Java Classes

GEEN163 Introduction to Computer Programming

“Never interrupt someone doing what you said couldn't be done.”

Amelia Earhart

MyCodeLab

- Read chapter 7 of the Java Illuminated textbook and answer the TuringsCraft questions for chapter 7
- You will earn 4 points for each correct answer up to a maximum of 100 points
- Complete this assignment by midnight on Thursday, March 17, 2016

Lab Quiz

- There will be a quiz in lab this Thursday
- A lab quiz is similar to a regular lab, but you must do it by yourself
- You may use your notes, book and the web
- You are not allowed to communicate with other people
- You must be in the Graham 203 lab

Exam

- The second GEEN163 exam will be in lecture on Wednesday, March 23, 2016
- It will cover everything since the first exam

Classes, Objects, & Methods

- Object-oriented programming uses classes, objects, and methods as basic programming components
- These components help to
  - organize a large program into small modules
  - design and think about an intricate program
  - abstract and isolate concepts

Nomenclature

- **Class** – defines a kind of object
- **Object** – an instance of a class
- **Instantiate** – creating an object of a class
- **Instance** – An object is an instance of a class
- **Method** – coordinated sequence of instructions or function that an object can perform

Classes and Objects

- Programmers can define new classes in a Java program
- You can create objects from existing classes in your programs

Defining a Java Class

- An object should contain the information about something
- An object should encapsulate a concept
- A class is a new data type, like double or String. You can create new data types as you need in Java.

Syntax of a Java Class Definition

```
public class ClassName {
    // data declarations
    // methods
}
```

Example Class

```
public class Widget {
  private int count = 0;                    // class data value

  public void inc() {                       // method to increment
    count++;
  }
  public int getCount() {                   // accessor method
    return count;
  }
 public void setCount( int value ) {        // modifier method
    count = value;
  }
}
```

Object Data

- Each object has its own copy of its data
- If you have two objects of the same class, their data values are independent

```
Widget  dog = new Widget();
Widget  cat = new Widget();
dog.setCount( 47 );
```

The Widget class does not have a main method because

A. it is a JFrame
B. it is to be used in another class
C. it has private data
D. the inc method serves as the main

Declaring an Object

- Imagine we have a class named Widget
- We can declare an object of type Widget just like we declare a variable to be an int or a double

```
Widget  thing;
Widget  dodad, whatchamacallit;
```

Instantiating Objects

```
Widget gazelle = new Widget();
Widget zebra;
zebra = new Widget();
```

- After the word "new" is a call to a constructor method that helps create the object. The constructor method may or may not have parameters.

What is displayed?

```
Widget cat = new Widget();
Widget dog = new Widget();
cat.setCount(5);
dog.setCount(7);
int goat = cat.getCount();
System.out.println( goat );
```

A. 3

B. 5

C. 7

D. none of the above

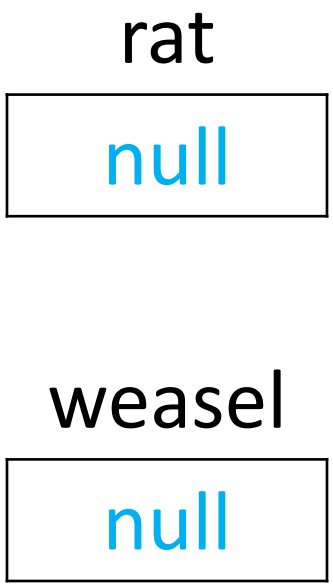
Reference Variables

- When you declare a primitive data item, such as a double or int, Java reserves some memory to store the data
- When you declare an object, Java does **not** reserve space for the object until it is created
- You can instantiate a new object with the keyword **new**

