Review for the Third Exam

GEEN163

“I was thinking about how people seem to read the Bible a whole lot more as they get older; then it dawned on me - they're cramming for their final exam.”

George Carlin

Exam 3 Topics

	book section
• classes	7
– constructors	
– class variables	
• static and non-static class variables	
– creating objects	
• methods	7
– parameter passing	
• scoping	7
• arrays	8

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

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

Demanding Good Input

```
String[] arrayA = { “good”, “bad”, “ugly” };
int answer;
boolean good = false;
do {
        System.out.println(“Enter a number from 1 to 3”);
        answer = keyboard.nextInt();
        if (answer >= 1 && answer <= 3)  {
                good = true;
        } else {
                System.out.println(“Hey fool! Only 1 to 3”);
        }
while ( !good );
```

Write with your team

- Write a method that returns **`true`** if the parameter beetle is between low and high and **`false`** if it is not

```
boolean isOK( int beetle, int low, int high)
```

Possible Solution

```
boolean isOK( int beetle, int low, int high) {
    if ( beetle > low && beetle < high)
        return true;
    else
        return false;
}
```

Another Possible Solution

```
boolean isOK( int beetle, int low, int high) {
    return beetle > low && beetle < high;
}
```

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}

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

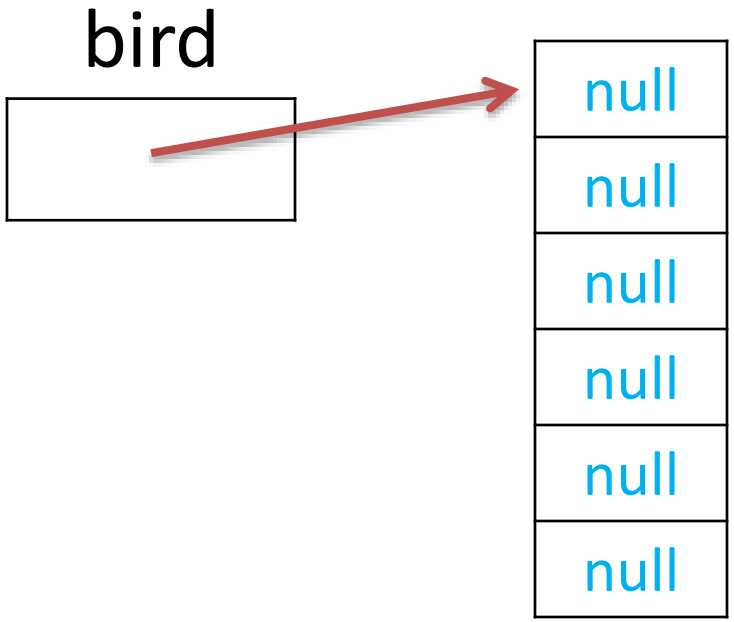
- When you declare an array of objects. The variable does not yet hold the array

Widget[] bird;

Creating an Array of Objects

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

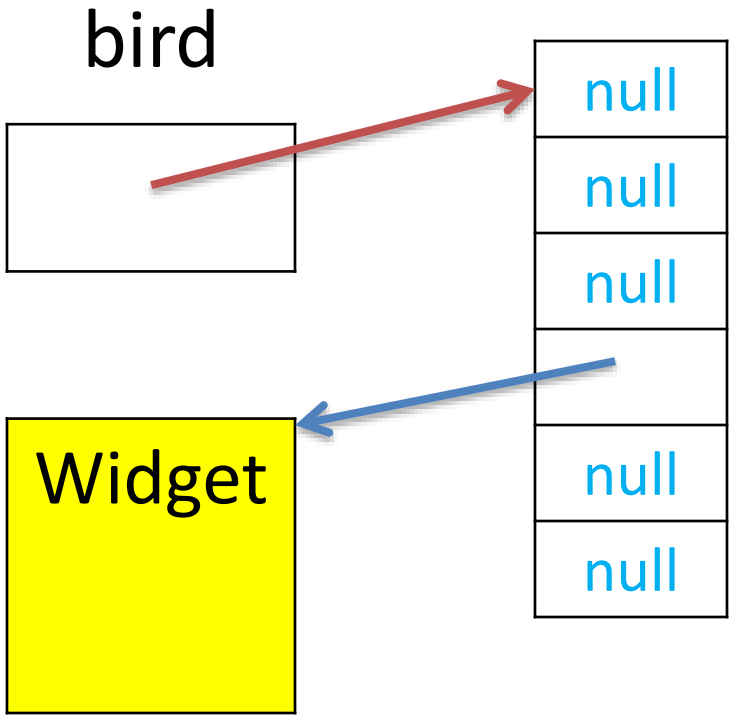
```
Widget[] bird;
bird = new Widget[6];
```



