Review for the Third Exam

COMP163

“On the plains of hesitation lie the blackened bones of countless millions who at the dawn of victory lay down to rest, and in resting died.”

Adlai E. Stevenson

"... to the United States and the world he was the voice of a reasonable, civilized, and elevated America."

biographer writing about Adlai Stevenson

COMP163 Schedule

		Fri, Nov 16 review
Mon, Nov 19 Exam 3	Wed, Nov 21 Thanksgiving Holiday (no classes)	Fri, Nov 23 Thanksgiving Holiday (no classes)
Mon, Nov 26 Software engineering	Wed, Nov 28 final review	Thursday, Nov 29 Lab Final
Tuesday, Dec 4 10:30 – 12:30 Final Exam		

Teaching Evaluation

- The official University teaching evaluation survey is on Blackboard
- Complete the survey for ALL classes

- 26% of the students in COMP163 have completed the evaluation

Exam 3 Topics

	ZyBook section
• classes	8
– constructors	
– class variables	
• static and non-static class variables	
– creating objects	
• methods	9
– parameter passing	
• scoping	8
• arrays	10
• Secure programming	

Anything is fair game

- The exam may contain questions from any of the material covered in class since the last exam

- Knowledge is cumulative, so using the newer topics may require you to know previous topics

One Page of Notes

- You are allowed one and only one 8½ by 11 inch page of notes during this exam
- You are not allowed to use more than 187 square inches of paper surface
- You will do better if you make your own page of notes and not copy your friend's notes

Practice Exam

- A practice exam from a previous semester is on Blackboard under course materials

- The exam will ask questions on the same topics as the practice exam, but will not be the same as the practice exam

Exam Format

- The exam will be similar to previous exams, labs, quizzes and homework
- Exams tend to be programming oriented
- Most question will be *“write a method to”*, *“declare an array of”* or *“what does this program display”*

Secure Programming

- A good program checks all input values to make sure they are reasonable
- Numbers should be within range
- Avoid characters that may signal code injection
- This not only makes the program more secure, but it improves the user interface
- Be aware of the possibility of integer overflow

What input could cause the cow array to generate an error?

```
int dog = keyboard.nextInt();
if ( dog > 0 && dog + 3 < 100 ) {
  double[] cow = new double[ dog + 3 ];
}
```

A. 0
B. 3
C. 2147483647
D. It will always work
E. It will never work

Integer Overflow

- Integers can only hold numbers up to about 2.1 billion
- If you do arithmetic that results in an answer greater than 2.1 billion, the integer will overflow and become negative
- Doubles become `INF` (infinity) if the value gets too big, approximately 1.8×10^{308}
- Java does not get a run time error (with error message and stack trace) if an int or double overflows

Methods of Objects

- Imagine you have this class

```
public class Turkey {
        public int eagle = 5;
        public Turkey( int condor ) {
                eagle = condor;
        }
        public void setEagle( int vulture ) {
                eagle = vulture;
        }
}
```

Accessing an Object

- You can make a Turkey object by

```
Turkey pelican = new Turkey(3);
```

- The main program can access the eagle value of an object directly

```
int albatross = pelican.eagle;
```

- You can call the method by

```
pelican.setEagle( 7 );
```

Empty Array

- When you create an array of double or int, the array initially contains many doubles or ints
- When you create an array of objects, the array is initially empty

