CPSC 233: Introduction to Classes and Objects, Part I

- Attributes and methods
- Creating new classes
- References: Dynamic memory allocation and automatic garbage collection
- Encapsulation and information hiding
- Constructors
- Shadowing
- Arrays

What Does Object-Oriented Mean?

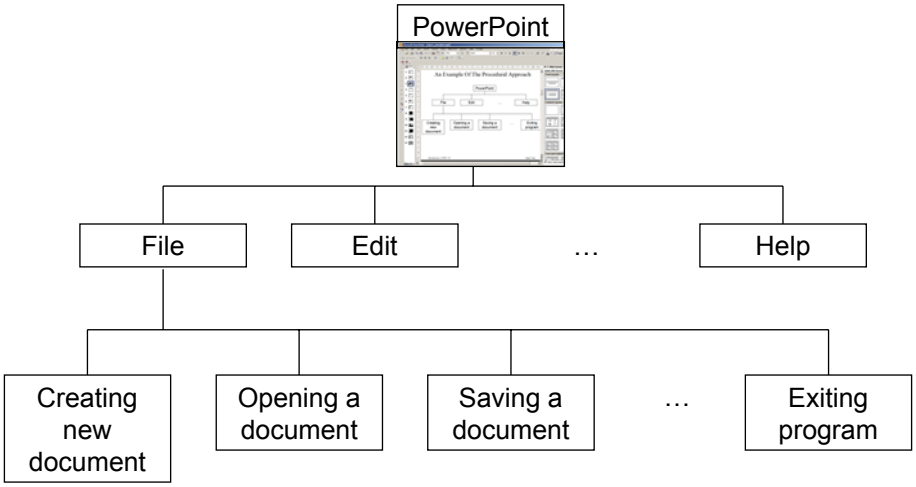
Procedural approach (CPSC 231)

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

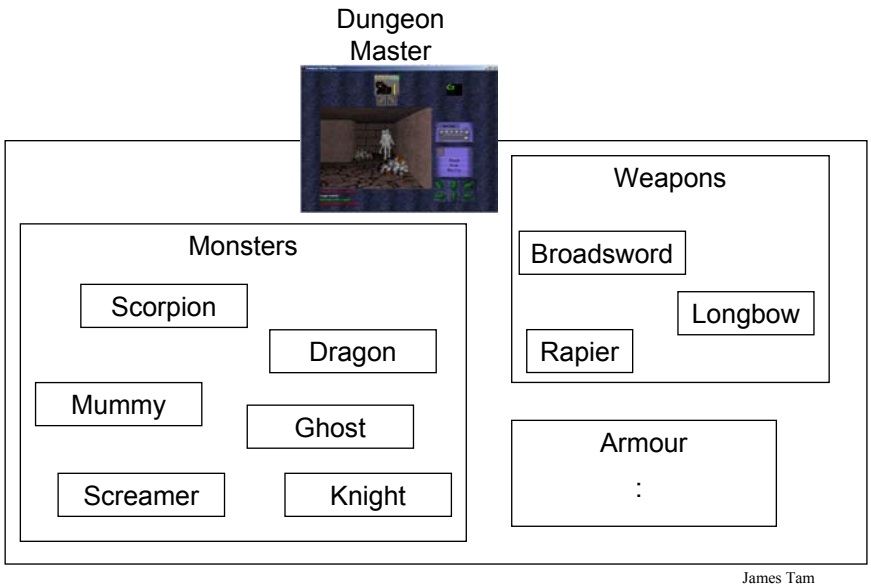
Object-Oriented approach (CPSC 233)

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

An Example Of The Procedural Approach



An Example Of The Object-Oriented Approach



Example Objects: Monsters From Dungeon Master

Dragon 

Scorpion 

Couatl 

Ways Of Describing A Monster

What are the dragon's attributes?

What can the dragon do?