Object References

- When you declare an object, you are creating a reference to the object

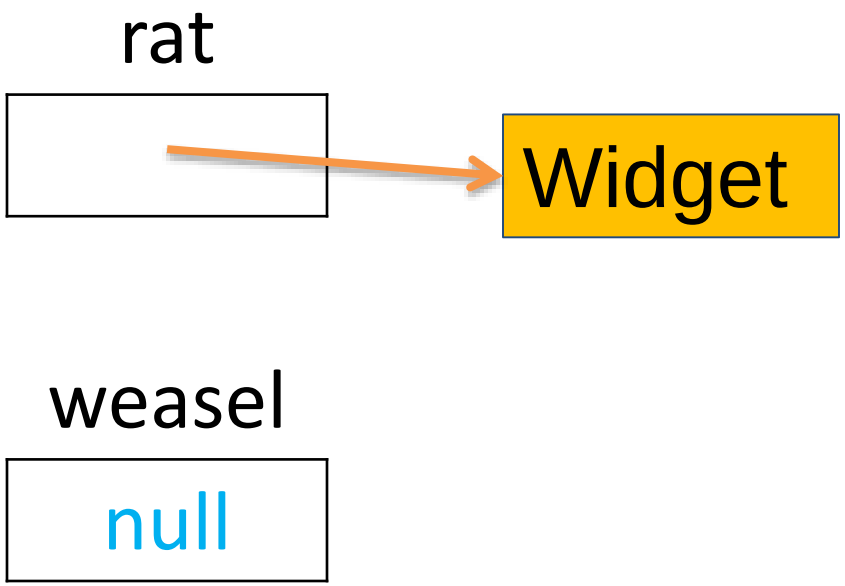
```
Widget rat;
Widget weasel;
```



Object References

- When you declare an object, you are creating a reference to the object

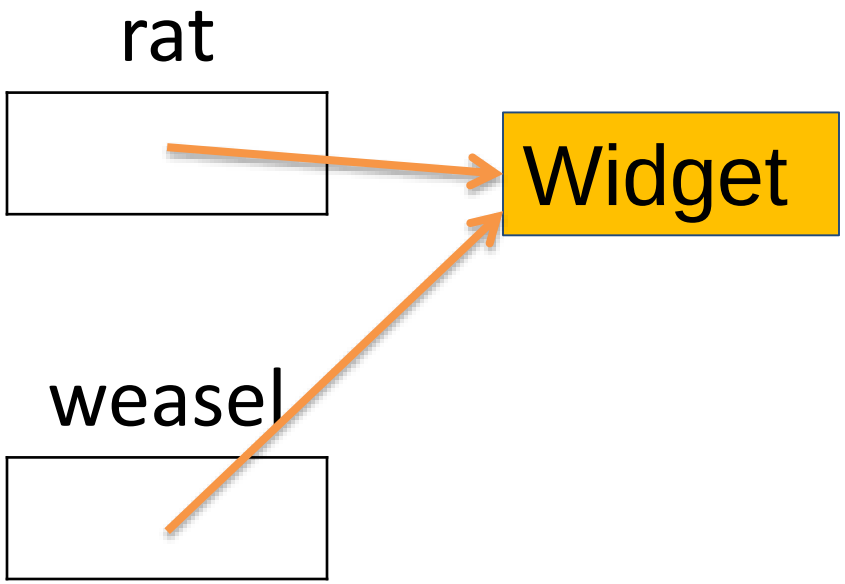
```
Widget rat;
Widget weasel;
rat = new Widget();
```



Object References

- When you declare an object, you are creating a reference to the object

```
Widget rat;
Widget weasel;
rat = new Widget();
weasel = rat;
```



- Changes to weasel will be seen in rat

Objects have Data

- An object can have data variables
- Data values are called **Instance Variables**, **Fields** or **Properties**
- Instance Variables can be primitive data types, such as double or int, or they can be other objects

What is displayed?

```
Widget cat = new Widget();
Widget dog = new Widget();
cat.setCount(5);
dog.setCount(7);
cat = dog;
dog.setCount(3);
int goat = cat.getCount();
System.out.println( goat );
```

A. 3

B. 5

C. 7

D. none of the above

Field Declaration

- Instance variables are declared within the class definition, but outside of any method

```
public class Student {
        int                 identifier;
        public double       grade;
        public String       name;
}
```

Accessing Object Fields

- The field variables of an object can be used by any method of that object just like a variable defined in the method

```
public class Student {
        int                     identifier;
        public double           grade;
        public String           name;
        public void setGrade(double score) {
                grade = score;
        }
}
```

Access Outside the Class

- Public field variables can be accessed from outside of the class
- To access a field variable, write the object name, a period and then the field variable's name

```
Student fred = new Student();
fred.name = "Fred Smith";
```

public and private

- **Public** data items can be accessed anywhere
- **Public** methods can be called by any part of the program
- **Private** data items can be accessed only by methods of the class
- **Private** methods can only be called by other methods of the class

Private example

```
public class Student {
        public    int        id;
        private   double     grade;
        public    String     name;
        public void setGrade(double score) {
                grade = score;
        }
}
```

Accessing the Example

- In another class, you can access the public but not the private values

```
public class myProg {
   public static void main(String[] x ) {
      Student fred = new Student();
      fred.name = "Sally";
      fred.grade = 4.0;        // not allowed
      fred.setGrade( 4.0 );  // allowed
   }
}
```

Data Abstraction

- Classes are described by their interface, that is, what they can do and how they are used
- The internal data and operation are hidden
- In this way if the internal operation is changed, programs using the class will not be impacted as long as the interface remains consistent

