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?

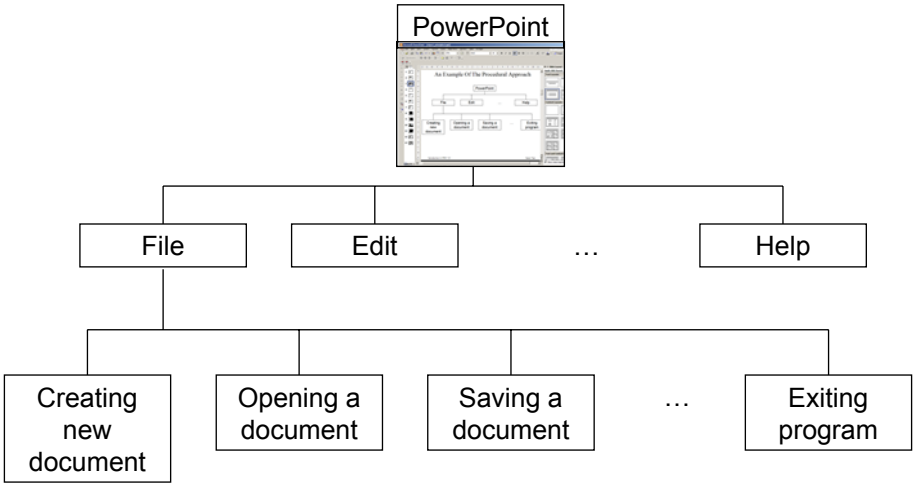
Procedural approach (CPSC 231)

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

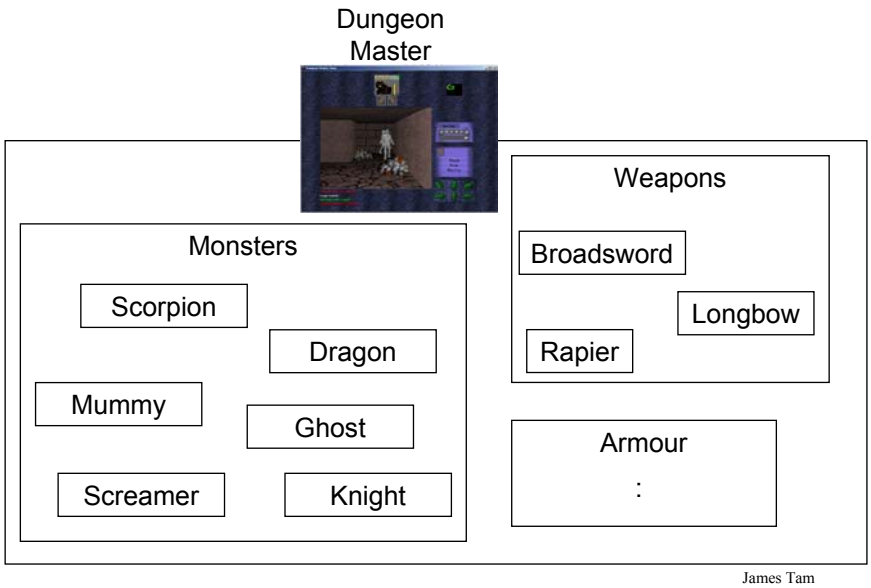
Object-Oriented approach (CPSC 233)

- 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 information can be used to describe the dragon?

What can the dragon do?

Monsters: Attributes

Represents information about the monster:

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

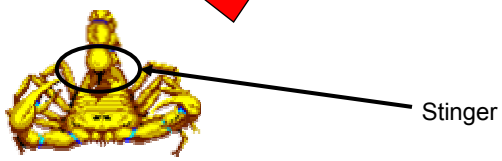
:

Monsters: Operations

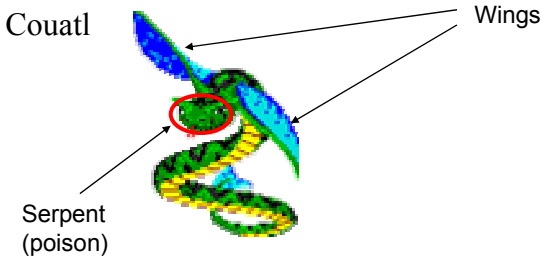
Represents what each monster can do (verb part):

Dragon

Scorpion



Monsters: Operations

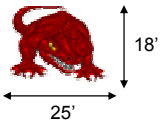


Pascal Records Vs. Java Objects

Composite type (Records)

Information
(attributes)

•Information about the variable.

