Submission by Ren Taguchi (2999942)

Grade: 55.13%

Scratch space:

```
i: 3
j: 4
```

This test has a time limit of 100 minutes. This test will auto-save every ten seconds and submit automatically when the time expires.

- Multiple attempts are not allowed.
- Note that a problem may have more than one part. Be sure to read the entirety of every problem carefully and scroll to the end of each problem.
- You may use any of the Java techniques we have learned so far and only those.
- You are allowed to have one sheet of notes with you during the exam. You may also have scrap paper.
- Remember that the phrase “design a class” or “design a method” means more than just writing code. It means to design them according to the design recipe and the **course’s design guidelines**. You do not need to repeat any test cases given to you, but you should add tests wherever appropriate.
- You may write tests as "actual --> expected" or "error --> new RuntimeException" as a shortcut rather than writing complete test methods.
- To construct `ArrayLists`, you may use `Arrays.asList(item1, ..., itemN)`.
- You do **not** have to add templates explicitly for each method or class. However, if you are stuck on how to proceed, writing the template may help you.
- We will not answer any questions during the exam, except possibly to fix typos or errors that we may have overlooked.
- It is your job to make sure that you submit all of your answers within the allotted time. Answers will not be accepted after that by email or by any other means. If you have any Hourglass issues, you must send a message in Hourglass to the instructor immediately.
- Please remember to read and sign the honor pledge and enter the password for your lab section. We will not grade your exam if you do not have the password and sign the honor pledge.

Question 1: Honor pledge and password

Part A: Password — 0 / 0 points

Enter the password for this lab section that is shown on the whiteboard.

```
85719
```

Up to 0 points, counting up

0 points Has the correct password

(Leena Razzaq)

Part B: Pledge — 0 / 0 points

On my honor, I affirm that this exam represents my own work, and I did not use any outside assistance in completing it. I also affirm that I did not attempt to assist any other student with their exam.

Please sign your name (by typing it) and the time that you finished.

```
Ren Taguchi
```

At most 0 points, counting down

0 points Signed the pledge

(Leena Razzaq)

Question 2: ArrayLists and Loops

```
1 //signatures for methods of ArrayList<T> that can be used in this problem:
```

```
2
3  //Returns the size of the list
4  int size();
5
6  // EFFECT: Adds the given value at the end of the list
7  boolean add(T value);
8
9  // Returns the value at the given index
10 T get(int index);
11
12 // Returns the current value at the given index
13 // EFFECT: Updates the value at the given index to the given new value
14 T set(int index, T newValue);
15
16 //returns the element that was removed
17 //EFFECT: removes the element at the specified position in the list
18 T remove(int index);
19
```

Part A

10 / 15 points

Design the method:

```
<T> void noDupes(ArrayList<T> list1)
```

which modifies `list1` to remove any duplicate items.

You may assume the given list is not null and that this method will be in a class called `ArrayUtils`.

Here is an example to clarify:

Assume a list named `nums1` holds integers 1, 2, 2, 1 and 3

Then `noDupes` given `nums1` will mutate `nums1` to hold 1, 2 and 3

For full credit, you may only use the `ArrayList` methods shown above and you must use **counted for loops**. (It may be helpful to look ahead at part B of this problem.)

You should include examples and tests and you may use the shortcuts provided in the exam instructions.

```
1  <T> void noDupes(ArrayList<T> list1) {
2    for(int i = 0; i < list1.size(); i++) {
3      for(int j = 1; i < list1.size(); j++) {
4        if(list1.get(i).equals(list1.get(j))) {
5          list1.remove(i);
6        }
7      }
8    }
9  }
10
11 public class ExampleArrayDupes {
12   ArrayList<Integer> intList = new ArrayList<Integer>();
13   ArrayList<Integer> intListFix = new ArrayList<Integer>();
14   ArrayList<Character> charList = new ArrayList<Character>();
15   ArrayList<Character> charListFix = new ArrayList<Character>();
16   intList.add(new Integer(1));
17   intList.add(new Integer(2));
18   intList.add(new Integer(2));
19   intList.add(new Integer(1));
20   intList.add(new Integer(3));
21   charList.add(new Character("a"));
22   charList.add(new Character("b"));
23   charList.add(new Character("a"));
24   charList.add(new Character("c"));
```

```
25    intListFix.add(new Integer(1));
26    intListFix.add(new Integer(2));
27    intListFix.add(new Integer(3));
28    charListFix.add(new Character("a"));
29    charListFix.add(new Character("b"));
30    charListFix.add(new Character("c"));
31
32
33
34    //tests:
35    actual --> expected
36    noDupes(intList) --> intListFix
37    noDupes(charList) --> charListFix
38
39  }
40
```

At most 15 points, counting down

-2 points Does not correctly keep track of index after removals (Manasa Manjunath)

-1 point Logic is little incorrect In line 3, why is j always starting with 1, also why is the condition i < list1.size() and not j < list1.size() (Manasa Manjunath)

-1 point does not test with an empty list (Manasa Manjunath)

-1 point does not have an effect statement (Manasa Manjunath)

Part B

1 / 2 points

Explain which notion of sameness you used to compare values in the `noDupes` method and why.

