Object-Oriented Principles in Java: Part I

Encapsulation

Information Hiding

Implementation Hiding

Creating Objects in Java

Class attributes vs. instance attributes

What Does Object-Oriented Mean?

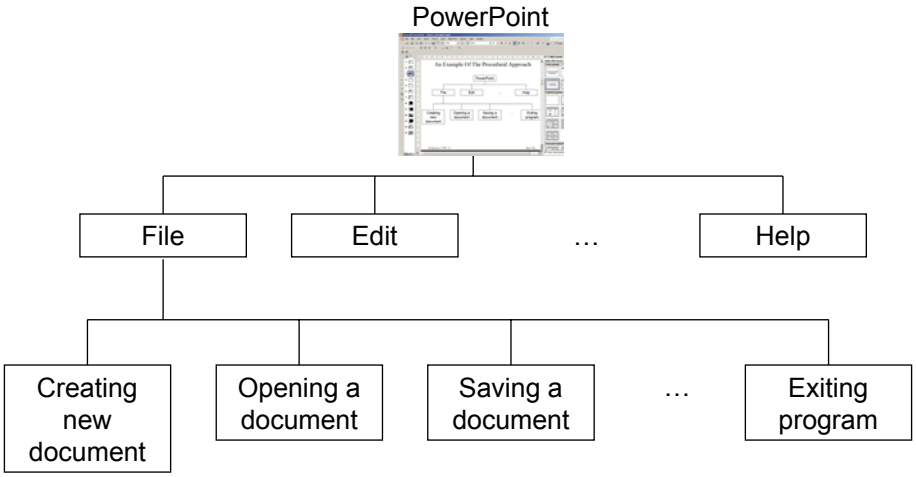
Procedural approach (CPSC 231)

- Design and build the software in terms of actions (verbs)

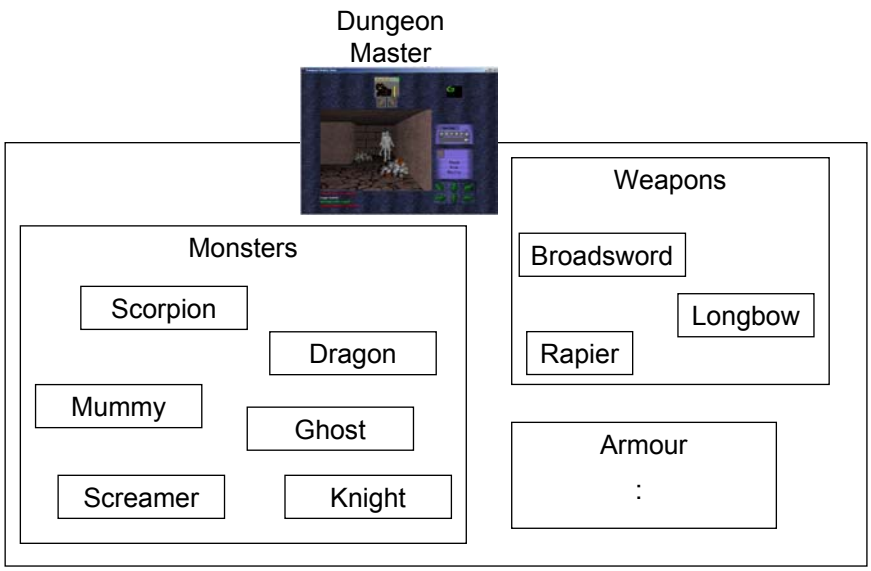
Object-Oriented approach (most of CPSC 233)

- Design and build the software in terms of things (nouns)

An Example Of The Procedural Approach



An Example Of The Object-Oriented Approach



Characteristics Of The Object-Oriented Approach

Two are covered in this section of notes:

- Encapsulation
- Hiding (information / implementation)

Characteristics Of The Object-Oriented Approach

Two are covered in this section of notes:

- **Encapsulation**
- Hiding (information / implementation)

Encapsulation

"Grouping together under a name"

You've already done this in CPSC 231:

- Problem decomposition: Procedures and functions in A5: Checkers
- Composite types: Records for A6 & A7: Movie list