Monsters: Attributes

Represents information about the monster:

- Name
- Damage it inflicts
- Damage it can sustain
- Speed

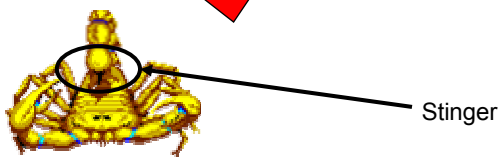
:

Monsters: Operations

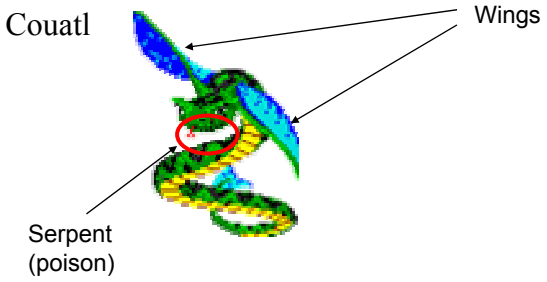
Represents what each monster can do (verb part):

Dragon

Scorpion



Monsters: Operations

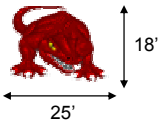


Pascal Records Vs. Java Objects

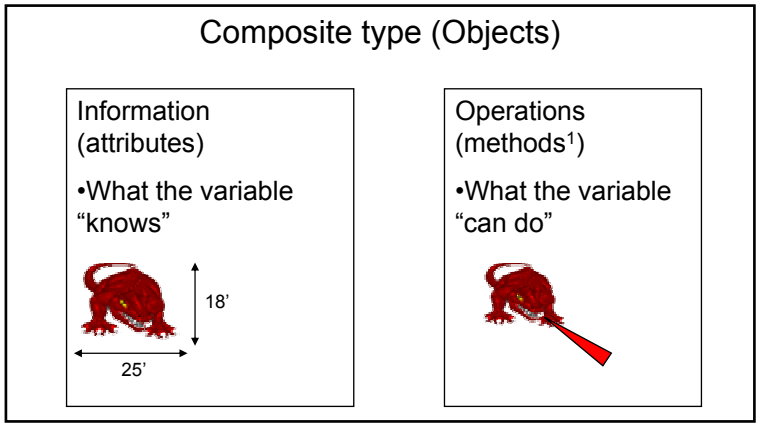
Composite type (Records)

Information
(attributes)

•What the variable
"knows"



Pascal Records Vs. Java Objects



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

Information Hiding

An important part of Object-Oriented programming

Protects the inner-workings (data) of a class

Only allow access to the core of an object in a controlled fashion (use the *public* parts to access the *private* sections)

Illustrating The Need For Information Hiding: An Example

Creating a new monster: “The Critter”
Attribute: Height (must be 60” – 72”)



James Tam

Illustrating The Need For Information Hiding: An Example

Creating a new monster: “The Critter”
Attribute: Height (must be 60” – 72”)




James Tam

Working With Objects In Java

I) Define the class
II) Create an instance of the class (instantiate an object)
III) Using different parts of an object

I) Defining A Java Class

Format of class definition:

```
class <name of class>
{
    instance fields/attributes
    instance methods
}
```

Defining A Java Class (2)

Format of instance fields:

<access modifier>[1] *<type of the field> <name of the field>*;

Format of instance methods:

<access modifier>[1] *<return type*[2]*> <method name>* (*<p1 type> <p1 name>*…)
{
    <Body of the method>
}

1) Can be public or private but typically instance fields are private while instance methods are public

2) Valid return types include the simple types (e.g., int, char etc.), predefined classes (e.g., String) or new classes that you have defined in your program. A method that returns nothing has return type of "void".

Defining A Java Class (3)

Example:

```
class Foo
{
    private int num;
    public void greet ()
    {
        System.out.println("Hello");
    }
}
```

II) Creating Instances Of A Class

Format:

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

Example:

```
Foo f = new Foo ();
```

Note: “f” is not an object of type “Foo” but a reference to an object of type “Foo”.