Creating an Array of Objects

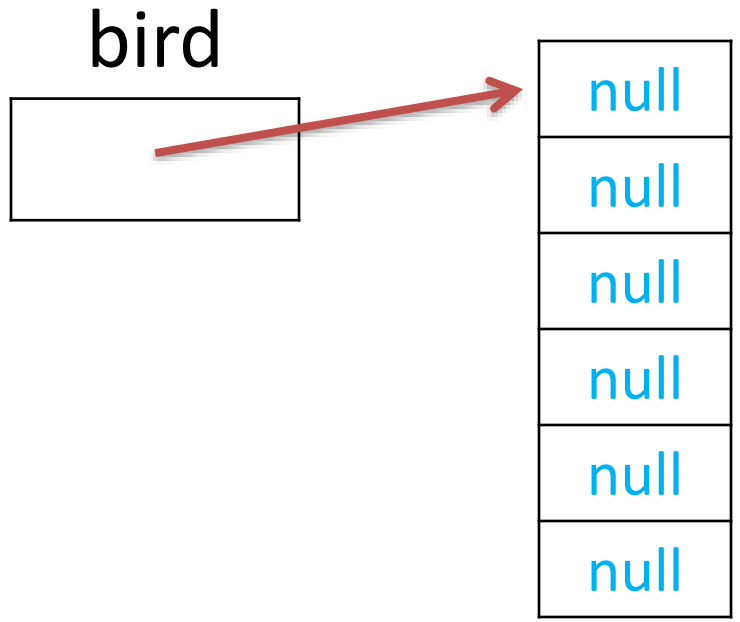
- When you declare an array of objects. The variable does not yet hold the array

Turkey[] bird;

Creating an Array of Objects

- An array is created the same way you create other objects. The **new** create the array

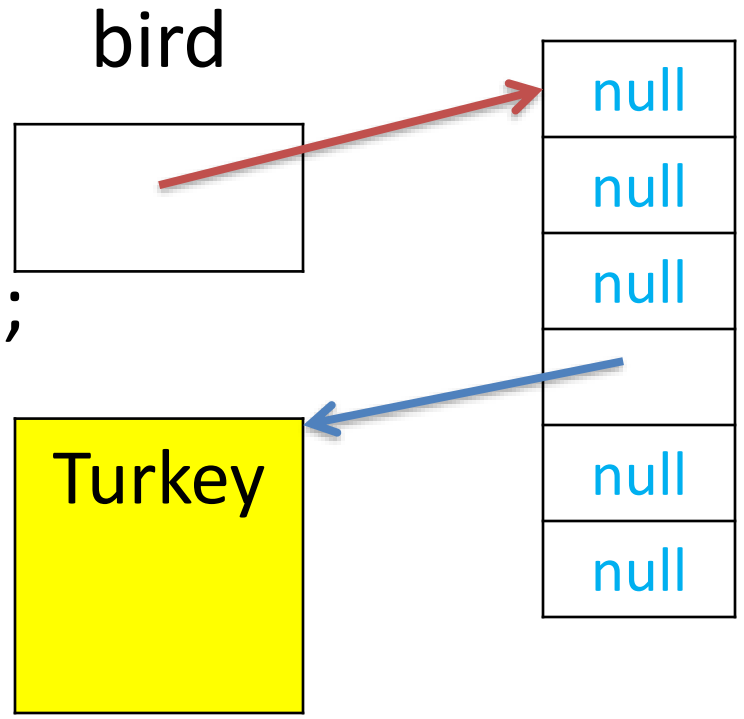
```
Turkey[] bird;
bird = new Turkey[6];
```



Creating an Array of Objects

- You have to create each of the element objects in the array The **new** creates each element

