Responsible CS
Facebook Encryption Plans & Law Enforcement

- US, UK, and Australian officials ask Facebook not to use end-to-end encryption.
- Whatsapp already encrypts, extending to Instagram and Messenger.
- Continued effort by Justice Department to bypass encryption.
- Previously, Apple refused to disable password security measures after act of terrorism.

Sources:
https://time.com/5692543/attorney-general-william-barr-facebook-read-encrypted-messaging/

Consequences

- Potential Gov’t Surveillance Abuse, 4th amendment.
- Previous Facebook measures to reduce duration of data storage, prevent snooping.
- 2 billion users: Building backdoor makes messaging less secure/private.
- Facebook working on detecting bad behavior w/o seeing content.
- Security measures limit police abilities to investigate crimes: exploitation, terrorism, extortion.
- “Going Dark” previously easily accessible info.

What should be prioritized, cybersecurity and privacy or lawful access of data? Can a balance be found?

Lecture 12
Arrays

[0] [1] [2] [3] [4] [5] [6] [7]
Outline

• **Purpose**

• **Syntax**
  - Top Hat Questions: Slide 19, Slide 31

• **Multi-Dimensional Arrays**
  - Top Hat Question: Slide 40

• **ArrayLists**
  - Top Hat Questions: Slide 61, Slide 67

• **Generics**

---

**Why Use Arrays? (1/2)**

- So far, we've only studied variables that hold references to single objects
- But what about holding lots of data? Many programs need to keep track of hundreds/thousands of data instances
- Want to hold arbitrary number of objects with single reference – represents a **collection of elements**
  - allows for simple communication to multiple elements
- Arrays are the simplest **data structure** or **collection** - we'll also cover lists, queues, stacks, trees, hash tables

---

**Why Use Arrays? (2/2)**

- Arrays allow instances of particular type to be “packaged” together and accessed as group
- What if there are 10 instances of Dwight:
  - store all Dwight instances in array for easy access (to tell them to annoy Michael, Jim, and others!)
- Arrays are **ordered** - helpful when wanting to store or access instances in particular order, e.g., alphabetically
Your lovely TAs

- We want access to all 50 TAs!
  - Aalia, Andy, ..., Zoe
- Could use instance variables:
  ```java
class CS15TAs {
  private TA _aalia, _andy, ..., _zoe;
}
```
- Can’t access 50 instance variables very easily
  - what if we wanted to access CS15 TAs from 2018, 2017, ...

Fibonacci Sequence (1/2)

- Pervasive in nature, along with golden ratio \( \phi = 1.618 \), logarithmic spiral, etc.
  - [http://jwilson.coe.uga.edu/emat6680/parveen/fib_nature.htm](http://jwilson.coe.uga.edu/emat6680/parveen/fib_nature.htm)
- Tracking first 20 numbers in the Fibonacci Sequence:
  - sequence begins with 0 and 1; successive numbers determined by adding previous two numbers
  - 0, 1, 0+1=1, 1+1=2, 2+1=3, ...

Fibonacci Sequence (2/2)

- Beginning of sequence:
  - 0 1 1 2 3 5 8 13 21 34 55 89
- Could use instance variables for this too:
  ```java
class FibSequence {
  private int _firstNum, _secondNum, ... _twentiethNum;
}
```
- Gets tiresome and isn't flexible
  - try making sequence with forty numbers, one thousand?
  - in algebra, there’s subscript notation: \( F_0, F_1, F_2, ... \)
Arrays (1/4)

- Arrays store specified, constant number of data elements of the same type – our first homogeneous collection
  - each element must be same type or subclass of same type (polymorphism)
- Arrays are special in Java
  - special syntax to access array elements:
    ```
    _studentArray[index]
    ```
  - the index of array is always of type `int`
  - neither base type nor class, but Java `construct`
  - use new to initialize an array (even though it’s not a class!)  
  - special syntax, does not invoke constructor like for a class

Arrays (2/4)

- Arrays only hold elements of specified type
  - when declaring arrays, state `type` of object it stores:
    - base type
    - class
    - sub-arrays (for multi-dimensional arrays – soon)
    - or for max polymorphic flexibility, interface or superclass
  - type can even be `java.lang.Object` to store any instance, but that isn’t useful: wouldn’t take advantage of compiler’s type-checking

Arrays (3/4)

