ACTIVITY **4.1.2**

Level Loading

INTRODUCTION

In Activity 4.1.1, you saw a graphical room rendered on the screen of your device. You programmatically created and altered the configuration of tiles that formed the room. In this activity, you will manually create the configuration of tiles in the room in advance, storing it in a plaintext file. The app will then read this file to determine what to display where. After you do this, anyone can make a level for your game by simply adding some numerical values to a text file and adding it to the project structure.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or device emulator

Procedure

Part I: Load Static Data

1. Import *`4.1.2EmuOnTheLoose_StarterCode`* as instructed by your teacher.
2. Choose one of the following methods to upgrade Emu On The Loose for this activity. If you were unable to complete the previous activity, choose method two.
   - **Method One**: Use the Java files contained in *`4.1.2EmuOnTheLoose`* source files to overwrite the files of the same name in your project from *Activity 4.1.1*.
   - **Method Two**: Use the Java files contained in *`4.1.2EmuOnTheLoose`* source files to start a new LibGDX project. Follow steps 6-18 in *Activity 4.1.1 LibGDX Setup Part II Import Project.*
3. Open the file *`FloorMap.txt`*. Each 1 found in this file represents a wooden floor tile and each 0 represents a blue floor tile.

   What are the dimensions of the floor?

   Check your answer

   6 x 6

4. Add constants to your project for the width and height of this level, as well as for the numerical values associated with blue and wooden floor tiles.
5. Create a new class `LevelLoader`. This class will be used by `WorldModel` to get the model layer data from the text files that store it.
6. Add a load method to `LevelLoader`. `WorldModel` will use this method to get an `int[][]` from `LevelLoader`.
7. Use the skeleton code below to help you fill out the body of the load method. The goal is to take *412FloorMap.txt* and separate each line. Then from each line, read each `int` and store it in the 2D array from the previous step.

```
Scanner lineSeparator = null; // Make it convert from multi-line to single-line
// Declare a 2d array of type int identified by level, using your constants
// for level height and level width. Refer to sampleMap in WorldModel for
// an example of declaring and initializing a 2d array
try {
    FileHandle fileHandle = Gdx.files.internal("412FloorMap.txt");
    String fileAsString = new String(fileHandle.readString());
    lineSeparator = new Scanner(fileAsString);
    // Use Scanner methods that you find on the Java API website to help you:
    // -iterate through each line in the file with lineSeparator
    // -create a new Scanner for each line in the file
    // -use the new Scanner to collect each int in the line
    // -store each int in the appropriate location in the 2d array, level
    // -make sure to close the Scanner when you are done using it
} catch (Exception e) {
    Gdx.app.error(TAG, e.getMessage()); // Like Log but for LibGDX apps
} finally {
    // check to see if the Scanner you used to separate the lines is null
    // if it is not, close it. Closing a scanner ensures that no further
    // operations can be performed with it without reopening it.
}
return level;
```

8. Create a method `initSampleMap1` that uses your new `LevelLoader` class to determine which type of tile to render, calling the appropriate methods. In `WorldModel`, you may use `initSampleMap` as an example to help you write this new method.

In case your code does not produce the desired results when you run it, it will be useful to have some non-graphical output.

9. In the `finally` block of your code from Step 8, add a nested enhanced for loop and relevant code to output the contents of level.

   The map that renders should be missing a row and column of floor tiles.

10. Figure out why and fix it.

11. Change the dimensions of the level by modifying the *`.txt`* file to have an extra column of 1's.

    What happens when you run the game?

Part II: Load Dynamically with a Hack

12. Examine the contents of *`412FloorMap2.txt`*.

    What is the significance of the numbers 6 and 5?

    Check your answer

    6 and 5 indicate the dimensions of the floor.