References

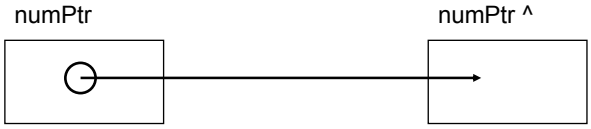
It is a pointer that cannot be de-referenced by the programmer

Automatically garbage collected when no longer needed

De-Referencing Pointers: Pascal Example I

```
var
        numPtr : ^ integer;

begin
        new(numPtr);
```



De-Referencing Pointers: Pascal Example II

```
type
  Client =  record
              firstName : array [1..24] of char;
              lastName  : array [1..24] of char;
              income    : real;
              email     : array [1..50] of char;
            end; (* Declaration of record Client *)

  NodePointer = ^ Node;
  Node = record
           data           : Client;
           nextPointer : NodePointer;
         end; (* Declaration of record Node *)
            :                    :
            :                    :
      writeln('First name: ':20, currentNode^.data.firstName);
```

III) Using The Parts Of A Class

Format:

<instance name>.<attribute name>;

<instance name>.<method name>;

Example:

```
Foo f = new Foo ();
f.greet();
```

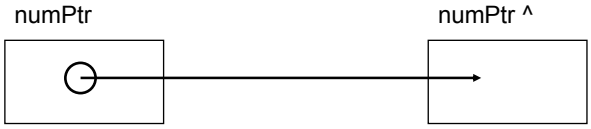
Note: In order to use the dot-operator "." the instance field or method cannot have a private level of access

Java References

It is a pointer that cannot be de-referenced by the programmer

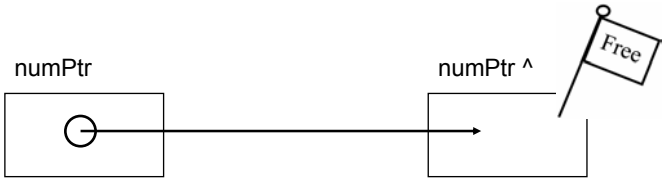
Automatically garbage collected when no longer needed

Garbage Collection And Pointers: Pascal Example



Garbage Collection And Pointers: Pascal Example

```
dispose(numPtr);
```

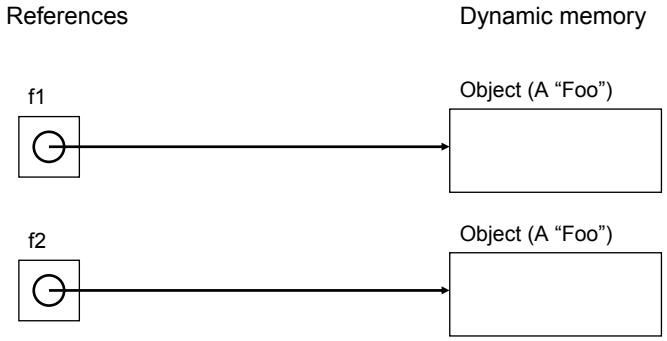


Garbage Collection And Pointers: Pascal Example

```
dispose(numPtr);
numPtr := NIL;
```

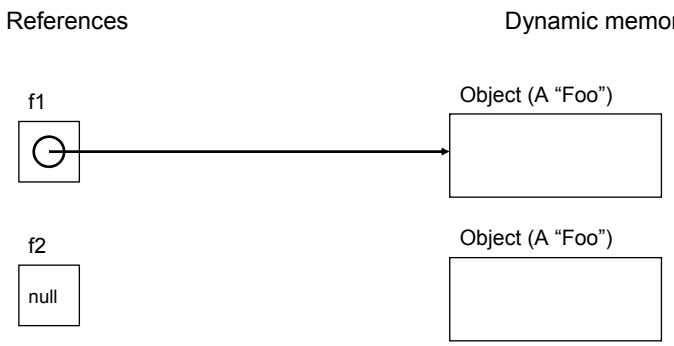
Automatic Garbage Collection Of Java References

Dynamically allocated memory is automatically freed up when it is no longer referenced



Automatic Garbage Collection Of Java References (2)