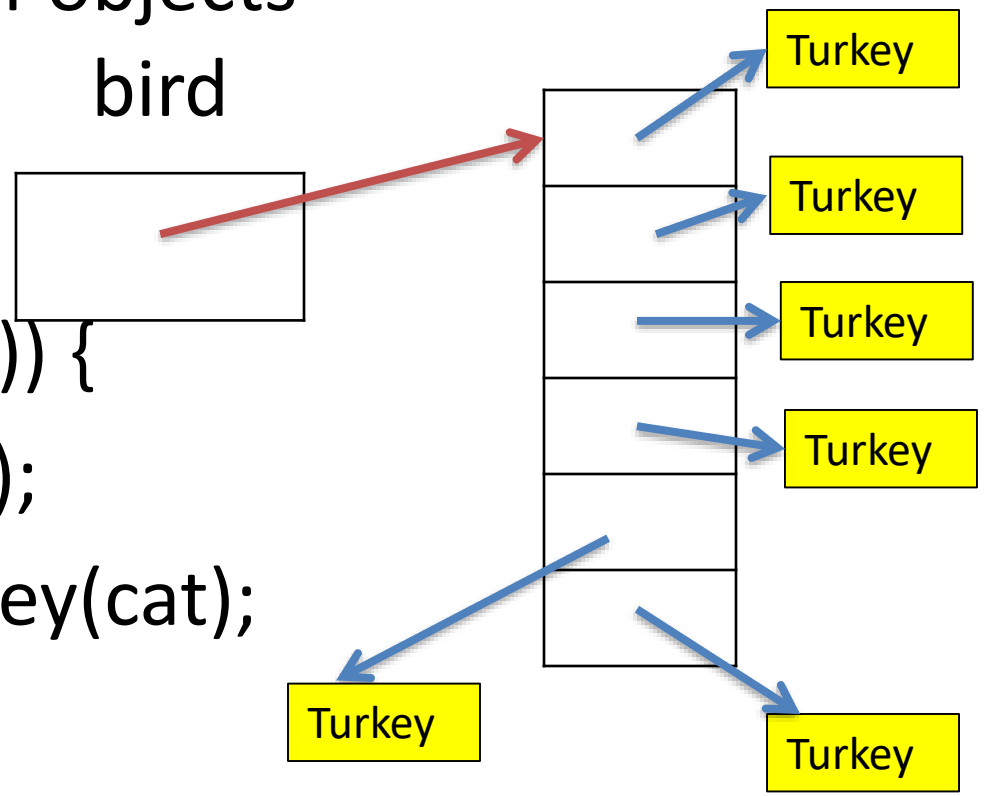
```
Turkey[] bird;
bird = new Turkey[6];
bird[3] = new Turkey(47);
```



Creating an Array of Objects

- You should create a new object for each element of the array of objects

```
Turkey[] bird;
bird = new Turkey[6];
while (inFile.hasNextInt()) {
   int cat = inFile.nextInt();
   bird[count] = new Turkey(cat);
   count++;
}
```



Creating Objects

- You create an object in Java with new

```
Turkey hummingbird = new Turkey( 5 );
```

class

constructor method

- After new is a call to the constructor method

Local Variables

- Variables declared inside a method are known as local variables
- Variables declared inside the main method are also local variables
- Local variables can only be used inside the method where they are declared

Parameter Variables

- In addition to local variables, a method can use its parameter variables

```
double avg( int dog, int cat ) {
    double sum = dog + cat;
    return sum / 2.0;
}
```

- **sum** is a local variable while **dog** and **cat** are parameter variables

Class Data

- An object can have data variables called **Fields, Properties** or **Instance variables**
- Data can be primitive data types, such as double or int, or other objects
- A class encapsulates data values that form a logical entity, such as
  - Information about a student in class
  - phone book
  - picture

Write with your team

- Create a class called Phone that has two data values, owner's name and phone number
- Write a constructor to initialize both

Possible Solution

```
public class Phone {
    String owner;
    String phoneNumber;
    public Phone(String owner, String dial) {
        this.owner = owner;
        phoneNumber = dial;
    }
}
```

Class Variables

```
public class Reptile {
    int skink = 11;
    int doit( int toad ){
        int lizard = skink + toad;
        return lizard;
    }
}
```

- **skink** is an instance variable
- **toad** is a parameter variable
- **lizard** is a local variable

Class Instance & Local Variables

- Local variables in methods are reset every time you call the method
- Data in object instance variables last for the life of the object
- Data in a static instance variables last for the life of the program

What is displayed?

```
public class Click {
    int  frog= 5, toad= 7;
 void myfunc(int pollywog) {
        int frog;
        frog = pollywog + 1;
        System.out.print( frog );
 }
 public static void main(...) {
    Click newt = new Click();
    newt.myfunc( 3 );
    System.out.print( newt.frog );
  }
}
```

A. 4 4

B. 4 5

C. 5 5

D. none of above

Nowhat is displayed?

```
public class Click {
    int  frog= 5, toad= 7;
 void myfunc(int pollywog) {

       frog = pollywog + 1;
       System.out.print( frog );
 }
 public static void main(...) {
    Click newt = new Click();
    newt.myfunc( 3 );
    System.out.print( newt.frog );
  }
}
```