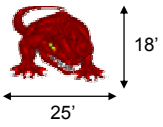


Pascal Records Vs. Java Objects

Composite type (Objects)

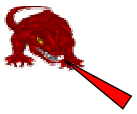
Information (attributes)

- Information about the variable.



Operations (methods[1])

- What the variable "can do"



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

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".

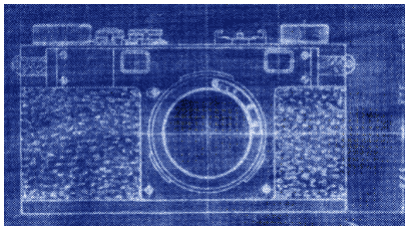
Defining A Java Class (3)

Example:

```
class Foo
{
        private int num;
        public void greet ()
        {
            int value = 1;
            System.out.println(value + " Hello");
        }
        public void setNum (int newValue) { num = newValue; }
        public int getNum () { return num; }
}
```


A Class Is Like A Blueprint

- It indicates the format for what an example of the class should look like (methods and attributes)
- No memory is allocated.



II) Creating/Instantiating 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".

An Instance Is An Actual Example Of A Class

- Instantiating a class is when an actual example/instance of a class is created.

Attributes Vs. Local Variables

Class attributes (variables or constants)

- Declared inside the body of a class definition but outside the body of any class methods.
- Typically there is a separate attribute for each instance of a class and it lasts for the life of the class.

Local variables and constants

- Declared within the body of a class method.
- Last for the life of the method

Examples Of Attributes

```
class Foo
{
      final int NUM1 = 10;
      int num2 = 100;
}
      :
Foo f1 = new Foo ();
Foo f2 = new Foo ();
```

Examples Of Local Variables

```
class Bar
{
  public void method ()
  {
     final int NUM3 = 20;
     int num4 = 200;
  }
}
  :
Bar b1 = new Bar ();
b1.method();
```


Scope Of Local Variables

Enter into scope

- Just after declaration

Exit out of scope

- When the proper enclosing brace is encountered

```
class Bar
{
      public void aMethod ()
      {
         int num1 = 2;
         if (num1 % 2 == 0)
         {
              int num2;
              num2 = 2;
          }
      }
```

Scope of num1

Scope Of Local Variables

Enter into scope
- Just after declaration

Exit out of scope
- When the proper enclosing brace is encountered

```
class Bar
{
    public void aMethod ()
    {
        int num1 = 2;
        if (num1 % 2 == 0)
        {
            int num2;
            num2 = 2;
        }
    }
```

Scope of num2

Scope Of Attributes

```
class Bar
{
    private int num1;
        :       :
    public void methodOne ()
    {


    }
    public void method Two ()
    {


    }
        :       :
    private int num2;
}
```

Scope of num1 & num2

Scope Of Methods

```
class Bar
{
        private int num1;
              :          :
        public void methodOne ()
        {
              num1 = 1;
              num2 = 2;
        }
        public void methodTwo ()
        {


        }
              :          :
        private int num2;
}
```

Scope of methodOne & methodTwo

Scope Of Attributes

```
class Bar
{
        private int num1;
              :          :
        public void methodOne ()
        {
              num1 = 1;
              num2 = 2;
        }
        public void methodTwo ()
        {
              methodOne ();

        }
              :          :
        private int num2;
}
```

III) Using The Parts Of A Class

Format:

<instance name>.<attribute name>;

<instance name>.<method name>(*<p1 name>*, *<p2 name>*…);

Example:

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

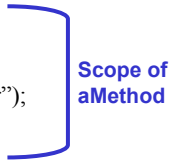
Note: In order to use the dot-operator "." the instance field or method cannot have a private level of access

Referring To Attributes And Methods Outside Of A Class

- Must be qualified by indicating which instance of a class you are referring to.
- This is because you are now outside the attribute's or method's scope.

Referring To Attributes And Methods Outside Of A Class: An Example

```
class Bar
{
    public void aMethod ()
    {
        System.out.println("Calling aMethod of class Bar");
    }
}
```



Referring To Attributes And Methods Outside Of A Class: An Example

```
class Bar
{
    public void aMethod ()
    {
        System.out.println("Calling aMethod of class Bar");
    }
}

class Boo
{
    public void bMethod ()
    {
        Bar b1 = new Bar ();
        Bar b2 = new Bar ();
        b1.aMethod();
    }
}
```

Parameter Passing: Method Definition

Format:

<method name> (*<p1 type>* *<p1 name>*, *<p2 type>* *<p2 name>*…)
{
    // Statements in the body of the method
}

Example:

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


Parameter Passing: Method Call

Format:

<instance name>.*<method name>*(*<p1 name>*, *<p2 name>*…);