- Every array element is an object reference, subarray, or base type. What real-world objects can be organized by arrays?
  - number of electoral votes by state
  - streets in Providence
  - Strings representing names or Banner ID’s of people in a course
- Elements ordered sequentially by numerical index
  - in Java, use `index` inside brackets, i.e., for an array of students:
    ```
    students[0], students[1], ..., students[n-1]
    ```
Arrays (4/4)

- Arrays store objects in numbered slots
  - for array of size \( n \), first index is always 0, last index is always \( n-1 \)
- Common graphical representations of arrays:

```
0 1 2 3 4 5
[0] [1] [2] [3]
[10] [11] [12] [13] [14] [15] [16] [17] [18]
```

Array Examples

- Houses on a Neighborhood Street
  - array size: 8
  - array index: house number
  - element type: house
  
  **Note:** arrays don’t need to be full (e.g., no house 0, 4, or 7)

- Sunlab Computers
  - array size: 72
  - array index: node number
  - element type: computer
  
  **Note:** Could be modeled as a 2D array (see slide 35)

Java’s Syntax for Arrays (1/5)

Declaration:

```
<visibility> <type>[] <array-name>;
```

- i.e. `private House[] _houses;`
- `<visibility>` denotes the accessibility, i.e. public, private, etc. – we’ve seen this before!
- `<type>` denotes data type array holds: can be class, base type, interface, superclass, or another array (nested arrays)
- Unlike some other programming languages, size of array doesn’t get specified in declaration, but in initialization
  - also no reserved word “array” - [] brackets suffice
Java’s Syntax for Arrays (2/5)

Initialization:

\[
\text{new } \text{type}[\text{size}];
\]

- \text{size} must be integer value greater than 0; indices range from 0 to \text{size}-1
- We use \text{new} here, because arrays are a Java construct
- Only array is initialized, not elements of array; all references are set to a default of null for Objects, 0 for ints, false for boolean, etc.
  - we are still responsible for initializing elements, typically in a constructor

Initializing an Array

- Houses on a Neighborhood Street
  
  \text{House[]} houses = new \text{House}[8];

- Sunlab Computers
  
  \text{Computer[]} sunlab = new \text{Computer}[80];

Java’s Syntax for Arrays (3/5)

- Arrays can be local variables, so they can get declared and initialized in single statement - just like objects and base types:
  
  \text{Colorable[]} other\text{Colorables} = \text{new Colorable}[5];

- Arrays can also be instance variables, which get declared and then initialized separately in constructor:
  
  \text{private Colorable[]} _\text{myColorables};
  ...

  //in constructor of class that contains the array
  _\text{myColorables} = \text{new Colorable}[10];
Java’s Syntax for Arrays (4/5)

- Accessing individual elements:
  - `<array-name>[<index>]`
  - index must be integer between 0 and (array size-1)
  - result is variable stored at that index
  - if `<index>` > size, or < 0, `ArrayIndexOutOfBoundsException` gets thrown
  - also useful to check for uninitialized entries with `<ref> != null` — See slide 50

- Think of student[i] as the “name” of that particular student (like student[i]) – simpler way to refer to each individual element in collection, better than having to use unique names

---

Accessing Array Elements Example

- Houses on a Neighborhood Street
  - House[] houses = new House[8];
  - //code initializing array elements elided
  - House myHouse = houses[6];

- Sunlab Computers
  - CPU[] sunlab = new CPU[72];
  - //code initializing array elements elided
  - CPU myCPU = sunlab[42];

Top Hat Question 1

Which of the following is the correct way to declare and initialize an array of Strings named suspects, of size 5?

A. String suspects = new array(5);
B. String[] suspects = new array(5);
C. String[] suspects = new String[5];
D. String[5] suspects = new String[];
Java's Syntax for Arrays (5/5)

- An array element will work anywhere a variable or constant would. For example, in your PaneOrganizer*:
  
  ```java
  // initialize first element of array Colorables to be Ball
  myColorables[0] = new Ball();
  
  // send a message to 3rd element
  myColorables[2].setColor(javafx.scene.paint.Color.RED);
  
  // assign fourth element to a local variable
  Colorable myColorableVar = myColorables[3];
  
  // pass 5th as a parameter
  _myPaintShop.paintRandomColor(myColorables[4]);
  ```

Arrays as Parameters (1/3)

- Can pass entire array as parameter by adding array brackets to type inside signature
  
  ```java
  public int sum(int[] numbers){
    // code to compute sum of elements in the int array
  }
  ```

- Now we can do the following (somewhere else in the class that contains sum()):
  
  ```java
  int[] myNumbers = new int[5];
  // code elided - initializes myNumbers with values
  System.out.println(sum(myNumbers));
  ```

  Note: There is no way to tell from the use of sum that myNumbers is an array - would need to see how myNumbers was declared to know that

Arrays as Parameters (2/3)

- How do we determine size of array?
  