Encapsulation Through Procedures And Functions

An example: Displaying the checker board

```
Current player is RED

   1 2 3 4 5 6 7 8
   - - - - - - - -
 1| |r| |r| |r| |r|
   - - - - - - - -
 2|r| |r| |r| |r| |
   - - - - - - - -
 3| |r| |r| |r| |r|
   - - - - - - - -
 4| | | | | | | | |
   - - - - - - - -
 5| | | | | | | | |
   - - - - - - - -
 6|w| |w| |w| |w| |
   - - - - - - - -
 7| |w| |w| |w| |w|
   - - - - - - - -
 8|w| |w| |w| |w| |
   - - - - - - - -
```

Encapsulation Through Procedures And Functions

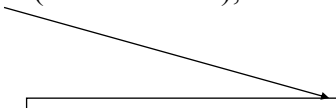
An example: Displaying the checker board

```
writeln('1 2 3 4 5 6 7 8 ':19);
for r := 1 to BOARDSIZE do
begin
  writeln('- - - - - - - - ':19);
  write(r:2, '|');
    for c := 1 to BOARDSIZE do
      begin
         write(checkerBoard[r][c], '|');
      end;
      writeln;
   end;
   writeln('- - - - - - - - ':19);
end
```


Encapsulation Through Procedures And Functions

An example: Displaying the checker board

displayBoard(checkerBoard);



```
procedure displayBoard (checkerBoard : Board);
begin
for r := 1 to BOARDSIZE do
     for c := 1 to BOARDSIZE do
         write(checkerBoard[r][c]);
end;
```

The Need For Encapsulation With Records

Each movie is described with several attributes:

- Movie name: *array [1..80] of char;*
- Cast: *array [1..5, 1..80] of char;*
- Genre: *array [1..80] of char;*
- Rating: *array [1..5] of char;*
- Number of stars: integer;
- Director: *array [1..80] of char;*
- Format:
  - DVD: *boolean;*
  - VHS: *boolean;*

```
proc (name, cast, genre, rating, stars, director,dvd,vhs);
```

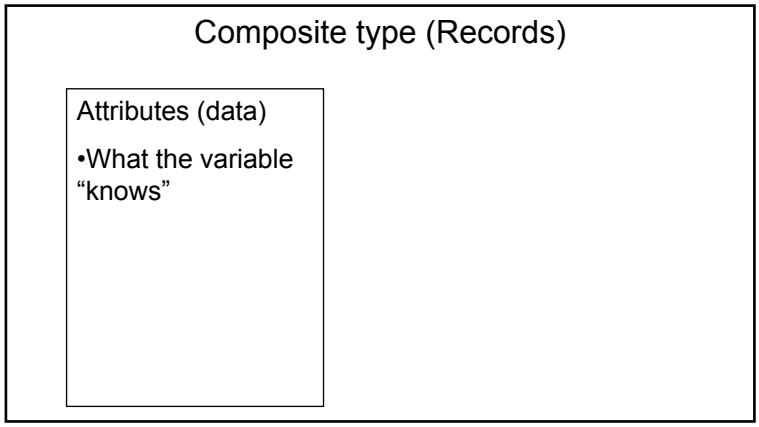

Encapsulation With Records

Each movie is described with several attributes:

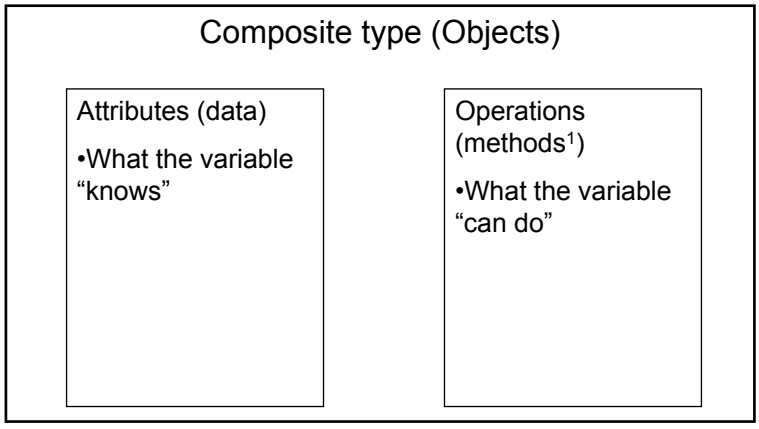
```
Movie = record
        name        : array [1..80] of char;
        cast        : array [1..80] of char;
        genre       : array [1..80] of char;
        rating      : array [1..5] of char;
        stars       : integer;
        director    : array [1..80] of char;
        dvd         : boolean;
        vhs         : boolean;
      end;
```

```
proc(movieVariable);
```

