions below to submit.

Instructions | My submission | Review classmates | Discussions

Learn how to apply the Composite pattern.

Review criteria less

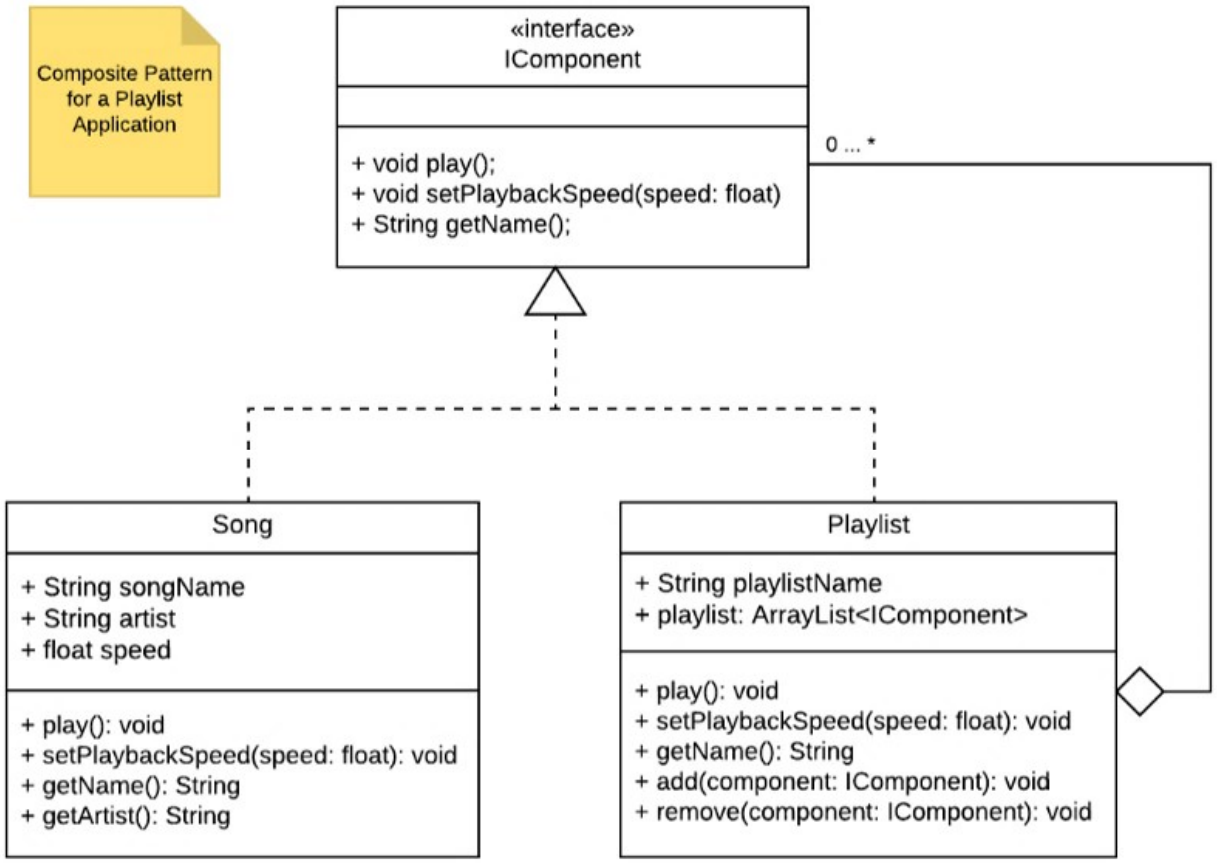
You have been asked to create a playlist application that will be used on Android devices (using the Java language). We will assume that each playlist can be composed of songs or other playlists, or a combination of both.

Your project manager has told you that the composite pattern is best used in this situation. The following UML class diagram that communicates the application's objects and relationships using the composite pattern.

In this assignment you are required to complete the provided code. (Note: With the exception of the Playlist class, you do not need to actually implement the methods, just write filler comments (eg., // play song). With the Playlist class, write out the method to add songs to the playlist).

UML Class Diagram less

Use the UML class diagram pictured below to help modify the provided code.