  - arrays have length as a public property (not a method)
  - use special "dot" syntax to determine length; here we inquire it, then store it for later

  ```java
  int arrayLength = <array-name>.length;
  ```
Arrays as Parameters (3/3)

- How does `.length` work in actual code?

```java
public int sum (int[] numbers){
    //sum all entries in array
    int total = 0;
    for (int i=0; i < numbers.length; i++){
        total += numbers[i];
    }
    return total;
}
```

Note: For loop often used to traverse through all elements of array. Can use loop counter (`i` in this case) inside the body of loop, but should never reset it. Incrementing/decrementing counter is done by for loop itself!

Example: OfficeEmployees

- Design and implement a cartoon with `OfficeEmployees`
- When the “Find Work!” button is pressed, all `OfficeEmployees` should execute `pretendToWork()` method

OfficeEmployees: Quick Look at Design

Things we need:
- App class
- PaneOrganizer class
- OfficeEmployee class
- Private inner ClickHandler class for the button

But once we have all of that...
- How do we make a button do something for all instances of `OfficeEmployee` in sequence?
private class ClickHandler implements EventHandler<ActionEvent> {
    public void handle(ActionEvent event) {
        // loop through array, telling each OfficeEmployee in turn to find an activity
        for (int i = 0; i < employees.length; i++) {
            employees[i].pretendToWork();
        }
    }
}
Adding and Deleting in Arrays (1/2)

- When adding at particular index, all other elements falling in and after that index must get shifted right by programmer (their indices are incremented by 1) otherwise data at index of insertion will be erased and replaced.

Adding and Deleting in Arrays (2/2)

- When deleting from particular index, all other elements falling in and after that index must get shifted left by programmer to fill the newly opened space (index decremented by 1).

Top Hat Question 2

Consider the sum function from slide 23:

```java
public int sum (int[] numbers){
    int total = 0;
    for (int i=0; i < numbers.length; i++){
        total += numbers[i];
    }
    return total;
}
```

What if the code read

```java
i <= numbers.length?
```

A. It would wrap around and add the value at index 0 again
B. It would reach the last element of the array
C. It would raise an ArrayIndexOutOfBoundsException
D. None of the above
Multi-Dimensional Arrays (1/2)

- Modeling chess board:
  - not linear group of squares
  - more like grid of squares
- Can declare array to be 2 (or more) dimensions, by adding more brackets
  - one pair per dimension
  - 2D: `int [[a][b];`
  - 3D: `int [[[a][b][c];`
    // a, b, x, y, z are ints whose values are set elsewhere

Multi-Dimensional Arrays (2/2)

- Multi-dimensional arrays are arrays of arrays of...
- Syntax in previous slide is for rectangular, cuboid, etc. multi-dimensional arrays
  - since multi-dimensional arrays are just arrays of arrays, it is possible (using different syntax) to have jagged arrays, where each sub-array is of different length
  - thus can have "triangle" shaped array
  - don’t use this in CS15; even in CS16 and beyond, it is unlikely you will need this (used predominately for scientific/engineering computation)

2-Dimensional Array Examples (1/2)

- Pixel Array
  - 2D Array size: pixel width by pixel height
  - array indices: x, y
  - element type: RGB color
  - `Pixel[][] microsoft = new Pixel[x][y];`

- Connect Four
  - 2D Array size: 6 by 7
  - array indices: row, column
  - element type: checker
  - `Checker[][] connect4 = new Checker[6][7];`
2-Dimensional Array Examples (2/2)
- The Sunlab
  - 2D Array size: 10 by 8 (approx.)
  - array indices: row, column
  - element type: computer
  - `Computer[][] sunlab = new Computer[10][8];`