Example:

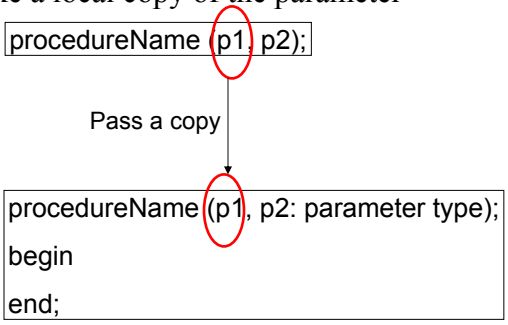
```
f.setNum(10);
```

Methods Of Parameter Passing

- Passing parameters as value parameters (pass by value)
- Passing parameters as variable parameters (pass by reference)

Passing Parameters As Value Parameters

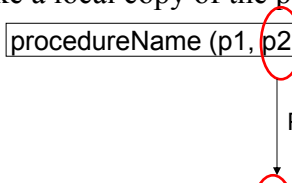
Make a local copy of the parameter



Passing Parameters As Value Parameters

Make a local copy of the parameter

```
procedureName (p1, p2);
```




```
procedureName (p1, p2: parameter type);
begin
end;
```


Passing Parameters As Variable Parameters

Referring to the parameter in the method refers to the original

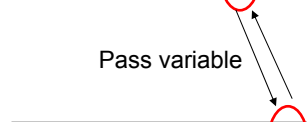
```
procedureName (p1, p2);
```

```
procedureName (var p1, p2: parameter type);
begin
end;
```

Passing Parameters As Variable Parameters

Referring to the parameter in the method refers to the original

```
procedureName (p1, p2);
```

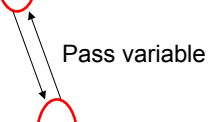



```
procedureName (var p1, p2: parameter type);
begin
end;
```


Passing Parameters As Variable Parameters

Referring to the parameter in the method refers to the original

```
procedureName (p1, p2);
```




```
procedureName (var p1, p2: parameter type);
begin
end;
```

Parameter Passing In Java: Simple Types

All simple types are passed by value in Java.

Type	Description
byte	8 bit signed integer
short	16 but signed integer
int	32 bit signed integer
long	64 bit signed integer
float	32 bit signed real number
double	64 bit signed real number
char	16 bit Unicode character
boolean	1 bit true or false value

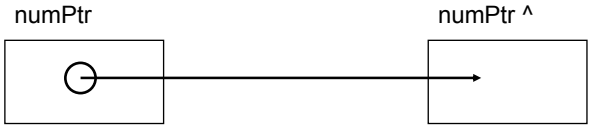
Java References

- It is a pointer that cannot be de-referenced by the programmer
- Automatically garbage collected when no longer needed

De-Referencing Pointers: Pascal Example

```
var
        numPtr : ^ integer;

begin
        new(numPtr);
```



De-Referencing: Java Example

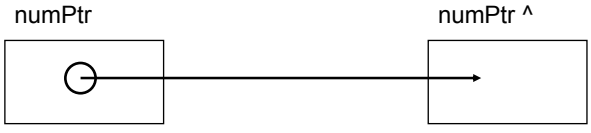
```
Foo f1 = new Foo ();
Foo f2 = new Foo ();

f1.greet();
f1 = f2;
```

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

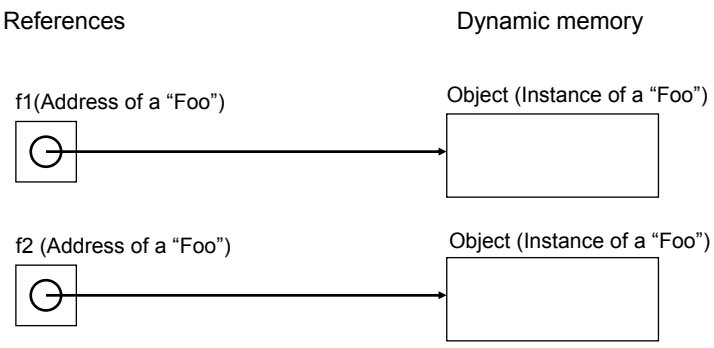
```
dispose(numPtr);
```

Garbage Collection And Pointers: Pascal Example

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

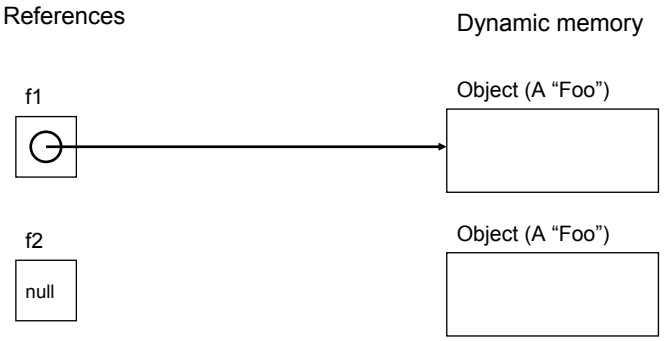
Automatic Garbage Collection Of Java References