Creating an Array of Objects

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

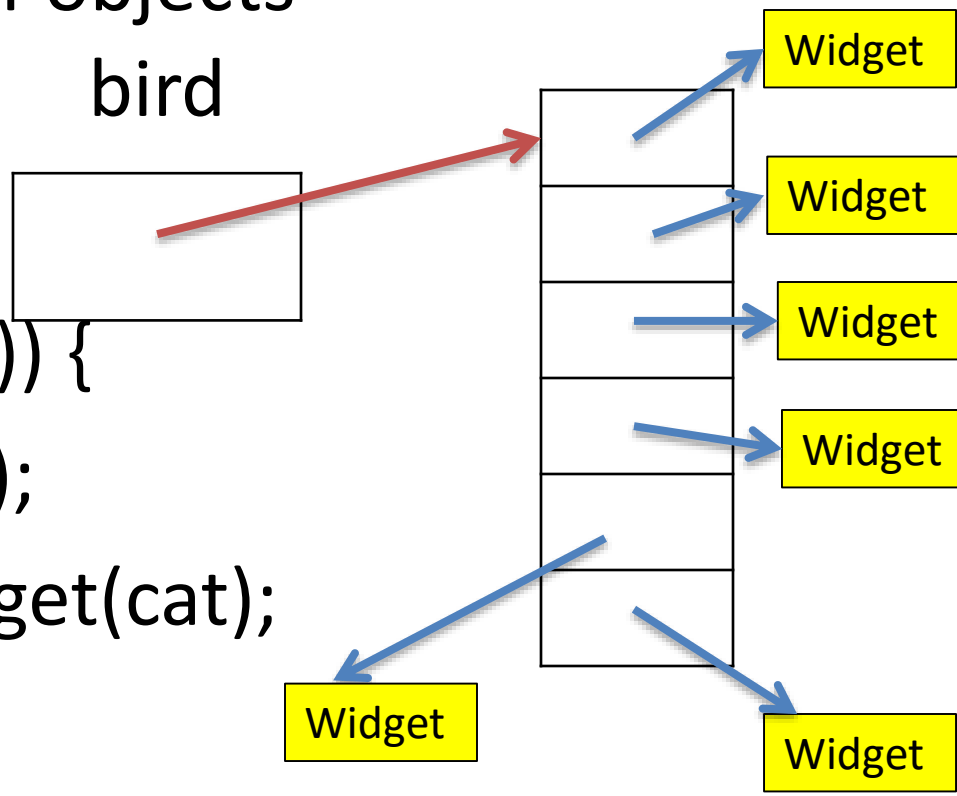
```
Widget[] bird;
bird = new Widget[6];
bird[3] = new Widget();
```



Creating an Array of Objects

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

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



Creating Objects

- You create an object in Java with new

```
Widget goat = new Widget( 5 );
```

- After new is a call to the constructor method

```
public class Widget {
    private int bull;
    public Widget(int cow) {
        bull = cow;
    }
}
```

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

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

```
double myfunc(double cow) {
    double bull;
    bull = cow * 2.0;
    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

```
double myfunc(double cow) {
    double bird;
    bird = cow * 2.0;
    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

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( frog );
  }
}
```

A. 4 4

B. 4 5

C. 5 5

D. none of above

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
    }
```

Looping in GUI Programs

- When you want a user to enter multiple values in a GUI program, you probably do **not** need a for loop or a while loop
- Each time the user enters a name, it will call actionPerformed

Making a GUI Application

1. Create a class that extends JFrame and implements java.awt.event.ActionListener
2. Create component objects
3. Specify the layout manager in the constructor
4. Add the component objects to the container
5. Add an action listener to the components that respond to user input
6. Create an actionPerformed method

What will select the pressed button?

```
JButton bass = new JButton();
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 )`

Conversion from Applet to Frame

- Change JApplet to JFrame
- Change void init() to constructor method
- Add to the end of the constructor method:

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
pack();
setVisible(true);
```

- create a main method

```
public static void main(String[] unused) {
    MyProg aardvark = new MyProg();
}
```

Alternate Conversion

- Change JApplet to JFrame
- Add to the end of the init() method:

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
pack();
setVisible(true);
```

- create a main method

```
public static void main(String[] unused) {
    MyProg aardvark = new MyProg();
    aardvark.init();
}
```

Likely Questions

- Create and initialize an array
- Write a class with constructor and other methods
- Evil questions with class and local variables that have the same name