Representing Multi-Dimensional arrays (1/2)
- Let's say we want to represent this grid of numbers:

```
1 2 3
4 5 6
7 8 9
```

Representing Multi-Dimensional arrays (2/2)
- How do we want to represent this grid? There are two equally valid options:

```
1 2 3
4 5 6
7 8 9
```

Array of rows

```
1 2 3
4 5 6
7 8 9
```

Array of columns
Ways to Think About Array Storage (1/2)

- Multi-dimensional arrays in Java do **not** make a distinction between rows or columns
  - think about 1D array – it doesn’t really matter if we call it a “row” or a “column”
  - can think of arrays as ordered sequences of data stored in contiguous positions in memory - no intrinsic geometry/layout implied

Ways to Think About Array Storage (2/2)

- Two visualizations of two-dimensional array (called ballArray) are equally valid
  - Column of Rows:
  - Row of Columns:

- Make sure there’s consistency in the way you index into your 2D array throughout your program!
  - Since the elements are not stored in a specific order, the way that we insert elements and initialize and index into our array determines the order.
    - row-major order, i.e., first index is row index (e.g., purple ball is at ballArray[2][0] – column 0, row 2)
    - column-major order, i.e., first index is column index (e.g., purple ball is at ballArray[0][2] – column 2, row 0)

Top Hat Question 3

Here’s a grid of colored golf balls in column major order. What index is the light blue golf ball in?

A. ballArray[2][3]
B. ballArray[2][1]
C. ballArray[3][2]
D. ballArray[1][2]
Example: Size of 2D Arrays

```java
public class ArraySize{

    // deciding which is row and which is column index is arbitrary, but must be consistent!!!
    private static final int NUM_ROWS = 10;
    private static final int NUM_COLS = 5;

    public ArraySize(){
        // String is just an arbitrary choice of type for this array!
        String[][] myArray = new String[NUM_ROWS][NUM_COLS];
        System.out.println("Number of rows = " + NUM_ROWS);
        System.out.println("Number of columns = " + NUM_COLS);
        System.out.println("Size of array = " + find2DArraySize(myArray));
    }

    public int find2DArraySize(String[][] array){
        // row major order, column of rows
        int numRows = array.length;
        // number of entries in column, each a row
        int numCols = array[0].length;
        // element/row 0 is an array
        return (numRows * numCols);
    }
}
```

Common Array Errors - Watch Out! (1/2)

- Cannot assign a scalar to an array

```java
int[] myArray = 5;
```

  - 5 is not an array
  - to initialize array elements, must loop over the array and assign values at each index. Here we assign 5 to each element:

```java
int[] myArray = new int[20]; // initializes array, not elements
for (int i=0; i < myArray.length; i++){
    myArray[i] = 5;
}
```

Common Array Errors - Watch Out! (2/2)

- Cannot assign arrays of different dimensions to each other

```java
int[][] my2DIntArray = new int[2][34];
myIntArray = my2DIntArray;
```

  - Doing so will result in this error:

> "Incompatible types: Can’t convert int[] to int[][]"

- Similar message for assigning arrays of mismatched type
- Take note that Java will automatically resize an array when assigning a smaller array to a larger one
Let’s Make a Board … What Kind?

- Warm-up for Tetris…
- Let’s start with a specification:

Write a Java program that draws sixty-four squares in a grid-like pattern of alternating colors, much like a checker board. The checker board should be eight squares long and eight squares wide. Additionally, the user should be able to click on a button and change the colors of the board from the traditional red and black to the new & bold colors white and blue.

Quick Look at the Design (1/2)

- Some things we’ve seen before:
  - Java program – creates a javafx.stage.Stage
  - buttons – uses javafx.scene.control.Button
  - red, black, white, blue - javafx.scene.paint.Colors
- New things:
  - sixty-four squares - we know about one square, Shape.Rectangle, but 64?
  - checker board - let’s make a 2D 8x8 array of squares
  - row, column – indices into array
- This sample program has crucial design hints for Tetris. Pay close attention!