Accessing public Instance Values

```
public class Car {
        double miles, gallons;
        public double mileage;
        // other data and method are not shown
}

// in another program
Car junker = new Car();
double mpg = junker.mileage;
```

Accessing Instance Values by get Method

```
public class Car {
        double miles, gallons;
        private double mileage;
        public double getMileage() {
                return mileage;
        }
        // other data and method are not shown
}

// in another program
Car junker = new Car();
double mpg = junker.getMileage();
```

Accessing Instance Values by a Method

```
public class Car {
        double miles, gallons;

        public double getMileage() {
                return miles / gallons;
        }
        // other data and method are not shown
}

// in another program
Car junker = new Car();
double mpg = junker.getMileage();
```

Method Purpose

- Constructors
  - Initialize an object
- Modifiers
  - Change the values of an object
- Accessors
  - Return the value of a object without changing anything
- Function
  - Compute some value or perform some action

Constructors

- A constructor method is automatically called when an object is created
- The name of the constructor method is always the same as the class name
- Constructors can be used to initialize the values of an object's variables
- Constructors may or may not have parameters

Constructor Example

```
public class Widget {
   private int count = 0;
   /* constructor method */
   public Widget( int initial ) {
      count = initial;
   }
}
```

Another Constructor Example

```
public class Widget {
   private int count = 0;
   /* constructor method */
   public Widget( int count ) {
      this.count = count;
   }
}
```

this

- The Java keyword "**`this`**" means this object
- You can use **`this`** to access anything of the object, but not the class
- If you have a field named **`xyz`**, then you can access the field as **`this.xyz`**

Tryit

```
public class Rodent{
        double  rat;
        int       mouse;
    // Write a constructor method to initialize
    // the class variables
}

        // in another program using the Rodent class
Rodent mole = new Rodent( 5.6, 14 );
Rodent gerbil = new Rodent( 3.25,  8 );
```

Possible Constructor

```
public class Rodent{
        double  rat;
        int         mouse;
        public Rodent( double vole, int shrew ) {
                rat = vole;
                mouse = shrew;
        }
}
        // in a program using the Rodent class
Rodent mole = new Rodent( 5.6, 14 );
Rodent gerbil = new Rodent( 3.25,  8 );
```

Accessor Methods

- Accessor methods return some value from the object
- Accessor methods do not change anything in the object
- Examples of accessor methods include:
  - String **`length()`** method

Modifier Methods

- Modifier methods change the state of an object
- A modifier method may just set the value of a single variable or may perform a lengthy sequence of actions
- Modifier methods are often void methods
- Examples of modifier methods include:
  - setCount from the earlier example

Naming Convention

- Class names start with an uppercase letter
- Objects are written in lowercase
- Method names are in lowercase
- Constructors must have the same name as the class
- Accessor methods have names that can be used as nouns
- Modifier methods have names that can be used as verbs

Calling Methods

- In Java you can use a method of an object by writing the object's name, a period, the name of the method

```
Widget  thing = new Widget();
   thing.whatever( 5 );
```

- This calls the whatever method of Widget object thing passing it an integer 5 as a parameter

Inheritance

- You can extend an existing class to add new features
- The new class has all of the data values and methods of the parent classes

Inheritance Example

- Some of our programs have input numbers from the JTextField
- The getText method of JTextField only returns a string
- We can extend JTextField to add methods to get a double or an int

Extending JTextField

```
public class NumField extends
                        javax.swing.JTextField {
    public int getInt() {
        String text = getText();
        int number = Integer.parseInt(text);
        return number;
    }

    public double getDouble() {
        String text = getText();
        double number = Double.parseDouble(text);
        return number;
    }
}
```

Using NumField

```
NumField inYear = new NumField();
NumField inLoan = new NumField();


public void actionPerformed(
            java.awt.event.ActionEvent thing) {
  double loan = inLoan.getDouble();
  double m = inYear.getDouble() * 12.0;
  double pay = loan * etc.;
  answer.setText("Monthly payments of $"+pay);
}
```

MyCodeLab

- Read chapter 7 of the Java Illuminated textbook and answer the TuringsCraft questions for chapter 7
- You will earn 4 points for each correct answer up to a maximum of 100 points
- Complete this assignment by midnight on Thursday, March 17, 2016

Lab Quiz

- There will be a quiz in lab this Thursday
- A lab quiz is similar to a regular lab, but you must do it by yourself
- You may use your notes, book and the web
- You are not allowed to communicate with other people
- You must be in the Graham 203 lab

Exam

- The second GEEN163 exam will be in lecture on Wednesday, March 23, 2016
- It will cover everything since the first exam