Encapsulation With The Procedural Approach (Pascal Records)



Encapsulation With The Object-Oriented Approach (Java Classes)



1 A method is the name for a procedure or function in Java

Example Objects: Monsters From Dungeon Master

Dragon 

•Scorpion 

•Couatl 

Monsters: Attributes

Represents information (may be common to all or unique to some):

•Name

•Speed

•Damage it inflicts

•Damage it can sustain

:

Monsters: Operations

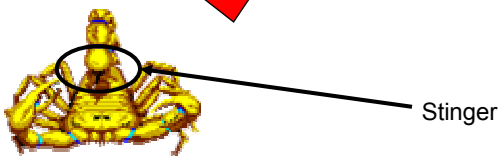
Represents what each monster can do

Often unique to a type of monster

Example

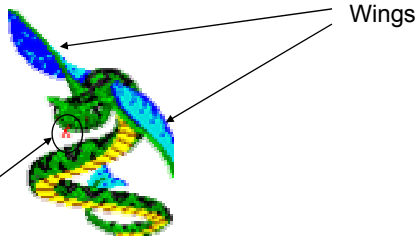
Dragon

Scorpion



Monsters: Operations

Couatl



Working With Objects In Java

Define the class

Declare an instance of the class (instantiate an object)

Accessing different parts of an object

Defining A Java Class

Format:

```
class <name of class>
{
    fields
    methods
}
```

Example:

```
class Foo
{
    int num;
    void displayNum ()
    {
        System.out.println(num);
    }
}
```

Instantiating Instances Of A Class

Format:

<*Class name*> <*instance name*> = new <Class name> ()*;*

Example:

Foo f = new Foo ();

Accessing Parts Of A Class

Format:

<*instance name*>.<*attribute name*>;

<instance name>.<*method name*>;

Example:

Foo f = new Foo ();

The Basics Of Classes: An Example

Example (The complete example can be found in the directory /home/profs/tamj/233/examples/inventoryClass/version1):

```
class Inventory
{
        final int CRITICAL = 10;
        int stockLevel;

          void addToInventory (int amount)
          {
               stockLevel = stockLevel + amount;
          }

          void removeFromInventory (int amount)
          {
              stockLevel = stockLevel - amount;
          }
```


The Basics Of Classes: An Example (2)

```
         boolean inventoryTooLow ()
         {
               if (stockLevel < CRITICAL)
                   return true;
               else
                   return false;
         }

        void displayInventoryLevel ()
        {
               System.out.println("No. items in stock: " + stockLevel);
         }
}
```

The Basics Of Classes: An Example (3)

```
import tio.*;
class Driver
{
   public static void main (String [] argv)
   {
      Inventory chinookInventory = new Inventory ();
      char menuOption;
      int amount;
      boolean temp;
      do
      {
         System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
         System.out.println("\t(A)dd new stock to inventory");
         System.out.println("\t(R)emove stock from inventory");
         System.out.println("\t(D)isplay stock level");
         System.out.println("\t(C)heck if stock level is critical");
         System.out.print("\t(Q)uit program");
         System.out.println();
```


The Basics Of Classes: An Example (4)

```
      System.out.print("Selection: ");
      menuOption = (char) Console.in.readChar();
      Console.in.readChar();
      System.out.println();

       switch (menuOption)
       {
          case 'A':
            System.out.print("No. items to add: ");
            amount = Console.in.readInt();
            Console.in.readChar();
            chinookInventory.addToInventory(amount);
            chinookInventory.displayInventoryLevel();
            break;
```

The Basics Of Classes: An Example (5)

```
            case 'R':
                System.out.print("No. items to remove: ");
                amount = Console.in.readInt();
                Console.in.readChar();
                chinookInventory.removeFromInventory(amount);
                chinookInventory.displayInventoryLevel();
                break;

            case 'D':
                chinookInventory.displayInventoryLevel();
                break;
```