I used .equals, since the T data type in the ArrayList is a pointer, so == would be comparing the pointers and not the values themselves.

At most 2 points, counting down

-1 point Used an acceptable type of comparison, but does not describe the notion of equality used correctly. You could use == or .equals. .equals will use intensional equality (comparing references) unless it has been overridden for the object. In which case it may use extensional equality. == always uses intensional equality for objects (Leena Razzaq)

Question 3: Iterators

9 / 15 points

```
1  // Represents the ability to produce a sequence of values
2  // of type T, one at a time
3  interface Iterator<T> {
4
5    // Does this sequence have at least one more value to produce?
6    boolean hasNext();
7
8    // Get the next value in this sequence
9    // EFFECT: Advance the iterator to the subsequent value
10   T next();
11
12 }
```

You are trying to implement HuskyVid, the next great competitor to YouTube. You've figured out that you need an `Iterator` to produce the sequence of frames of a video. You are now tasked with figuring out how to implement slow motion for videos. Design an Iterator that can be used to play a video by showing each frame a given number of times. To do this you will design a class:

```
class SlowMotionIter<X> implements Iterator<X>
```

whose constructor is given an `Iterator<X>` called `iter` when constructed. An `int` representing the number of times to repeat each value should also be passed into the constructor. You may not assume that `iter` will eventually have no more values.

For your reference, the `Iterator` interface is shown above.

You should add **valid** examples and tests for this problem. You may use the shortcuts provided in the exam instructions.

```
class SlowMotionIter<X> implements Iterator<X> {
  int curIdx;
  Iterator<X> iterator;
  int repeatNum;

  SlowMotionIter(Iterator<X> iter, int repeatNum) {
    this.curIdx = 0;
    this.repeatNum = repeatNum;
    this.iterator = iter;
  }

  // Does this sequence have at least one more value to produce?
  boolean hasNext() {
    return this.iterator.hasNext() || this.curIdx < this.repeatNum - 1;
  }

  // Get the next value in this sequence
  // EFFECT: Advance the iterator to the subsequent value
  T next(){
    if(!iter.hasNext()) {
      this.curIdx+=1;
      if(this.curIdx < this.repeatNum) {
        this.iter = this.iter.next().iterator();
      } else {
        this.iter = new ArrayList<X>().iterator();
      }
    }

    if(iter.hasNext()) {
      return iter.next();
    } else throw new IndexOutOfBoundsException("You have hit the end of this frame");
  }

}

class ExamplesSlowMotion {
  ArrayList<Integer> listInt = Arrays.asList(1, 2, 3);
  SlowMotionIter tenFrame = new SlowMotionIter(new Iterator<Integer>(), 10);
  for(Integer i: tenFrame) {
    int j = 0;
    actual --> expected
    i.equals(listInt.get(j)) --> true
    j++;
  }

}
```

At most 15 points, counting down

-1 point	does not return the current frame when it is being repeated	(Ha Ngo)
-1 point	does not construct a valid example for testing	(Ha Ngo)
-1 point	does not have at least one test for hasNext	(Ha Ngo)
-1 point	Does not have at least one test for next	(Ha Ngo)

-1 point does not check for whichever exception should be thrown when next is called and there are no more items (Ha Ngo)

-1 point should keep track of the current frame being produced (Ha Ngo)