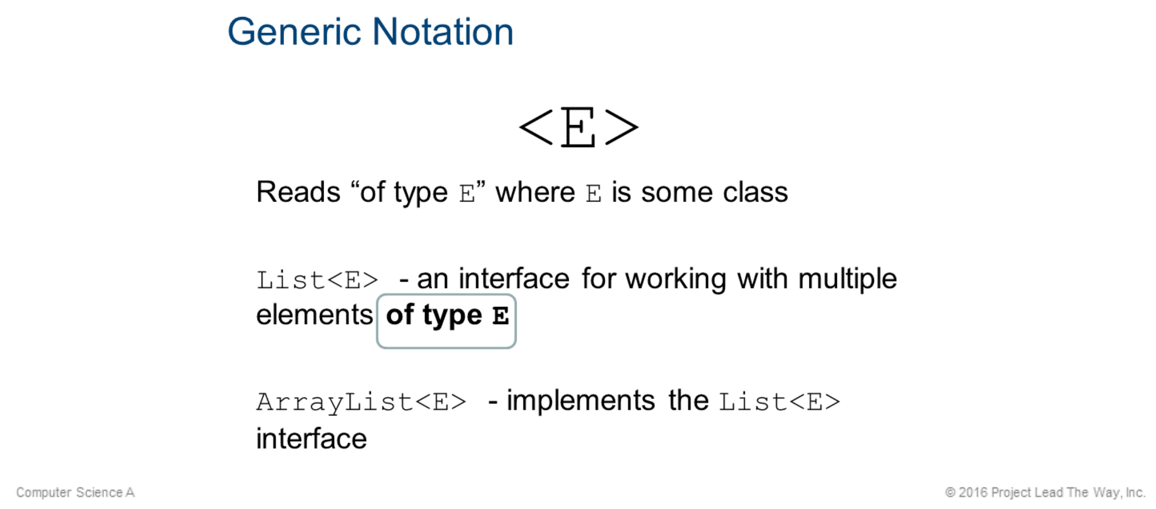
    Notice that the array of 0's and 1's is 6 x 5.

13. Modify your code from Part I of this activity so that it can use these two extra numbers to load maps of arbitrary size without needing to use constants for width and height.

14. Create your own floor map test file and verify that your code works correctly using it.

Part III: Load Dynamically with ArrayLists

15 Review the following slides.



Generic Notation

`<E>`

Reads "of type `E`" where `E` is some class

`List<E>` - an interface for working with multiple elements **of type `E`**

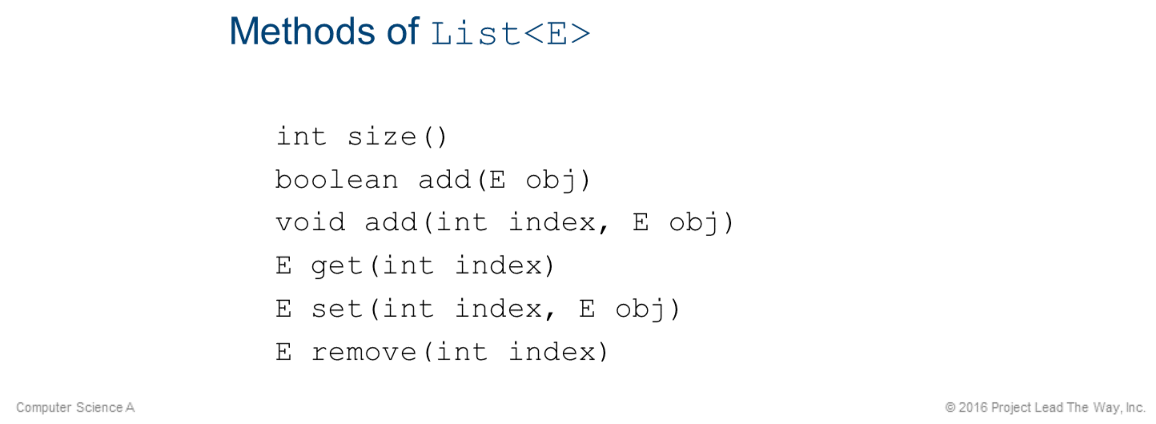
`ArrayList<E>` - implements the `List<E>` interface

Computer Science A © 2016 Project Lead The Way, Inc.