The Basics Of Classes: An Example (6)

```
            case 'C':
                temp = chinookInventory.inventoryTooLow();
                if (chinookInventory.inventoryTooLow())
                   System.out.println("Stock levels critical!");
                else
                   System.out.println("Stock levels okay");
                chinookInventory.displayInventoryLevel();
                break;

            case 'Q':
                System.out.println("Quitting program");
                break;

            default:
                System.out.println("Enter one of A, R, D, C or Q");
        }
    } while (menuOption != 'Q');
}
```

Characteristics Of The Object-Oriented Approach

Two are covered in this section of notes:

- Encapsulation
- **Hiding (information / implementation)**

Information Hiding

Protects the inner-workings (data) of a class

Only allow access to the core of an object in a controlled fashion

Information Hiding (2)

In Java use the keyword *private* for data

Private parts of a class can only be accessed through the methods of that class

Format:

private <attribute type> <*attribute name*>;

Example:

private int num;

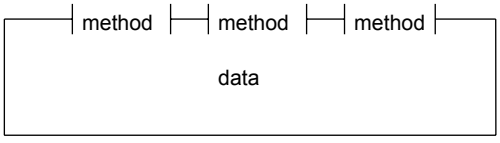
Rule of thumb:

- Unless there is a compelling reason, the data for a class should be hidden (private)

Accessing Hidden Information

In Java you <u>can</u> use the keyword *public* for methods

The public parts of a class can be accessed anywhere in the program (used to access the private parts)



Format:

public <method return type> <*method name*> (<parameters>)

Accessing Hidden Information (2)

Example:

```
public void addToInventory (int amount)
{
        stockLevel = amount;
}
```


How Does Hiding Information Protect The Class?

Protects the inner-workings (data) of a class

e.g., range checking for inventory levels (0 – 100)

The complete example can be found in directory:
/home/profs/tamj/233/examples/inventoryClass/version2

How Does Hiding Information Protect The Class?

```
class Inventory
{
   private final static int CRITICAL = 10;
   private final static int MIN = 0;
   private final static int MAX = 100;

  private int stockLevel;
```


How Does Hiding Information Protect The Class? (2)

```
  public void addToInventory (int amount)
  {
     int temp;
     temp = stockLevel + amount;
     if (temp > MAX)
     {
        System.out.print("Adding " + amount + " item will cause stock ");
        System.out.println("to become greater than " + MAX + " units");
     }
     else
     {
        stockLevel = stockLevel + amount;
     }
  }
```

How Does Hiding Information Protect The Class? (3)

```
public void removeFromInventory (int amount)
{
      int temp;
      temp = stockLevel - amount;
      if (temp < MIN)
      {
         System.out.println();
         System.out.print("Removing " + amount + " item will cause stock ");
         System.out.println("to become less than " + MIN + " units");
      }
      else
      {
         stockLevel = temp;
      }
}
```


How Does Hiding Information Protect The Class? (4)

```
  public boolean inventoryTooLow ()
  {
      if (stockLevel < CRITICAL)
         return true;
      else
         return false;
  }

  public void displayInventoryLevel ()
  {
       System.out.println("No. items in stock: " + stockLevel);
  }
}
```

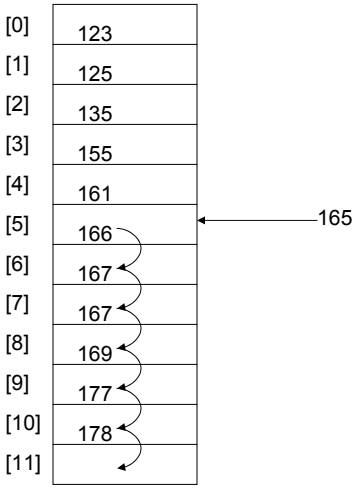
Implementation Hiding

Allows you to use a program module (e.g., a method) but you don't care about how the code in the module (implementation) was written.

For example, a list can be implemented as either an array or as a linked list.