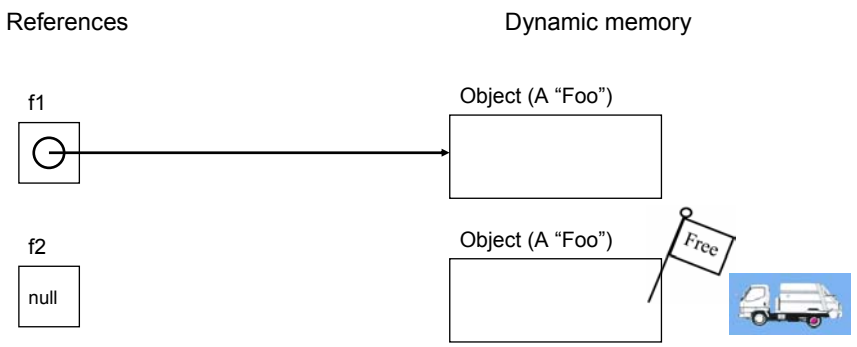
Dynamically allocated memory is automatically freed up when it is no longer referenced e.g., f2 = null;



James Tam

Automatic Garbage Collection Of Java References (2)

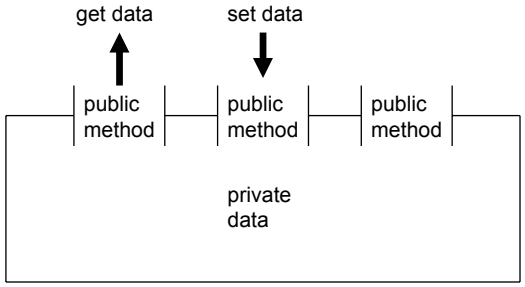
Dynamically allocated memory is automatically freed up when it is no longer referenced e.g., f2 = null;



James Tam

Public And Private Parts Of A Class

The public methods can be used do things such as access or change the instance fields of the class



Public And Private Parts Of A Class (2)

Types of methods that utilize the instance fields:

1) Accessor methods "get"
   - Used to determine the current value of a field
   - Example:

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

2) Mutator methods "set"
   - Used to set a field to a new value
   - Example:

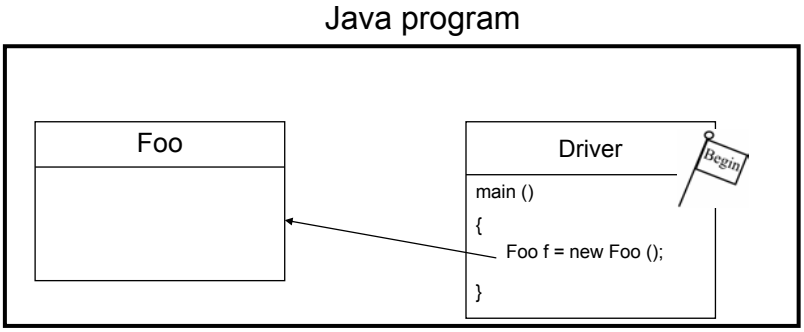
```
public void setNum (int n)
{
   num = n;
}
```

Laying Out Your Program

The program must contain a "driver" class

The driver class is the place where the program starts running (contains the one, and only one, main method)

Instances of other classes can be created here



Points To Keep In Mind About The Driver Class

- Contains the only main method of the whole program (where execution begins)
- Do not instantiate instances of the Driver[1]
- For now avoid:
  - Defining instance fields / attributes for the Driver[1]
  - Defining methods for the Driver (other than the main method)[1]

1 Details will be provided later in this course

Putting It Altogether: First Object-Oriented Example

Example (The complete example can be found in the directory /home/233/examples/classes_objects/firstExample

```
class Driver
{
   public static void main (String [] args)
   {
      Foo f = new Foo ();
      f.setNum(10);
      System.out.println("Current value of num = " + f.getNum());
   }
}
```


Putting It Altogether: First Object-Oriented Example (2)

```
class Foo
{
    private int num;
    public void setNum (int n)
   {
      num = n;
   }

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

UML[1] Representation Of A Class

<Name of class>
-<attribute *name*>: <attribute *type*> +<method name> ()

Foo
-num: int +setNum () +getNum ()

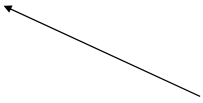
UML = Unified Modeling Language

Common Errors When Using References

- Forgetting to initialize the reference
- Using a null reference

Error: Forgetting To Initialize The Reference

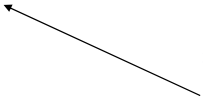
```
Foo f;
f.setNum(10);
```