A. 4 4

B. 4 5

C. 5 5

D. 6 6

E. none of above

Variable Range

- Variables defined in a block can only be used in that block following the variable declaration

```
{
        // you cannot use the variable moth here
        double moth = 72.5;
        // you may use the variable moth here
        // this is the scope of moth
}
// The variable moth is not allowed here
```

Local Variables

```
static void main( ) {
    double    cat = 5, bird = 47;
    cat = myfunc( bird );
    System.out.print(cat);
}
```

Scope of cat and bird

```
int myfunc(double cow) {
    int bull;
    bull = (int)cow * 2;
    return bull;
}
```

Scope of cow and bull

Local Variables *(Tricky)*

```
static void main( ) {
    double   cat = 5, bird = 47;
    cat = myfunc( bird );
    System.out.print(cat);
}
```

Scope of cat and bird

```
int myfunc(double cow) {
    int bird;
    bird = (int)cow * 2;
    return bird;
}
```

Scope of cow and bird (different bird)

Overlapping Names

- Java allows you to declare a variable with the same name in an inner block
- The declaration closest to the variables use is the one that applies

- This can be ***very*** confusing and should be avoided

More Name Collisions

```
public class Collide {
  int rat = 3;
  public int square( int mouse) {
     int rat = mouse * mouse;
     return rat;
  }
}
```

- The class instance variable rat is a different memory location from the local variable rat

Name Collision

```
public class Collide {
  int rat = 3;
  public int square( int rat ) {
     return this.rat * rat;
  }
}
Collide thing = new Collide();
int turtle = thing.square( 2 );
```

- turtle gets the value 6

Documenting Classes

- Javadoc comments start with /** and end with */
- A class must start with a Javadoc comment

```
/**
   Description of the program. More details.
   @author your name
*/
```

javadoc for Methods

- You should put a javadoc comment before every method defining what the method does
- The first sentence of your description will be used as the short description of the method
- You should also include
  - @param for each parameter
  - @return unless it is a void method

javadoc Example

```
/**
 * This is an example.
 * @author Ken Williams
 */
public class DocPage {
 /**
  * Compute the area of a rectangle.
  * @param high The height of the rectangle. The
  * height is the vertical dimension.
  * @param wide The width of the rectangle.
  * @return The area of the rectangle.
  */
 public double area(double high, double wide) {
```

What Button?

- Many GUIs have more than one active object
- You can determine where the action was taken by using the **getSource()** method of the event
- The result of **getSource()** should be compared to each GUI object

```
public void actionPerformed(
                    java.awt.event.ActionEvent   frog ) {
    if ( frog.getSource() == GUIobject ) {
```

Example Use of getSource()

```
JButton delete = new JButton("kill");
JButton save = new JButton("allow");
          - - - - - - - - - - - - - - - - - - - - - - - - -
public void actionPerformed(
                    java.awt.event.ActionEvent   toad) {
    if (toad.getSource() == delete ) {
        // delete something
    } else {  // must have been the save button
       // save something
    }
```

What will select the pressed button?

```
JButton bass = new JButton();   // class instance variable
public void actionPerformed(
                java.awt.event.ActionEvent  trout)
```

A. if(bass == JButton.getSource())
B. if(this == trout.getSource())
C. if(trout == bass)
D. if(trout.getSource() == JButton)
E. if(trout.getSource() == bass)

Exempt from the Final

- The final exam will be optional for a student when it has been determined by the instructor that it is statistically unlikely that the final exam will change the student's grade
- A student always has the option to take the final exam if they wish to do so
- When a student is permitted to not take the final exam, their course grade will be determined by the weighted average of all other graded work

Statistically Unlikely

- If it is improbable that the final exam will change your grade, there is no reason to take the final
- A student with a 98 average will have to earn a 26 on the final for their grade to drop to a A-
- Students with score over 90 or under 39 are exempt
- Other students in the middle of their grade range may also be exempt

Likely Numbers

Based on the grades of 174 COMP163 students at this time, about:

- 12% students will be exempt with an “A”
- 7% students will be exempt with a “C”
- 7% students will have a hopeless “F”