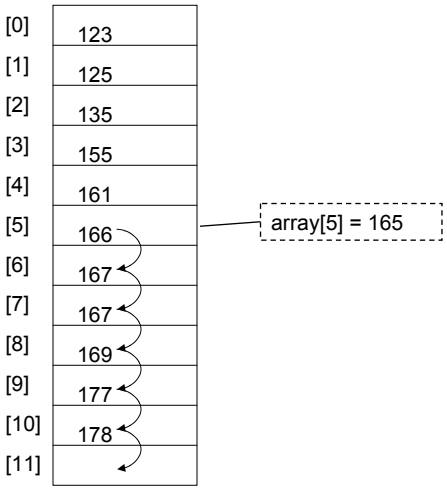
Implementation Hiding (2)

List implemented as an array (add element)



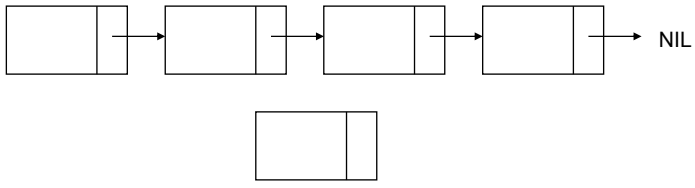
Implementation Hiding (2)

List implemented as an array (add element)



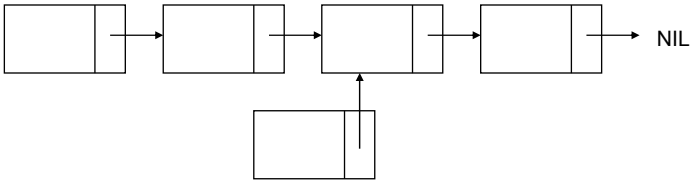
Implementation Hiding (3)

List implemented as a linked list (add element)



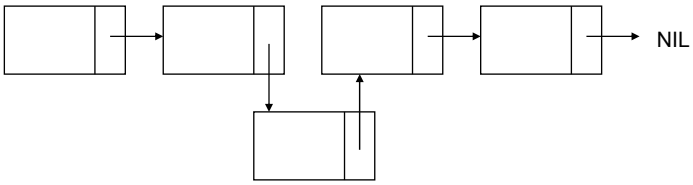
Implementation Hiding (3)

List implemented as a linked list (add element)



Implementation Hiding (3)

List implemented as a linked list (add element)



Implementation Hiding (4)

Changing the implementation of the list should have a minimal impact on the rest of the program

The “add” method is a black box.

We know how to use it without being effected by the details of how it works.

add (list, newElement)

Instantiating Objects: The Constructor

A method that is used to initialize objects as they are being instantiated (automatically called)

If no constructor is specified then the **default constructor** is called

e.g., Sheep jim = new Sheep();

Creating Your Own Constructor

Format:

```
public <class name> (<parameters>)
{
    // Statements to initialize the fields of the class
}
```

Example:

```
public Sheep ()
{
    System.out.println("Creating \"No name\" sheep");
    name = "No name";
}
```


Overloading The Constructor

Method overloading:

- Creating different versions of a method
- Each version is distinguished by the number, type and order of the parameters

```
public Sheep ()
public Sheep (String name)
```

- Different methods can have different return types (but must be distinguished from each other in another fashion)

Using Constructors: An Example

The complete example can be found in the directory
/home/profs/tamj/233/examples/sheepClass/version1

```
class Sheep
{
   private String name;

   public Sheep ()
   {
      name = "No name";
   }
   public Sheep (String name)
   {
      this.name = name;
   }
```

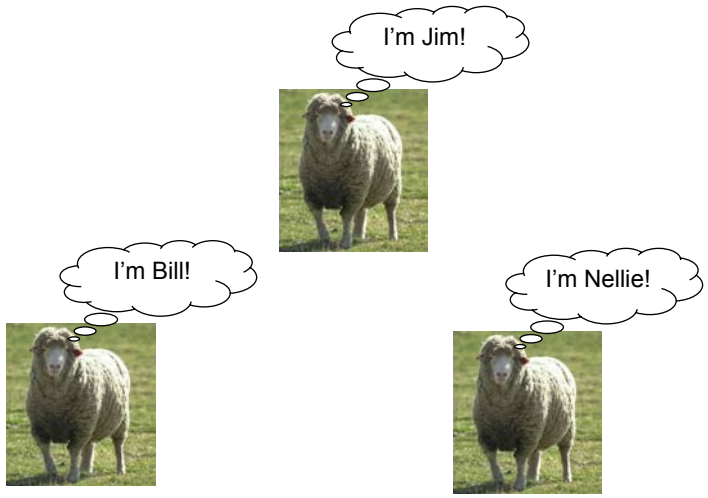

Using Constructors: An Example (2)

```
    public String getName ()
    {
        return name;
    }

    public void changeName (String name)
    {
        this.name = name;
    }
}
```