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



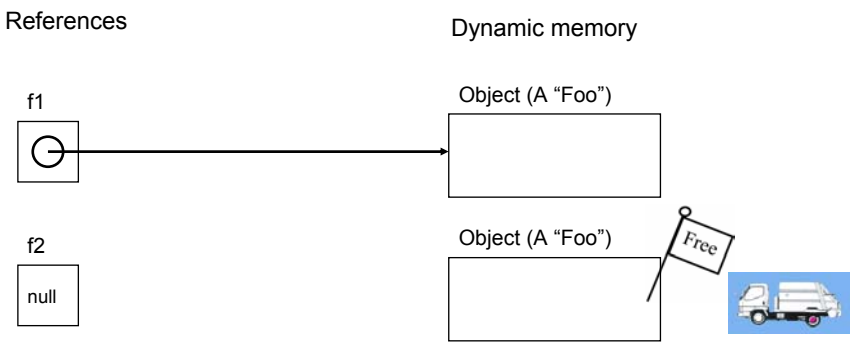
Automatic Garbage Collection Of Java References (2)

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



Automatic Garbage Collection Of Java References (2)

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



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

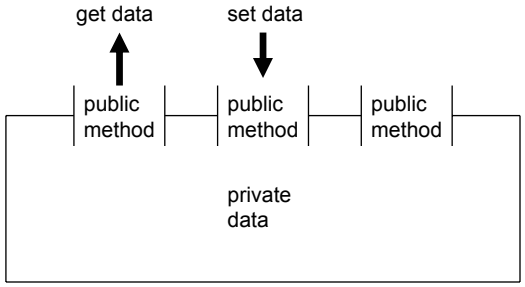
Creating a new monster: “The Critter”
Attribute: Height (must be 60” – 72”)




James Tam

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:

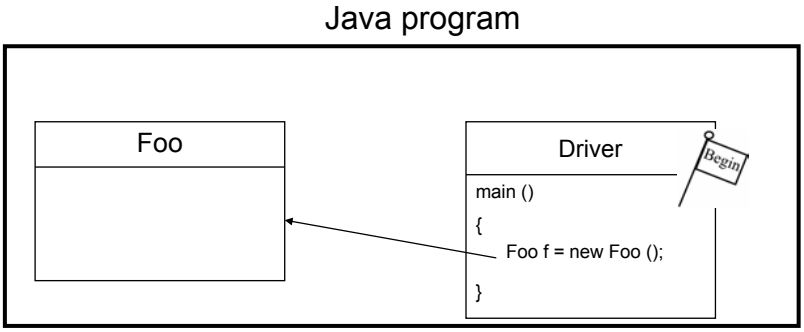
```
public void setNum (int newValue)
{
   num = newValue;
}
```

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



Putting It Altogether: First Object-Oriented Example

Example (The complete example can be found in the directory /home/233/examples/classesObjects1/firstExample

```
class Driver
{
   public static void main (String [] args)
   {
      Foo f = new Foo ();
      f.greet();
      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 = 0;
    public void greet ()
    {
       int value = 1;
       System.out.println(value + " Hello");
    }
    public void setNum (int newValue)
    {
       num = newValue;
    }
    public int getNum ()
    {
       return num;
    }
}
```


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

UML[2] Representation Of A Class

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

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

2 UML = Unified Modeling Language

Class Diagrams With Increased Details

<Name of class>
-<attribute *name*>: <attribute *type*> +<method name> (p1: p1type; p2 : p2 type..): <return type>

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

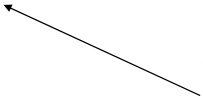
2 UML = Unified Modeling Language

Common Errors When Using References

- Forgetting to initialize the reference
- Using a null reference

Error: Forgetting To Initialize The Reference

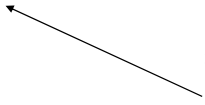
```
Foo f;
f.setNum(10);
```



```
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 (allows for information to be hidden).

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/classesObject1/secondExample

The Driver Class

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

The Driver Class (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();
```