```
Compilation error!
> javac Driver.java
Driver.java:14: variable f might not have been initialized
        f.setNum(10);
        ^
1 error
```


Error: Using Null References

```
Foo f = null;
f.setNum(10);
```



```
Run-time error!
> java Driver
Exception in thread "main"
java.lang.NullPointerException
        at Driver.main(Driver.java:14)
```

Encapsulation

Grouping data methods together within a class definition to allow the private attributes to be accessible only through the public methods.

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 the directory
/home/233/examples/classes_objects/secondExample

Showing The Need For Information Hiding: An Example

```
class Driver
{
   public static void main (String [] args)
   {
      Inventory chinookInventory = new Inventory ();
      int menuSelection;
      int amount;
```


Showing The Need For Information Hiding: An Example (2)

```
      do
      {
         System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
         System.out.println("\t(1)Add new stock to inventory");
         System.out.println("\t(2)Remove stock from inventory");
         System.out.println("\t(3)Display stock level");
         System.out.println("\t(4)Check if stock level is critically low");
         System.out.println("\t(5)Quit program");
         System.out.print("Selection: ");
         menuSelection = Console.in.readInt();
         Console.in.readChar();
         System.out.println();
```

Showing The Need For Information Hiding: An Example (3)

```
switch (menuSelection)
{
    case 1:
       System.out.print("No. items to add: ");
       amount = Console.in.readInt();
       Console.in.readChar();
       chinookInventory.stockLevel = chinookInventory.stockLevel
         + amount;
       System.out.println(chinookInventory.getInventoryLevel());
       break;

    case 2:
       System.out.print("No. items to remove: ");
       amount = Console.in.readInt();
       Console.in.readChar();
       chinookInventory.stockLevel = chinookInventory.stockLevel
         - amount;
       System.out.println(chinookInventory.getInventoryLevel());
       break;
```


Showing The Need For Information Hiding: An Example (4)

```
   case 3:
       System.out.println(chinookInventory.getInventoryLevel());
       break;

    case 4:
       if (chinookInventory.inventoryTooLow())
          System.out.println("Stock levels critical!");
       else
          System.out.println("Stock levels okay");
       System.out.println(chinookInventory.getInventoryLevel());
       break;

    case 5:
       System.out.println("Quitting program");
       break;
```

Showing The Need For Information Hiding: An Example (5)

```
            default:
                System.out.println("Enter one of 1, 2, 3, 4 or 5");
            }
        } while (menuSelection != 5);
    }
}
```


Showing The Need For Information Hiding: An Example (6)

```
class Inventory
{
   public int stockLevel;

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

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

Showing The Need For Information Hiding: An Example (7)

```
 public boolean inventoryTooLow ()
{
    final int CRITICAL = 10;
   if (stockLevel < CRITICAL)
         return true;
   else
      return false;
}

public String getInventoryLevel ()
{
   return("No. items in stock: " + stockLevel);
}
}
```


Utilizing Information Hiding: An Example

The complete example can be found in the directory
/home/233/examples/classes_objects/thirdExample

```
class Driver
{
   public static void main (String [] args)
   {
      Inventory chinookInventory = new Inventory ();
      int menuSelection;
      int amount;
```

Utilizing Information Hiding: An Example (2)

```
do
{
   System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
   System.out.println("\t(1)Add new stock to inventory");
   System.out.println("\t(2)Remove stock from inventory");
   System.out.println("\t(3)Display stock level");
   System.out.println("\t(4)Check if stock level is critically low");
   System.out.println("\t(5)Quit program");
   System.out.print("Selection: ");
   menuSelection = Console.in.readInt();
   Console.in.readChar();
   System.out.println();
```


Utilizing Information Hiding: An Example (3)

```
   switch (menuSelection)
   {
      case 1:
         System.out.print("No. items to add: ");
         amount = Console.in.readInt();
         Console.in.readChar();
         chinookInventory.addToInventory(amount);
         System.out.println(chinookInventory.getInventoryLevel());
         break;

      case 2:
         System.out.print("No. items to remove: ");
         amount = Console.in.readInt();
         Console.in.readChar();
         chinookInventory.removeFromInventory(amount);
         System.out.println(chinookInventory.getInventoryLevel());
         break;
```

