How to make the most of a review session

- Not be all end all. You cannot learn a course in two hours!
- Ask as many clarifying questions/general questions as possible.
- Work with partners!
- Maintain a follow-up list of things to do when you are home as well as content you find you are not as solid on
public class Interleaver {
  private String[] myString;

  public String[] interleaver2(String[] other) {

    String[] toRtn = new String[______________];
    // your code here
    for (_________________________)
    {
      // your code here
    }
    return toRtn;
  }
}
Simple Extension: Interleave

```java
public class Interleaver {
    private String[] myString;

    public String[] interleave2(String[] other) {
        String[] toRtn = new String[other.length];
        // your code here
        for (int i = 0; i < other.length; i++) {
            // your code here
        }
        // your code here
        return toRtn;
    }
}
```

*Arrays* do NOT have to be same length
• Takes a 2D array of “m” entries (you DON’T have a variable m to access)
• All arrays have to be of the same length! (whew….)
• Interleaves them all

```java
public static String[][] mInterleave(String[][][] entries) {
    // your code here
}
```
Crazy Extension

- For you to do at home
- Do m interleave with arrays of different length…. (yikes)

```java
public static String[][] mInterleave(String[][] entries)
{
    // your code here
}
```
public class What {
    public long n;

    public void increment() {
        n++;
    }

    public static void reset(What w) {
        w.increment();
        w = new What();
        w.n = 0;
    }

    public static void main(String[] args) {
        What w = new What();
        w.n = 7;
        reset(w);
        System.out.println("The number is " + w.n);
    }
}
The number is 8
public class Soda {
    public String name;

    public Soda() {
        Soda pop = new Soda();
        pop.name = "Dr. Pepper";
    }

    public static void main(String[] args) {
        System.out.println((new Soda()).name);
    }
}
public class Soda {
    public String name;

    public Soda() {
        Soda pop = new Soda();
        pop.name = "Dr. Pepper";
    }

    public static void main(String[] args) {
        System.out.println((new Soda()).name);
    }
}
The following method causes one compiler error. Point out the bug and show how to fix it.
Once the compiler error is fixed, point out which line sometimes throws a run-time exception, and show how to fix it by changing only that line, and without changing the line’s intended effect.
Now that you’ve debugged the program, what does it do?
public static void main(String[] a) {
    for (i = 0; i < a.length; i++) {
        if (a[i].length() > a[i - 1].length() && i != 0) {
            a[i] = a[i] + 9;
        }
    }
    System.out.println(a[i]);
}
public static void main(String[] a) {
    for (int i = 0; i < a.length; i++) {
        if (a[i].length() > a[i - 1].length() i != 0) {
            a[i] = a[i] + 9;
        }
    }
    System.out.println(a[i]);
}

Compile-time error: Must initialize i
public static void main(String[] a) {
    for (int i = 0; i < a.length; i++) {
        if (i != 0 && a[i].length() > a[i - 1].length()) {
            a[i] = a[i] + 9;
        }
        System.out.println(a[i]);
    }
}

Run-time Exception: Possible ArrayIndexOutOfBoundsException in a[-1]
```java
public static void main(String[] a) {
    for (int i = 0; i < a.length; i++) {
        if (i != 0 && a[i].length() > a[i - 1].length()) {
            a[i] = a[i] + 9;
        }
        System.out.println(a[i]);
    }
}
```

What it does: Prints out the command-line parameters, appending a “9” to each parameter that is longer than the previous parameter.
Remember Adding Machine?

- He he he....
Digit Machine

- The next slide has skeleton code for a "DigitMachine class"

- This class, like AddingMachine, keeps a running total
- given an arbitrary integer argument:

for each digit
- 0 ignores rest of input and means print subtotal
- odd numbers implies output squared and then added to the total (executed after other rules done)
- 7–9 requires taking subtotal % digit and adding it to subtotal
- given a 4 you parse the next number in sequence and add both to total (see examples)
- two non–special zeros at any location given a number signal the function is over and total is printed
- special defined as digit after zero which follows four (doesn't follow other rules)
Input: 101459

- $9 \% 0 = 9 \Rightarrow \text{subtotal} = 9$
- $5^2 = 25 \Rightarrow \text{subtotal} = 9 + 25 = 34$
- $4 \Rightarrow \text{read 1} \Rightarrow \text{subtotal} = 34 + 14 = 48$
- $0 \Rightarrow \text{print “subtotal: 48”}$
(a) Come Up with 5 great test cases
Don’t worry about integer overflow
Hints:
◦ Rubric for problems like these look for 10 or so things
◦ A test case may have more than one of the things we look for.
◦ Remember to say what you are testing but don’t get too complicated. Phrases like: ‘consecutive zeros’ and ‘no input’ are just as good as “this case is good because there are many zeros that are together in the same place and I have to blah blah blah”
Things we might like for

- Case with 4 or multiple fours
- Consecutive 4s
- One zero
- Multiple zeros
- Case with 7–9
- Case with odd
- Case with 7–9 and odd
- Case with 1–3 & 5–6
- All zeros/all one number
public class DigitMachine
{
    // feel free to define helper functions as necessary
    // be sure to define such functions correctly!

    public static void main (String[] args)
    {
        Scanner scan = new Scanner(System.in);
        int subtotal = 0;
        while (_______)
        {
            int input = scan.nextInt();
            // your code here

            while (_______)
            {
                // your code here
            }
            // your code here
        }
    }
}
The Old Maid Never Changes
Invariants Definition

DEF: An invariant is a boolean expression that always evaluates to true at given location in code.
The Old Maid Never Changes

- Problem is about invariants regarding popular card game old maid
- Description:
  - Old Maid is a fun card game with the following rules
  - one card is selected to be the old maid
  - players each are given a hand of equal size
  - they find all pairs in their hand and discard them
  - after all pairs are discarded they each take turns pulling one card from the person to their right
  - if the card they pull then forms a pair, they discard it
  - players keep following this procedure until
  - (1) they run out of cards (meaning they win)
  - (2) they only have the old maid left in their hand (they lose)
how we represent cards
2 element strings
first element: value (2, 3, 4, 5, 6, 7, 8, 9, 1, J, Q, K, A)
1 represents 10
second element: suit (1 = Spades, 2 = Diamonds, 3 = Clubs, 4 = Hearts)
• ex) A3 22 64 13 K2
definition of a pair
same face value and suit color
OldMaid NOT part of pair
Part (a): set Old Maid

- we want to write a function that given a card
- makes the card the old maid
- write this function (don't worry about illegal inputs for this part)

```java
30     public class OldMaid
31     {
32         private String[][] hands;
33         private String[] discardPile;
34         private String oldMaid;
35
36     public void setOldMaid(String card) {
37         // your code would go here
38     }
```
now the only thing you can assume is that the argument passed in is a string
add error handling that throws an IllegalArgumentException if the input is not a well-formed card representation
you must have at least 5 descriptive error messages. In order words,
identify 5 good cases of ill-formed inputs
Part (c) List ze invariants

- There are some cool invariants baked into this game
- Name 3 awesome invariants for this.
Part (f): isOk method

- Write an isOk method for this
public class OldMaid {
    private String[][][] hands;
    private String[] discardPile;
    private String oldMaid;

    public void setOldMaid(String card) {
        // your code would go here
    }

    public boolean noDuplicates() {
        // your code here
    }

    public boolean computeComplement() {
        // your code here
    }

    public boolean complementsConsistent() {
        // your code here
    }

    private boolean isOk() {
        // your code here
    }
}