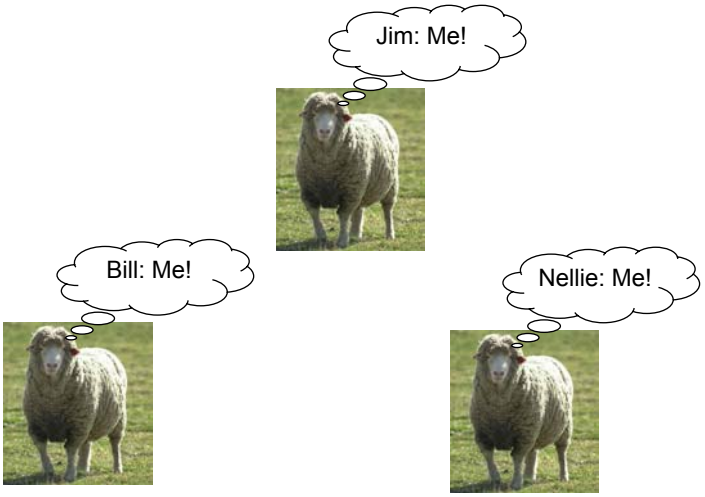
Using Constructors: An Example (3)

```
class Driver
{
   public static void main (String [] argv)
   {
      System.out.println();
      System.out.println("Creating herd...");
      Sheep nellie = new Sheep ("Nellie");
      Sheep bill = new Sheep("Bill");
      Sheep jim = new Sheep();
      jim.changeName("Jim");
      System.out.println("\t"+ nellie.getName());
      System.out.println("\t"+ bill.getName());
      System.out.println("\t"+ jim.getName());
      System.out.println();
   }
}
```


We Now Have Several Sheep



Question: Who Tracks The Size Of The Herd?



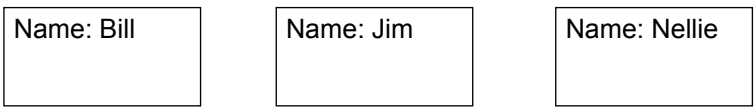
Answer: None Of The Above

Information about all instances of a class should not be tracked by an individual object

So far we have used instance fields

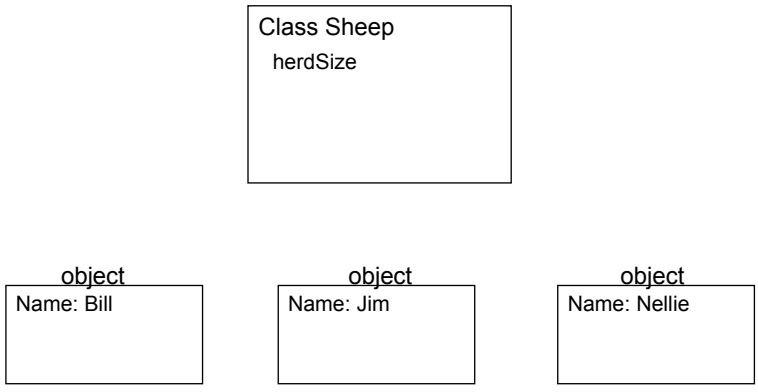
Each *instance* of an object contains *it's own set of instance fields* which can contain information unique to the instance

```
class Sheep
{
      private String name;
          :       :       :
}
```



Static: Revisted

Static fields: One instance of the field exists for the class (not for the instances of the class)



Static: Revisted (2)

Static methods (class methods):

- Act on the class fields
- Are associated with the class as a whole and not individual instances of the class

Static Revisited: An Example

The complete example can be found in the directory:
/home/profs/tamj/233/examples/sheepClass/version2

```
class Sheep
{
   // Class variable: Tracks the total number of sheep
   private static int herdSize;

   // Instance variable: Unique to each individual sheep
   private String name;

   // Constructors
   public Sheep ()
   {
      herdSize++;
      System.out.println("Creating \"No name\" sheep");
      name = "No name";
   }
```