Code less

```
--------------------
[Program.java]
--------------------
public class Program {

  public static void main(String args[]) {

  // Make new empty "Study" playlist
  Playlist studyPlaylist = new Playlist("Study");

  // Make "Synth Pop" playlist and add 2 songs to it.
  Playlist synthPopPlaylist = new Playlist("Synth Pop");
  Song synthPopSong1 = new Song("Girl Like You", "Toro Y Moi" );
  Song synthPopSong2 = new Song("Outside", "TOPS");
  synthPopPlaylist.add(synthPopSong1);
  synthPopPlaylist.add(synthPopSong2);

  // Make "Experimental" playlist and add 3 songs to it,
  // then set playback speed of the playlist to 0.5x
  Playlist experimentalPlaylist = new Playlist("Experimental");
  Song experimentalSong1 = new Song("About you", "XXYYXX");
  Song experimentalSong2 = new Song("Motivation", "Clams Casino");
  Song experimentalSong3 = new Song("Computer Vision", "Oneohtrix Point Never");
  experimentalPlaylist.add(experimentalSong1);
  experimentalPlaylist.add(experimentalSong2);
  experimentalPlaylist.add(experimentalSong3);
  float slowSpeed = 0.5f;
  experimentalPlaylist.setPlaybackSpeed(slowSpeed);

  // Add the "Synth Pop" playlist to the "Experimental" playlist
  experimentalPlaylist.add(synthPopPlaylist);

  // Add the "Experimental" playlist to the "Study" playlist
  studyPlaylist.add(experimentalPlaylist);

  // Create a new song and set its playback speed to 1.25x, play this song,
  // get the name of glitchSong → "Textuell", then get the artist of this song →
    "Oval"
  Song glitchSong = new Song("Textuell", "Oval");
  float fasterSpeed = 1.25f;
  glitchSong.setPlaybackSpeed(fasterSpeed);
  glitchSong.play();
  String name = glitchSong.getName();
```

```
38    Song glitchSong = new Song("Textuell", "Oval");
39    float fasterSpeed = 1.25f;
40    glitchSong.setPlaybackSpeed(fasterSpeed);
41    glitchSong.play();
42    String name = glitchSong.getName();
43    String artist = glitchSong.getArtist();
44    System.out.println ("The song name is " + name );
45    System.out.println ("The song artist is " + artist );
46
47    // Add glitchSong to the "Study" playlist
48    studyPlaylist.add(glitchSong);
49
50    // Play "Study" playlist.
51    studyPlaylist.play();
52
53    // Get the playlist name of studyPlaylist → "Study"
54    System.out.println ("The Playlist's name is " + studyPlaylist.getName() );
55    }
56  }
57
58  --------------------
59  [IComponent.java]
60  --------------------
61  public interface IComponent {
62
63  // Your code goes here!
64
65
66
67  }
68
69  --------------------
70  [Playlist.java]
71  --------------------
72  public class Playlist implements IComponent {
73
74    public String playlistName;
75    public ArrayList<IComponent> playlist = new ArrayList();
76
77    public Playlist(String playlistName) {
78      this.playlistName = playlistName;
79    }
80
81    // Your code goes here!
82
83
```

```
}

--------------------
[Playlist.java]
--------------------
public class Playlist implements IComponent {

  public String playlistName;
  public ArrayList<IComponent> playlist = new ArrayList();

  public Playlist(String playlistName) {
    this.playlistName = playlistName;
  }

  // Your code goes here!


}

--------------------
[Song.java]
--------------------
public class Song implements IComponent {
  public String songName;
  public String artist;
  public float speed = 1; // Default playback speed

  public Song(String songName, String artist ) {
    this.songName = songName;
    this.artist = artist;
  }

  // Your code goes here!

}
```

```
--------------------
[Program.java]
--------------------
public class Program {

  public static void main(String args[]) {

  // Make new empty "Study" playlist
  Playlist studyPlaylist = new Playlist("Study");

  // Make "Synth Pop" playlist and add 2 songs to it.
  Playlist synthPopPlaylist = new Playlist("Synth Pop");
  Song synthPopSong1 = new Song("Girl Like You", "Toro Y Moi" );
  Song synthPopSong2 = new Song("Outside", "TOPS");
  synthPopPlaylist.add(synthPopSong1);
  synthPopPlaylist.add(synthPopSong2);

  // Make "Experimental" playlist and add 3 songs to it,
  // then set playback speed of the playlist to 0.5x
  Playlist experimentalPlaylist = new Playlist("Experimental");
  Song experimentalSong1 = new Song("About you", "XXYYXX");