Utilizing Information Hiding: An Example (4)

```
            case 3:
               System.out.println(chinookInventory.getInventoryLevel());
               break;

            case 4:
               if (chinookInventory.inventoryTooLow())
                  System.out.println("Stock levels critical!");
               else
                  System.out.println("Stock levels okay");
               System.out.println(chinookInventory.getInventoryLevel());
               break;

            case 5:
               System.out.println("Quitting program");
               break;
```


Utilizing Information Hiding: An Example (5)

```
            default:
               System.out.println("Enter one of 1, 2, 3, 4 or 5");
         }
      } while (menuSelection != 5);
   }
}
```

Utilizing Information Hiding: An Example (6)

```
class Inventory
{
    private int stockLevel;
   public void addToInventory (int amount)
    {
        final int MAX = 100;
        int temp;
       temp = stockLevel + amount;
       if (temp > MAX)
       {
          System.out.println();
          System.out.print("Adding " + amount + " item will cause stock ");
          System.out.println("to become greater than " + MAX + " units
             (overstock)");
       }
```


Utilizing Information Hiding: An Example (7)

```
       else
       {
          stockLevel = stockLevel + amount;
       }
   } // End of method addToInventory
```

Utilizing Information Hiding: An Example (8)

```
public void removeFromInventory (int amount)
{
   final int MIN = 0;
   int temp;
   temp = stockLevel - amount;
   if (temp < MIN)
   {
      System.out.print("Removing " + amount + " item will cause stock ");
      System.out.println("to become less than " + MIN + " units (understock)");
   }
   else
   {
      stockLevel = temp;
   }
}
```

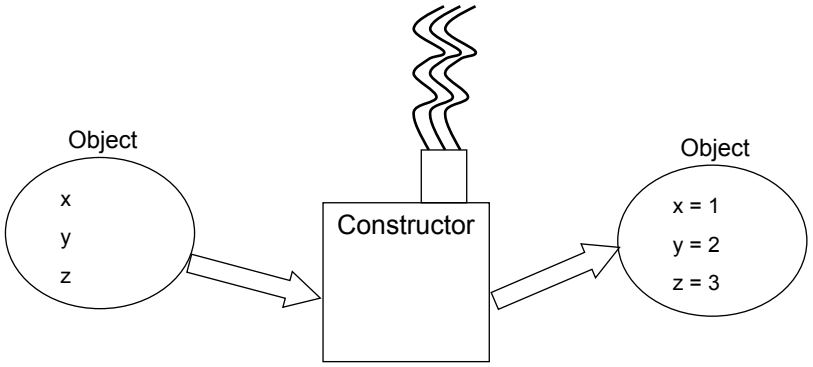

Utilizing Information Hiding: An Example (9)

```
 public boolean inventoryTooLow ()
{
   final int CRITICAL = 10;
   if (stockLevel < CRITICAL)
      return true;
   else
      return false;
}

public String getInventoryLevel ()
{
   return("No. items in stock: " + stockLevel);
}
}     // End of class Inventory
```

Creating Objects With The Constructor

A method that is used to initialize the attributes of an object as the objects are instantiated (automatically called)



Creating Objects With The Constructor (2)

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

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

Writing Your Own Constructor

Format (Note: *Constructors have no return type*):

```
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

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

```
public Sheep ()
public Sheep (String n)
```

Things to avoid when overloading constructors

1) Distinguishing constructors solely by the order of the parameters

2) Overloading constructors but having identical bodies for each

Constructors: An Example

The complete example can be found in the directory
/home/233/examples/classes_objects/fourthExample

```
class Driver
{
   public static void main (String [] args)
   {
      Sheep nellie, bill, jim;
      System.out.println();
      System.out.println("Creating flock...");
      nellie = new Sheep ("Nellie");
      bill = new Sheep("Bill");
      jim = new Sheep();
```


Constructors: An Example (2)

```
      jim.changeName("Jim");
      System.out.println("Displaying updated flock");
      System.out.println("\t"+ nellie.getName());
      System.out.println("\t"+ bill.getName());
      System.out.println("\t"+ jim.getName());
      System.out.println();
   }
}
```

Constructors: An Example (3)