Static Revisited: An Example (2)

```
   public Sheep (String name)
   {
      herdSize++;
      System.out.println("Creating the sheep called " + name);
      this.name = name;
   }

   // Class method
   public static int getHerdSize ()
   {
      return herdSize;
   }

   // Instance methods
   public String getName ()
   {
      return name;
   }
```

Static Revisited: An Example (3)

```
    public void changeName (String name)
    {
       this.name = name;
    }
}
```


Static Revisited: An Example (4)

```
class Driver
{
   public static void main (String [] argv)
   {
      System.out.println();
      System.out.println("You have " + Sheep.getHerdSize() + " sheep");
      System.out.println("Creating herd...");
      Sheep nellie = new Sheep ("Nellie");
      Sheep bill = new Sheep("Bill");
      Sheep jim = new Sheep();
      System.out.println("You now have " + Sheep.getHerdSize() + " sheep:");
      jim.changeName("Jim");
      System.out.println("\t"+ nellie.getName());
      System.out.println("\t"+ bill.getName());
      System.out.println("\t"+ jim.getName());
      System.out.println();
   }
}
```

Rules Of Thumb: Instance Vs. Class Fields

If a field or information can differ between instances of a class:

- The field probably should be an instance field

If the field or information relates to the class or to all instances of the class

- The field probably should be a static field of the class

Rules Of Thumb: Instance Vs. Class Methods

If a method acts on instance fields of individual objects:

- The method should probably be an instance method

If a method acts on static class fields or is not associated with multiple objects

- The method should probably be a static class method

Example Case: The System!

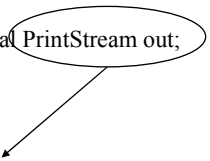
There is one computer system on which a Java program is running on (not multiple instances of the system).

```
public class System
{
        public static final PrintStream out;
                :
}

public class PrintStream
{
        public void print (boolean b);
        public void print (char c);
                :
}
```


Example Case: The System!

There is one computer system on which a Java program is running on (not multiple instances of the system).

```
public class System
{
        public static final PrintStream out;
                :
}

public class PrintStream
{
        public void print (boolean b);
        public void print (char c);
                :
}
```



Objects As Parameters

In Java all parameters are passed by value
Passing an object into a method actually passes a reference to the object

Do changes to the object made in the method still exist after the method exist?

•Answer: It depends!

The following example can be found in the directory:
/home/profs/tamj/233/examples/parameterExample

Objects As Parameters (2)

```
class IntegerWrapper
{
   private int num;
   public void setNum (int no)
   {
      num = no;
   }

   public int getNum ()
   {
      return num;
   }
}
```

Objects As Parameters (3)

```
class Driver
{
   public static void methodOne (IntegerWrapper iw)
   {
      IntegerWrapper temp = new IntegerWrapper ();
      temp.setNum(10);
      iw = temp;
      System.out.println("Method One");
      System.out.println("iw.num "+ iw.getNum() + "\t" + "temp.num " +
         temp.getNum());
   }

   public static void methodTwo (IntegerWrapper iw)
   {
      iw.setNum(10);
      System.out.println("Method Two");
      System.out.println("iw.num "+ iw.getNum());
   }
```


Objects As Parameters (4)

```
 public static void main (String [] argv)
{
      IntegerWrapper iw = new IntegerWrapper();
      iw.setNum(1);
      System.out.println("Main method");
      System.out.println("iw.num "+ iw.getNum());
      System.out.println();
      methodOne(iw);

      System.out.println("Main method");
      System.out.println("iw.num "+ iw.getNum());

      methodTwo(iw);
      System.out.println("Main method");
      System.out.println("iw.num "+ iw.getNum());
   }
}
```

Summary

The difference between the Procedural and the Object-Oriented approaches?

Some fundamental Object Oriented principles (in Java)

- Classes and instances
- Encapsulation
- Hiding (information / implementation)
- Fields unique to individual instances vs. data that is associated with the class
- Instance methods vs. class methods (static)

How are objects passed as parameters to methods?