  Song experimentalSong2 = new Song("Motivation", "Clams Casino");
  Song experimentalSong3 = new Song("Computer Vision", "Oneohtrix Point Never");
  experimentalPlaylist.add(experimentalSong1);
  experimentalPlaylist.add(experimentalSong2);
  experimentalPlaylist.add(experimentalSong3);
  float slowSpeed = 0.5f;
  experimentalPlaylist.setPlaybackSpeed(slowSpeed);

  // Add the "Synth Pop" playlist to the "Experimental" playlist
  experimentalPlaylist.add(synthPopPlaylist);

  // Add the "Experimental" playlist to the "Study" playlist
  studyPlaylist.add(experimentalPlaylist);

  // Create a new song and set its playback speed to 1.25x, play this song,
  // get the name of glitchSong → "Textuell", then get the artist of this song →
    "Oval"
  Song glitchSong = new Song("Textuell", "Oval");
  float fasterSpeed = 1.25f;
  glitchSong.setPlaybackSpeed(fasterSpeed);
  glitchSong.play();
  String name = glitchSong.getName();
  String artist = glitchSong.getArtist();
  System.out.println ("The song name is " + name );
  System.out.println ("The song artist is " + artist );
```

```
37    // get the name of glitchSong → "Textuell", then get the artist of this song →
        "Oval"
38    Song glitchSong = new Song("Textuell", "Oval");
39    float fasterSpeed = 1.25f;
40    glitchSong.setPlaybackSpeed(fasterSpeed);
41    glitchSong.play();
42    String name = glitchSong.getName();
43    String artist = glitchSong.getArtist();
44    System.out.println ("The song name is " + name );
45    System.out.println ("The song artist is " + artist );
46
47    // Add glitchSong to the "Study" playlist
48    studyPlaylist.add(glitchSong);
49
50    // Play "Study" playlist.
51    studyPlaylist.play();
52
53    // Get the playlist name of studyPlaylist → "Study"
54    System.out.println ("The Playlist's name is " + studyPlaylist.getName() );
55    }
56  }
57
58  --------------------
59  [IComponent.java]
60  --------------------
61  public interface IComponent {
62    void play();
63    void setPlaybackSpeed(float speed);
64    String getName();
65  }
66
67  --------------------
68  [Playlist.java]
69  --------------------
70  public class Playlist implements IComponent {
71
72    public String playlistName;
73    public ArrayList<IComponent> playlist = new ArrayList();
74
75    public Playlist(String playlistName) {
76      this.playlistName = playlistName;
77    }
78
79    public void add(IComponent component) {
80        playlist.add(component);
81  }
82
```

```
78
79    public void add(IComponent component) {
80         playlist.add(component);
81  }
82
83    public void remove(IComponent component) {
84      playlist.remove(component);
85    }
86
87    public void play(){
88      for(IComponent component : playlist) {
89        component.play();
90          }
91    }
92
93    public void setPlaybackSpeed(float speed) {
94          for(IComponent component : this.playlist){
95      component.setPlaybackSpeed(speed);
96        }
97    }
98
99    public String getName() {
100       return this.playlistName;
101   }
102 }
103
104 --------------------
105 [Song.java]
106 --------------------
107 public class Song implements IComponent {
108   public String songName;
109   public String artist;
110   public float speed = 1; // Default playback speed
111
112   public Song(String songName, String artist ) {
113     this.songName = songName;
114     this.artist = artist;
115   }
116
117   public void play() {
118     // Play the song using this.speed
119   }
120
121   public void setPlaybackSpeed(float speed) {
122     this.speed = speed;
123   }
124
```

```java
public class Song implements IComponent {
  public String songName;
  public String artist;
  public float speed = 1; // Default playback speed

  public Song(String songName, String artist ) {
    this.songName = songName;
    this.artist = artist;
  }

  public void play() {
    // Play the song using this.speed
  }

  public void setPlaybackSpeed(float speed) {
    this.speed = speed;
  }

  public String getName() {
    return this.songName;
  }

  public String getArtist() {
    return this.artist;
  }
}
```

This solution is provided for the "Composite Pattern" ungraded practice assignment. For greatest benefit, give the practice assignment your best effort and then compare your solution with the solution provided.

Mark as completed