- 74% of the students will have to take the finals

Exempt from both GEEN163 Finals

- A statement at the end of your grade summary will tell you if you are exempt from the GEEN163 final exam

- If you are exempt from the final, you do not have to take the lab final
- If you must take the final, you must also take the lab final

Initializing Arrays

- Like simple variables, arrays can be initialized to a value when they are declared
- When you initialize an array, you do not need to specify the size. The size is determined by how many values you use to initialize

```
double[] deer = { 3.2, 1.4, 26.1, 8.6};
```

- The initialization values must be the correct type
- Note the semicolon after the curly bracket

Loops

- When you are dealing with the elements of an array, you almost always use a loop

```
//  Add together all values of the array bear
double[] bear = new double[5000];
        //  Something is put in all elements of bear
double sum= 0.0
for (int deer = 0; deer < bear.length; deer++) {
        sum += bear[ deer ];
}
```

Arrays as Objects

- An array is an object in Java
- You can call a method on the array
- Arrays have an integer class variable length that contains the length of the array

```
double[] cobra = new double[10000];
int boa = cobra.length;                    // boa = 10000
```

Half Empty or Half Full

- An array might not contain as many useful values as its full capacity

```
Widget[] eagle = new Widget[100];
for (int egg = 0; egg < 47; egg++) {
        eagle[egg] = new Widget();
}
```

- The array eagle has 100 elements, but only 47 values have been used
- eagle[60].getCount() will cause a nullPointerException

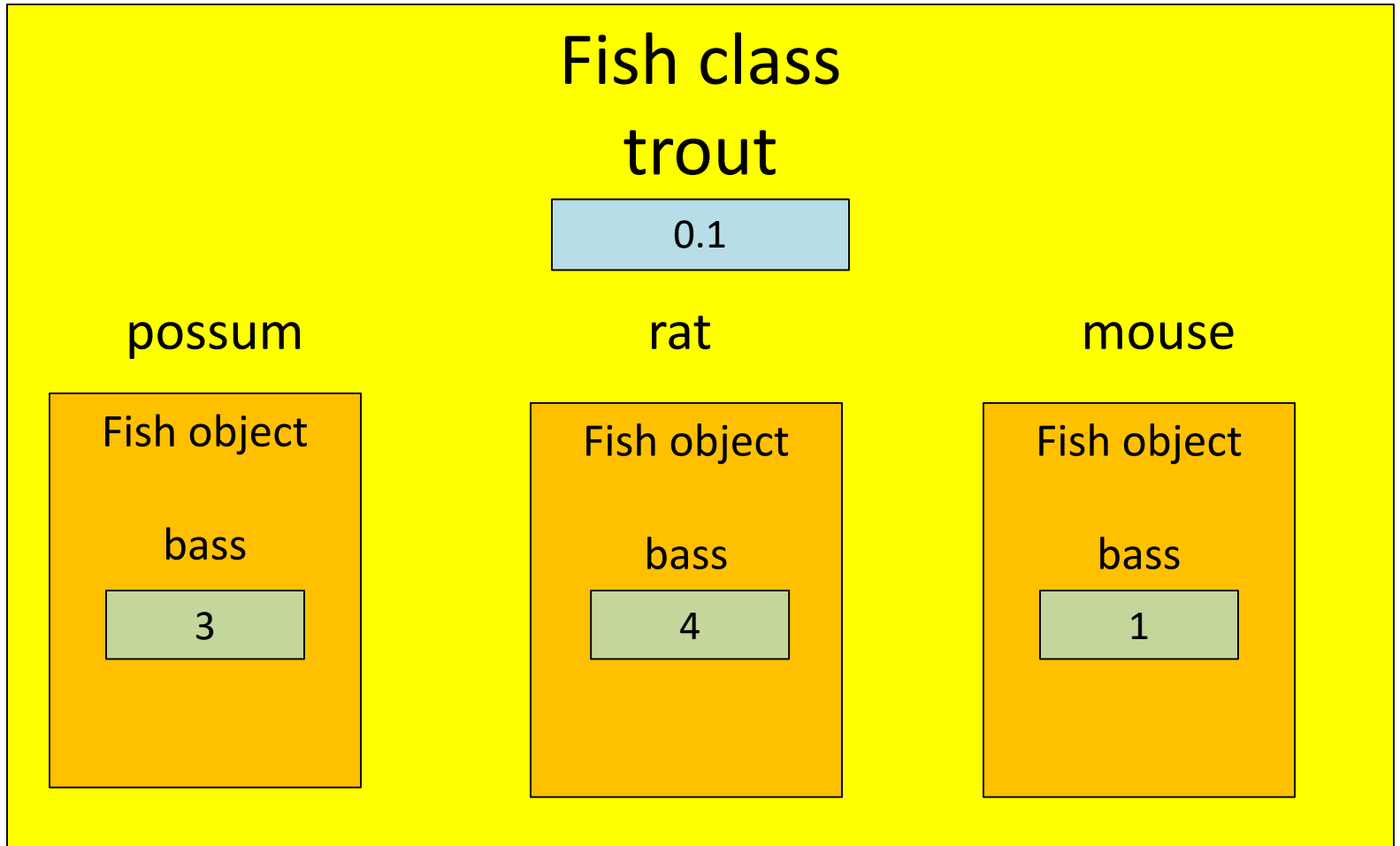
Multiple Objects

```
public class Fish {
    private int bass;
    private static double trout;
}
Fish possum = new Fish(3, 5.5 );
Fish rat    = new Fish( 4, 8.8 );
Fish mouse  = new Fish( 1, 0.1);
```