```
class Sheep
{
   private String name;

    public Sheep ()
    {
       System.out.println("Creating \"No name\" sheep");
       name = "No name";
    }
    public Sheep (String n)
    {
       System.out.println("Creating the sheep called " + n);
       name = n;
    }
```


Constructors: An Example (4)

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

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

Shadowing

1) When a variable local to the method of a class has the same name as an attribute of that class.
   - Be cautious of accidentally doing this.

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


Arrays In Java

Important points to remember for arrays in Java:

- An array of n elements will have an index of zero for the first element up to (n-1) for the last element
- The array index must be an integer
- Arrays employ dynamic memory allocation (references)
- Several error checking mechanisms are available

Arrays In Java

Important points to remember for arrays in Java:

- An array of n elements will have an index of zero for the first element up to (n-1) for the last element
- The array index must be an integer
- **Arrays employ dynamic memory allocation (references)**
- Several error checking mechanisms are available

Arrays In Java

Important points to remember for arrays in Java:

- An array of n elements will have an index of zero for the first element up to (n-1) for the last element
- The array index must be an integer
- **Arrays employ dynamic memory allocation (references)**
- Several error checking mechanisms are available

James Tam

Declaring Arrays

Arrays in Java involve a reference to the array so declaring an array requires two steps:

1) Declaring a reference to the array
2) Allocating the memory for the array

Declaring A Reference To An Array

Format:

<*type*> [] <array *name*>;

Example:

```
int [] arr;
int [][] arr;
```

Allocating Memory For An Array

Format:

<*array name*> = new <*array type*> [<*no elements*>];

Example:

arr = new int[SIZE];
arr = new int[SIZE][SIZE];

(Or combining both steps together):
int [] arr = new int[SIZE];

Arrays: An Example

```
int i, len;
int [] arr;
System.out.print("Enter the number of array elements: ");
len = Console.in.readInt();
arr = new int [len];
System.out.println("Array Arr has " + arr.length + " elements.");
for (i = 0; i < arr.length; i++)
{
   arr[i] = i;
   System.out.println("Element[" + i + "]=" + arr[i]);
}
```

Arrays In Java

Important points to remember for arrays in Java:

- An array of n elements will have an index of zero for the first element up to (n-1) for the last element
- The array index must be an integer
- Arrays involve dynamic memory allocation (references)
- Several error checking mechanisms are available
  - Using a null array reference
  - Array bounds checking

Using A Null Reference

```
int [] arr = null;
arr[0] = 1;
```

NullPointerException

Exceeding The Array Bounds

```
int [] arr  = new int [4];
int i;
for (i = 0; i <= 4; i++)
  arr[i] = i;
```

ArrayIndexOutOfBoundsException
(when i = 4)

Arrays Of Objects (References)

- An array of objects is actually an array of references to objects

  e.g., Foo [] arr = new Foo [4];

- The elements are initialized to null by default

```
arr[0].setNum(1);
```

NullPointerException

Arrays Of References To Objects: An Example

The complete example can be found in the directory
/home/233/examples/classes_objects/fourthExample

```
class Driver
{
   public static void main (String [] args)
   {
      Foo [] arr = new Foo [4];
      int i;
      for (i = 0; i < 4; i++)
      {
         arr[i] = new Foo (i);
         System.out.println(arr[i].getNum());
      }
   }
}
```


Arrays Of References To Objects: An Example (2)

```
class Foo
{
   private int num;

   public Foo ()
   {
      num = 0;
   }

   public Foo (int no)
   {
      num = no;
   }
```

Arrays Of References To Objects: An Example (3)

```
    public void setNum (int n)
    {
       num = n;
    }
    public int getNum ()
    {
       return num;
    }
}
```


You Should Now Know

- The difference between the Object-Oriented and the Procedural approaches to software design
- How to use classes and objects in a Java program
  - Defining new classes
  - Creating references to new instances of a class
  - Using the attributes and methods of a object
- What is information hiding and what are the benefits of this approach in the design a classes
- How to write a Java program with multiple classes (driver and with an additional class)
- How to write and overload constructors
- How to declare and manipulate arrays