In earlier lessons, we used <T> to indicate a generic; however, you may also see <E> in official documentation. They are interchangeable.

The <E> notation is a way of generically saying "of type E". This means that the specific type E will be determined at a later time. This is handy when designing a class like ArrayList, whose purpose is to provide functionality related to storing and manipulating many elements of a specified type, because the client of the ArrayList class can decide what type they want E to be... it could be String, Integer, Object, or any other class or interface.

Note: List and ArrayList are both tested on the AP exam.



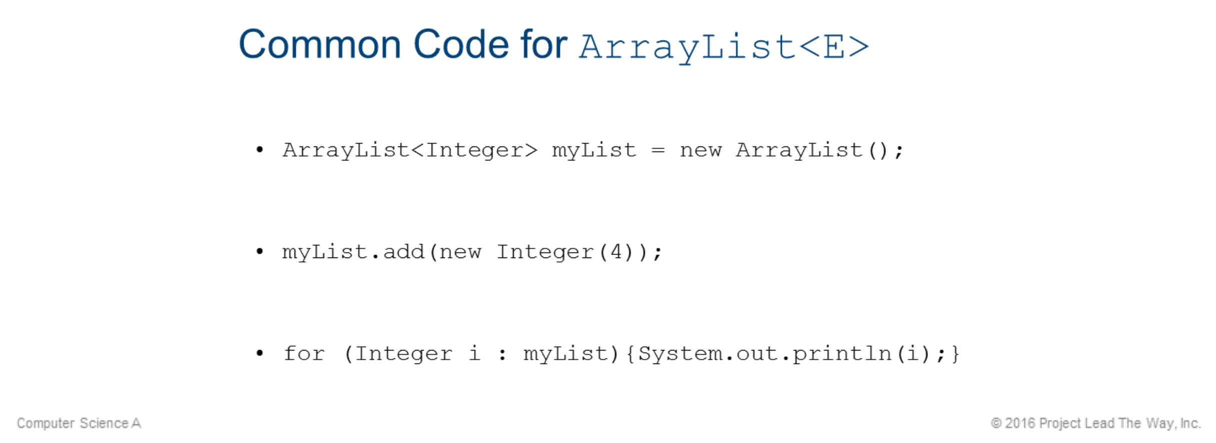
Methods of `List<E>`

```
int size()
boolean add(E obj)
void add(int index, E obj)
E get(int index)
E set(int index, E obj)
E remove(int index)
```

Computer Science A © 2016 Project Lead The Way, Inc.

The methods on this slide are the methods of the List<E> interface that are within the AP subset. For a complete list of methods of the List<E> interface and descriptions of each, go to: http://docs.oracle.com/javase/7/docs/api/java/util/List.html

- size - returns the number of elements in the list
- add - provides a method to insert an object of type E at the end of the list or at a specific index
- get - returns the element at the specified index in the list
- set - replaces the element at the given index with the one provided
- remove - takes the element at the given index out of the list and shifts the remaining elements to fill the empty space