Quick Look at Design (2/2)

- What classes will we write?
  - PaneOrganizer which creates graphical items and then adds to the Scene Graph
  - CheckerBoard which contains a 2D array of CheckerSquares
  - CheckerSquare which has the ability to toggle its color
- Let’s build them bottom-up
Building CheckerSquares that Change Colors

- Stores 2 colors and toggles between them

```java
public class CheckerSquare {
    // red toggles to white, black toggles to blue
    private Color _currentColor;
    private Color _otherColor;
    private Rectangle _rect;

    public CheckerSquare(Color primaryColor, Color secondaryColor) {
        _rect = new Rectangle();
        _rect.setWidth(Constants.SQR_SIZE);
        _rect.setHeight(Constants.SQR_SIZE);
        _currentColor = primaryColor;
        _otherColor = secondaryColor;
        _rect.setFill(_currentColor);
    }

    public void setLocation(int x, int y) {
        _rect.setX(x);
        _rect.setY(y);
    }

    public void toggleColor() {
        Color temp = _currentColor;
        _currentColor = _otherColor;
        _otherColor = temp;
        _rect.setFill(_currentColor);
    }

    // for board pane to build scene graph
    public Rectangle getRect() {
        return _rect;
    }
}
```

Building Checkerboard (1/2)

- Let's start with standard stuff
  - contains array of CheckerSquares; think of it as a checkerboard with a few methods
  - all CheckerSquares get added to _root in PaneOrganizer – so it needs accessor for the array

Row-major or column-major order? For pixel locations or for square (CheckerSquare) location, natural to think of (x, y) order
  - column-major order corresponds to columns as first index (x-coordinate) and rows (y-coordinate) as second

```java
public class CheckerBoard {
    private CheckerSquare[][] _checkerArray;

    public CheckerBoard() {
        _checkerArray = new CheckerSquare[Constants.NUM_SQRS][Constants.NUM_SQRS];
        for (int col = 0; col < Constants.NUM_SQRS; col++) {
            for (int row = 0; row < Constants.NUM_SQRS; row++) {
                // every other square should be red
                if (((row + col) % 2) == 0) {
                    CheckerSquare rect = new CheckerSquare(Color.RED, Color.WHITE);
                } else {
                    CheckerSquare rect = new CheckerSquare(Color.BLACK, Color.BLUE);
                }
                rect.setLocation(col * Constants.SQR_SIZE, row * Constants.SQR_SIZE);
                _checkerArray[col][row] = rect;
            }
        }

        // for board pane to build scene graph in PaneOrganizer
    }

    public CheckerSquare[][] getRectangles() {
        return _checkerArray;
    }
}
```

Building Checkerboard (2/2)

/* Checkerboard is a thin wrapper around an array of CheckerSquares that in the constructor does a nested for loop to initialize the array, and provides an accessor for it. The event handler in handle() in the private inner class ClickHandler has access to the array so it can toggle all squares */
Updating CheckerSquares

- ClickHandler private inner-class sends message to CheckerBoard to toggle color of all its squares

```java
private inner class ClickHandler implements EventHandler<ActionEvent> {  
    @Override  
    public void handle(ActionEvent event) {  
        for (int col = 0; col < Constants.NUM_SQRS; col++) {  
            for (int row = 0; row < Constants.NUM_SQRS; row++) {  
                // make sure value of array element isn't null (i.e., array initialized correctly)
                if (_checkerArray[col][row] != null) {  
                    _checkerArray[col][row].toggleColor();  
                }  
            }  
        }  
    }  
}
```

What would happen if we didn't check for `null`? We might get a `NullPointerException`! In this code, that couldn't happen, but in general, it's a useful technique to avoid crashing.

PaneOrganizer class

```java
public class PaneOrganizer {  
    private CheckerBoard _board;  
    private BorderPane _root;  
    private CheckerSquare[][] _checkerArray;  
    public PaneOrganizer() {  
        _root = new BorderPane();  
        _board = new CheckerBoard();  
        _checkerArray = _board.getRectangles();  
        this.setUpBoardPane();  
        this.setUpButtonPane();  
    }  
    private void setUpButtonPane() {  
        HBox buttonPane = new HBox();  
        _root.setBottom(buttonPane);  
        Button button = new Button("Change Color!");  
        button.setOnAction(new ClickHandler());  
        buttonPane.getChildren().add(button);  
        buttonPane.setStyle("-fx-background-color: gray");  
        buttonPane.setAlignment(Pos.CENTER);  
    }  
    public Pane getRoot() {  
        return _root;  
    }  
}
```