Question 4: Debugging

Consider the provided code which attempts to create the board for the Fifteen game from Lab 10. The board was to be made of 4 rows of 4 tiles each, labeled with numbers 0-15 in a random order. However, there is a bug in the code, that does not create the board as intended. **Hint**: the bug has nothing to do with formatting, style or design recipe steps.

```
1  //Represents an individual tile
2  class Tile {
3    // The number on the tile.  Use 0 to represent the blank tile
4    int value;
5
6    public Tile(int value) {
7      this.value = value;
8    }
9  }
10
11 class FifteenGame extends World {
12   ArrayList<ArrayList<Tile>> board;
13   Random rand;
14
15   FifteenGame() {
16     this.rand = new Random();
17     this.board = this.initBoard();
18   }
19
20   //Constructs a list of lists of Tiles that contain the numbers 0-15.
21   //The list contains 4 lists each of which contains 4 Tiles
22   ArrayList<ArrayList<Tile>> initBoard() {
23     ArrayList<ArrayList<Tile>> tiles = new ArrayList<ArrayList<Tile>>();
24     ArrayList<Integer> numbers =
25         new ArrayList<Integer>(Arrays.asList(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15));
26
27     ArrayList<Tile> row = new ArrayList<Tile>();
28
29     for (int i = 0; i < 4; i++) {
30
31       for (int j = 0; j < 4; j++) {
32
33         int rIdx = this.rand.nextInt(numbers.size());
34         row.add(new Tile(numbers.get(rIdx)));
35         numbers.remove(numbers.get(rIdx));
36
37       }
38       tiles.add(row);
39
40       for (int k = 0; k < 4; k++) {
41         row.remove(0);
42       }
43     }
44
45     return tiles;
46   }
47
48   //implementations of event handlers omitted on purpose. They are not a source of
49   //the bug you need to identify.
50 }
```

Part A

0.5 / 3 points

For this part write a test that illustrates the problem, and in your own words describe what the bug is. A correctly written test should *fail* on the code as-provided, but should pass when you fix the code. This is for you to show that you have found the problematic scenario that prevents the board from being created correctly. In a comment above the test, please state the purpose of the test (i.e. the scenario you are testing).

```
FifteenGame game = new FifteenGame();

ArrayList<Tile> tiles = new ArrayList<Tile>();
for (int i = 0; i < 4; i++) {
    tiles = game.tiles.get(i).get(j);
}
// the initBoard() function never removes values from numbers correctly,
// since it's removing using a wrapper Integer instead of just an int,
// so it can add the same value twice
for (int i = 0; i < 4; i++) {
  for (int j = 0; j < 4; j++) {
    for(int k = 0; k < 4; k++) {
      actual --> expected
      tiles.get(k).equals(game.tiles.get(i).get(j)) --> true
    }
  }
}

// different test
Random rand = new Random();
ArrayList<Integer> numbers =
        new ArrayList<Integer>(Arrays.asList(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15));
int rIdx = this.rand.nextInt(numbers.size());
numbers.remove(numbers.get(rIdx));
// numbers should be smaller by 1 number,
// but instead is not changed since you are
// trying to remove at an index of an Integer(wrapped)
actual --> expected
numbers.size() --> 15
```

At most 3 points, counting down

-2.5 points all lists are size 0 due to unwanted aliasing/mutation in lines 27,38,41 (Bambi Zhuang)

Part B

1 / 4 points

Fix the bug in the code. You may comment out (but not remove) any code provided by us that you believe is buggy. You should clearly illustrate (with comments) which code you added. No need to write additional tests.

```
//Represents an individual tile
class Tile {
  // The number on the tile.  Use 0 to represent the blank tile
  int value;

  public Tile(int value) {
    this.value = value;
  }
}

class FifteenGame extends World {
  ArrayList<ArrayList<Tile>> board;
```

```
13  Random rand;
14
15  FifteenGame() {
16    this.rand = new Random();
17    this.board = this.initBoard();
18  }
19
20  //Constructs a list of lists of Tiles that contain the numbers 0-15.
21  //The list contains 4 lists each of which contains 4 Tiles
22  ArrayList<ArrayList<Tile>> initBoard() {
23    ArrayList<ArrayList<Tile>> tiles = new ArrayList<ArrayList<Tile>>();
24    ArrayList<Integer> numbers =
25        new ArrayList<Integer>(Arrays.asList(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15));
26
27    ArrayList<Tile> row = new ArrayList<Tile>();
28
29    for (int i = 0; i < 4; i++) {
30
31      for (int j = 0; j < 4; j++) {
32
33        int rIdx = this.rand.nextInt(numbers.size());
34        row.add(new Tile(numbers.get(rIdx)));
35        // in the code below, changed numbers.get(rIdx) to just rIdx
36        numbers.remove(rIdx);
37
38      }
39      tiles.add(row);
40
41      for (int k = 0; k < 4; k++) {
42        row.remove(0);
43      }
44    }
45
46    return tiles;
47  }
48
49  //implementations of event handlers omitted on purpose. They are not a source of
50  //the bug you need to identify.
51  }
```

At most 4 points, counting down

-3 points Made a change to something else, but did not fix the aliasing problem. The main problem is that the same reference for row was used to make all of the rows. So all of the rows will have the same values. (Leena Razzaq)