- There are three Fish objects
- Each object has its own copy of bas
- There is only one trout variable

Static and Instance Variables

- There is only one trout variable, but there is a dog for every object



What is displayed?

```
public class Stest {
   static int svar = 1;
   int dvar = 1;
   void incBoth() {
      svar++;
      dvar++;
      System.out.println(svar+" "+dvar);
   }
   static public void main(String[] x) {
      Stest cat = new Stest();
      Stest dog = new Stest();
      cat.incBoth();
      dog.incBoth();
   }
}
```

A. 1 1
   1 1

B. 2 2
   2 2

C. 2 2
   3 3

D. 2 2
   3 2

Write with your Neighbors

- Write a method to count the number of negative values in an array of doubles

```
int  countNegative( double[] dog )
```

Possible Solution

- Write a method to count the number of negative values in an array of doubles

```
int  countNegative( double[] dog ){
    int count = 0;
    for (double cat : dog) {
        if (cat < 0) {
            count++;
        }
    }
    return count;
}
```

Another Possible Solution

- Write a method to count the number of negative values in an array of doubles

```
int  countNegative( double[] dog ){
    int count = 0;
    for (int cat=0; cat <dog; cat++) {
        if (dog[cat] < 0) {
            count++;
        }
    }
    return count;
}
```

Putting a Positive Spin on It

- Consider a modification to our method that makes all of the negative number a positive number of the same magnitude

Will this work?

- Make all the numbers positive

```
int  makePositive( double[] dog ){
    int count = 0;
    for (int cat=0; cat <dog; cat++) {
        if (dog[cat] < 0) {
            dog[cat] = -dog[cat];
        }
    }
    return count;
}
```

A. No – you cannot change the array
B. No – you can't put a dash before dog
C. All the above
D. Yes

Likely Questions

- Create and initialize an array
- Write a class with constructor and other methods
- Write Javadoc for a method
- Write a method
- Complete a GUI

COMP163 Schedule

		Fri, Nov 16 review
Mon, Nov 19 Exam 3	Wed, Nov 21 Thanksgiving Holiday (no classes)	Fri, Nov 23 Thanksgiving Holiday (no classes)
Mon, Nov 26 Software engineering	Wed, Nov 28 final review	Thursday, Nov 29 Lab Final
Tuesday, Dec 4 10:30 – 12:30 Final Exam		

Teaching Evaluation

- The official University teaching evaluation survey is on Blackboard
- Complete the survey for ALL classes

- 26% of the students in COMP163 have completed the evaluation