Common Code for `ArrayList<E>`

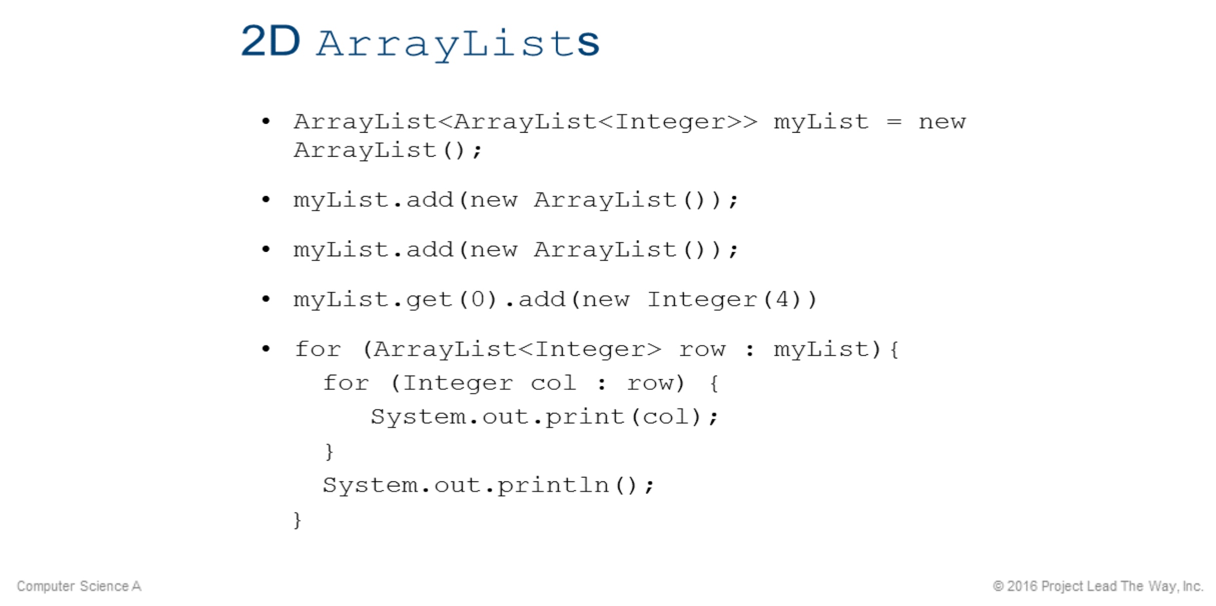
- `ArrayList<Integer> myList = new ArrayList();`
- `myList.add(new Integer(4));`
- `for (Integer i : myList){System.out.println(i);}`

Computer Science A

ArrayList implements List, and as such, implements all of the methods on the preceding slide and more.

The examples on this slide assume that <E> has been substituted by the client as <Integer> to make them more concrete.

- Declaring and initializing an ArrayList
- Add a new element to the ArrayList
- For each element of type Integer in myList, print that element



2D `ArrayLists`

-
  ```
  ArrayList<ArrayList<Integer>> myList = new
  ArrayList();
  ```
- `myList.add(new ArrayList());`
- `myList.add(new ArrayList());`
- `myList.get(0).add(new Integer(4))`

-
  ```
  for (ArrayList<Integer> row : myList){
    for (Integer col : row) {
        System.out.print(col);
    }
    System.out.println();
  }
  ```

Computer Science A

In Part III of Activity 4.1.2, you are asked to use a two-dimensional ArrayList. ArrayLists in two dimensions are not part of the AP subset, though they will be quite useful to you in making your game.

This slide contains ArrayList code to do the following:

- Initialize an ArrayList with two dimensions. At this step, myList is an empty ArrayList.
- Insert a new ArrayList at the end of myList; myList becomes a 1 row by 0 column structure.
- Insert a new ArrayList at the end of myList; myList becomes a 2 row by 0 column structure
- Retrieve the first ArrayList from myList and add Integer(4) to it; myList is now a 2-row structure with 1 column in the first row and 0 columns in the second. This is called a “jagged array” (even though technically it is a jagged ArrayList).
- Access the elements of myList, printing each row on a line and then advancing to the next line for the next row.

Any 2D ArrayList has the potential to be jagged.

16 Implement the `LevelLoader` so that it returns a 2D `ArrayList` that grows dynamically as the `LevelLoader` reads in data from the txt file.

CONCLUSION

1. When is it advantageous to use an `ArrayList` instead of an array?
2. When is it advantageous to use an array instead of an `ArrayList`?