SciLi Tetris: Loops and Arrays Writ Large

- In 2000, Tech House constructed then the largest Tetris game on the SciLi – the Wiz flew out to play it!
- 5 months of work: 11 custom-built circuit boards, a 12-story data network, a Linux PC, a radio-frequency video game controller, and over 10,000 Christmas lights – see [http://bastilleweb.techhouse.org/](http://bastilleweb.techhouse.org/)
- Video: [https://www.youtube.com/watch?v=tqkiWioo8rU&t=21s](https://www.youtube.com/watch?v=tqkiWioo8rU&t=21s)
java.util.ArrayList (1/2)

- **java.util.ArrayList**, like arrays, hold references to **many** objects of **same** data type
- Another kind of **collection**, also using an index, but much easier management of making changes to array at runtime
- As name implies, it has properties of both arrays and **lists** (covered later)
- Differences with arrays:
  - don’t need to be initialized with size - can hold an arbitrary and mutable number of references
  - are Java classes, not Java constructs, so have methods

java.util.ArrayList (2/2)

- Why use them instead of arrays?
  - when number of elements to be held is unknown
  - storing more data in an array that’s too small leads to errors
  - making array too large is inefficient, takes up more memory than necessary
  - handles update dynamics (shifting elements in memory) for you
- Why use arrays instead of array lists?
  - want something simple
  - want to use less memory (when expect both array and array list to hold same number of elements)
  - want faster operations

**Objects**

- **ArrayLists** can hold **any** **Object**!
- Every class implicitly extends **Object**
  - every object is an **Object**
  - methods of **Object** you can usefully redefine (i.e., override):
    - **boolean equals (Object obj)**: checks for equality
    - **void finalize()**: used in garbage collection
    - **String toString()**: returns object’s “state” as string, could be used to print all instance variables’ values
What can ArrayLists hold?

- **Upside:** ArrayLists store things as Object—maximum polymorphic flexibility
  - since everything is an Object, ArrayLists can hold instances of any and every class: total heterogeneity
  - easy adding/removing anything

- **Downside:** ArrayLists only store Objects:
  - only methods available are trivial ones of Object itself: equals(), toString(), and finalize()
  - typically want homogenous collection to store only objects of particular type (and its subtypes) AND have the compiler do type-checking for that type to enforce homogeneity

Generics! (1/2)

- Generics allow designer to write collection class A to hold instances of another class B, without regard for what class B will be (can be any Object for ArrayLists). User of that class A then decides how to restrict/specialize type for that homogeneous collection
- This is the constructor of the generic ArrayList (a collection class):
  ```java
  public ArrayList<ElementType>();
  ```
  - Think of ElementType as a "type parameter" that is used as a placeholder that the user will substitute for with any non-primitive type (class, interface, array, ...)
  - examples of primitive types: boolean, int, double
  - For example, we saw the use of generics to specialize implementation of EventHandler interface to handle a specific type of Event, e.g., ActionEvent
  - Provides flexibility to have collection store any type while still having compiler help by doing type-checking

Generics! (2/2)

- With generics, ArrayList was implemented by the Java team to hold any Object, but once an instance of an ArrayList is created by a programmer, they must specify the type. Let’s create an ArrayList of Pamphlets for The Office!
  ```java
  ArrayList<Pamphlet> pamphlets = new ArrayList<Pamphlet>();
  ```
- We specify Pamphlets as the type that our ArrayList, pamphlets, can hold.
  - Java will then replace ElementType with Pamphlet in ArrayList method parameters and return types
- Can think of generics as a kind of parameter, just with different syntax (the <>)
  - since only methods have parameters, not classes. In this case, ElementType acts as the formal parameter and Pamphlet is the argument
- Generics, like classes and methods with parameters, provide generality in programming! (as does polymorphism in parameter passing)
java.util.ArrayList Methods (1/6)

// Note: only most important methods shown (All defined for you)
// see Javadoc for full class
// Note: literal use of < and >

public ArrayList<ElementType>()
// one of the many constructors for ArrayList class - specialize
// it by providing ElementType, just as Array has the type it
// stores. Note: < and > are literal - think of them as "of type"
public ElementType get(int index)
// returns the object of type ElementType at that index

public void add(int index, ElementType element)
/* inserts the specified element at the specified position in
this ArrayList; just as with arrays, causes indices of elements
"to the right" to be incremented - but is done automagically */

public boolean add(ElementType element)
// inserts specified element at end of ArrayList

public ElementType remove(int index)
// removes the ElementType at given index

java.util.ArrayList Methods (2/6)

public int size()
// returns number of elements stored in ArrayList

public boolean isEmpty()
// returns true if ArrayList contains zero elements; false otherwise

java.util.ArrayList Methods (3/6)
Top Hat Question 4
Given an array of Cats, we would call .length to get how many elements were in the array. What would we call to get the number of elements in an ArrayList of Dogs?

A. .length
B. .length()
C. .size
D. .size()
java.util.ArrayList Methods (6/6)

- Some other ArrayList notes...
  - can add object in particular slot or append to end
  - can retrieve object stored at particular index and perform operations on it
  - can use for loop to access all objects in ArrayList
  - shifting elements for adding/deleting from ArrayList is done automatically by Java!
    - beware that indices past an insertion/deletion will increment/decrement respectively

Summary of ArrayLists (1/2)

- More flexible than arrays for insertion/deletion
  - dynamically shifting elements and adjusting size in response to insert/delete is all done automatically
- Useful methods and return types:
  - ElementType get(int index)
  - boolean add(ElementType element)
  - void add(int index, ElementType element)
  - int indexOf(ElementType elem) //search
  - ElementType remove (int index)
  - boolean remove (ElementType elem)
  - int size()
  - boolean isEmpty()

Summary of ArrayLists (2/2)

- Can hold heterogeneous collection of any kind of Object; want homogeneous collections...
- Specialize the ArrayList type by adding “generic” specification to a declaration or instantiation - thereby specifying two classes in one statement: the collection and the type of object it will hold and return

```java
ArrayList<Pamphlet> pamphlets = new ArrayList<Pamphlet>();
```

- Remember to use literal <> for specialized type!
Top Hat Question 5
Which of the following uses an ArrayList correctly?

A. ArrayList<Pamphlet> pamphlets = new ArrayList<Pamphlet>();
   Pamphlet dramaticPamphlet = new Pamphlet();
   pamphlets.add(dramaticPamphlet);
B. ArrayList<ElementType> pamphlets = new ArrayList;
   Pamphlet shockingPamphlet = pamphlets[0];
C. ArrayList<Pamphlet> pamphlets = new ArrayList<ElementType>();
   Pamphlet sadPamphlet = pamphlets.first();
D. ArrayList<String> pamphlets = new ArrayList<Pamphlet>;
   Pamphlet crazyPamphlet = new Pamphlet();
   pamphlets.add(crazyPamphlet);

Example: Pamphlets (1/5)
public class OfficePamphletDivision{  
   /*OfficePamphletDivision is a "wrapper" for an ArrayList that augments the 
    * functionality of an ArrayList with, for example, the code that adds and 
    * removes items from the scenegraph. Thus, it provides controlled access to the 
    * ArrayList's items, an important part of the scenegraph. This is another 
    * commonly repeated pattern. We'll also restrict OfficePamphletDivision to have no more 
    * than twenty pamphlets. Replace all occurrences of ElementType with Pamphlet, 
    * a class modeling pamphlets, including where ElementType occurs in literal <> brackets. */ 
   private ArrayList<Pamphlet> _stackOfPamphlets;
   public OfficePamphletDivision(){
      //ArrayList initialization - note literal <>
      _stackOfPamphlets = new ArrayList<Pamphlet>();
      for (int i=0; i<20; i++){
         //Add a Pamphlet at end in each pass
         _stackOfPamphlets.add(new Pamphlet());
         //scenegraph code elided
      }
   } //class definition continued on next slide
   Note: Pamphlet is a class that models a pamphlet

Example: Pamphlets (2/5)
public void addPamphlet(int numPamphlets){
   for (int i = 0; i < numPamphlets; i++) {
      _stackOfPamphlets.add(new Pamphlet());
   }
   System.out.println("Pamphlet stack is full!");
}
*/
/*If _stackOfPamphlets still has pamphlets, remove and return one, thereby removing it from the 
ArrayList, else return null*/
public Pamphlet removePamphlet(){
   if (_stackOfPamphlets.size()) {
      return _stackOfPamphlets.remove(0);
   } else {
      System.out.println("Pamphlet stack is empty!");
      return null;
   }
} //End of Class
Example: Pamphlets (3/5)

- `<Pamphlet>` indicates use of Java generics
  - Ensures only `Pamphlet` instances can be stored and retrieved from this `ArrayList`
- In `OfficePamphletDivision`'s constructor, adding a new `Pamphlet` works:
  ```java
  _stackOfPamphlets.add(new Pamphlet());
  ```
- However, adding another type to `ArrayList` of `Pamphlet` will fail:
  ```java
  _stackOfPamphlets.add(5)
  ```
  - "The method `add(Pamphlet)` in the type `ArrayList<Pamphlet>` is not applicable for the arguments (int)"

Example: Pamphlets (4/5)

```java
public class Michael
private OfficePamphletDivision _division;
public Michael(OfficePampletDivision division){
    // Michael is associated with _division
    _division = division;
}
/*Method allows Michael to approve Pamphlets if the arraylist isn't empty*/
public void approvePamphlets(int n){
    for(int i = 0; i < n; i++) {
        if(_division.removePamphlet() == null){
            this.orderMore(10);
            break;
        } else {
            this.approve(_division.removePamphlet());
        }
    }
}
```

Example: Pamphlets (5/5)

```java
public class DunderMifflinCo
private OfficePamphletDivision _division;
private Michael _michael;
public DunderMifflinCo(){
    _division = new OfficePamphletDivision();
    _michael = new Michael(_division);
}
public void startWork(){
    // Can remove from arraylist because it contains 20 pamphlets initially
    _michael.approvePamphlets(15);
}
```
**for vs. for-each loop (1/4)**

- Intended to simplify most common form of iteration, when loop body gets applied to each member of collection

- How do `for-each` loop and `for` loops differ?
  - `for` loop gives access to index where item is stored
  - `for-each` loops don’t have direct access to index, but can easily access item (see next example)

**for vs. for-each loop (2/4)**

- `for` loops were extended to `for-each` (or `for-in`) loops, which iterate over the contents of a data structure rather than indices
  - `<var>` here NOT literal, i.e., not for generics
    ```java
    for (<type> <var>: <structure>){
        <loop body>
    }
    ```
  - `<type>`: class of objects stored in the `<structure>`
  - `<var>`: name of current element—holds each successive element in turn
  - `<structure>`: data structure (array or other collection) to iterate through

**for vs. for-each loop (3/4)**

- If every element needs to be iterated and loop body doesn’t need element index, `for-each` loops suffice:
  ```java
  // Instead of only even-numbered members of _employees, now everyone must find work!
  for (OfficeEmployee employee: _employees){
      // notice how don’t need to use index to get members from ArrayList
      employee.pretendToWork();
  }
  ```

- Great advantage of `for-each` loops is that they don’t raise `ArrayIndexOutOfBoundsException`! Why?
  - Java does the indexing for you!
for vs. for-each loop (4/4)

- Consider this for loop:
  ```java
  //Somewhere in TheOffice class
  //note: _employees is an ArrayList<OfficeEmployee>
  for (int i=0; i < _employees.size(); i++)
      if (i % 2 == 0){ //if index 'i' is even
          _employees.get(i).pretendToWork();
      }
  ```
- Only want to call pretendToWork() on elements at even indices, but for-each loop wouldn't work:
  - we don't execute pretendToWork() on every element in the ArrayList; we only care about elements at specific indices

ConcurrentModificationExceptions

- When trying to modify an ArrayList while iterating through it with a for-each loop, you will get a ConcurrentModificationException:
  - Adding and removing cannot be done within a for-each loop because of the shifting of the elements in the list that Java does in response to an add or remove
  - Note: this is important for DoodleJump! We'll go over this issue in detail during the project help slides.

*Understanding Mainline (and optional params)*

- You've seen the mainline before, but let's talk about its parameters:
  - If we type this in a terminal:
    ```java
demos/Mainline/App hello CS15
    ```
  - Output says:
    ```
    hello CS15
    ```
  - Why? argv's size is exactly equal to the number of parameters passed to mainline. We are accessing the first and second elements, and if those are out of the size bound this would raise an error
  - You won't need to use mainline parameters in CS15, but it's a good thing to know!
Announcements

• Lab section in the Sunlab this week!

• Early handin for Cartoon is tonight
  o On-time handin on Thursday at 11:59pm
  o Late handin on Saturday at 11:59pm