The Driver Class (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.stockLevel);
         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.stockLevel);
```

The Driver Class (4)

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

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

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


The Driver Class (5)

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

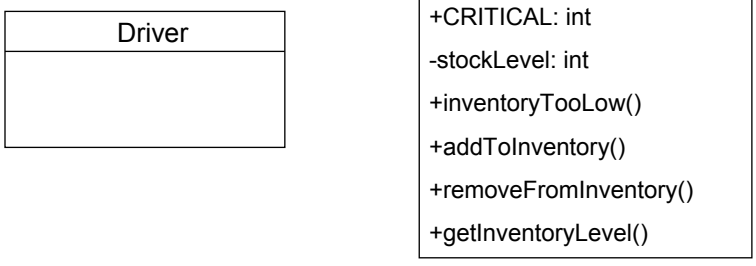
Class Inventory

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

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


Utilizing Information Hiding: An Example

The complete example can be found in the directory
/home/233/examples/classesObjects1/thirdExample



The Driver Class

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


The Driver Class (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();
```

The Driver Class (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;
```


The Driver Class (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;
```

The Driver Class (5)

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


Class Inventory

```
class Inventory
{
   public final int MIN = 0;
   public  final int MAX = 100;
   public final int CRITICAL = 10;
   private int stockLevel = 0;
   public void addToInventory (int amount)
   {
      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)");
      }
```

Class Inventory (2)

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


Class Inventory (3)

```
public void removeFromInventory (int amount)
{
  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;
    }
}
```

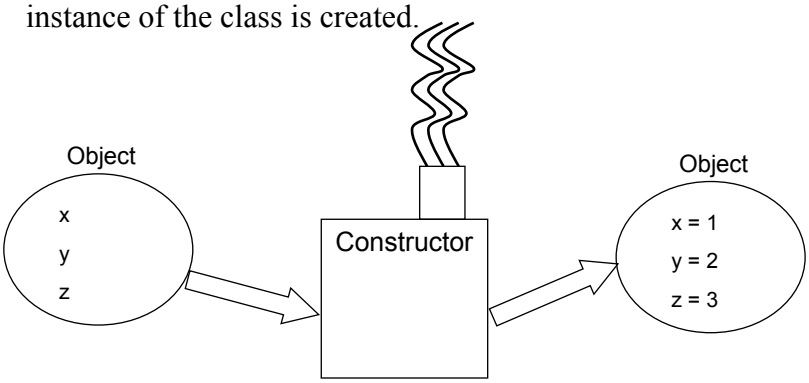
Class Inventory (4)

```
 public boolean inventoryTooLow ()
{
  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 (created).
- The constructor is automatically invoked whenever an instance of the class is created.



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

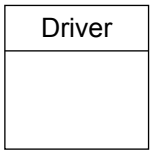
```
public Sheep ()
public Sheep (String n)
```

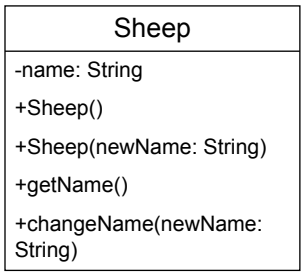
Things to avoid when overloading constructors

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/classesObjects1/fourthExample





The Driver Class

```
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();
```


The Driver Class (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();
   }
}
```

Class Sheep

```
class Sheep
{
   private String name;

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


Class Sheep (2)

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

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

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 newName)
    {
        String name;
        System.out.println("Creating the sheep called " + newName);
        name = newName;
    }
```


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

Declaring Arrays

Arrays in Java involve a reference to the array so creating 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)

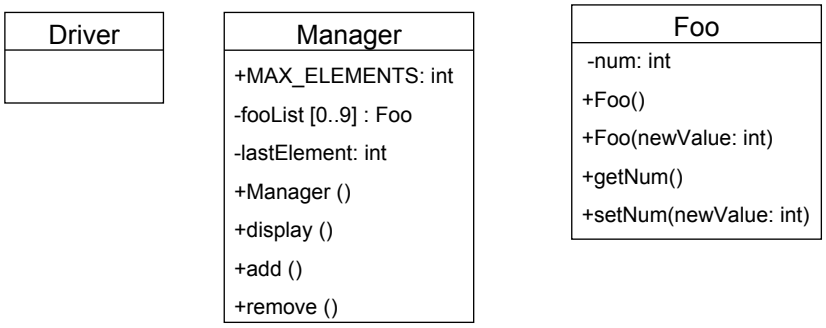
- An array of objects is actually an array of references to objects

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

- The elements are initialized to null by default

Arrays Of References To Objects: An Example

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



The Driver Class

```
import tio.*;
class Driver
{
   public static void main (String [] args)
   {
      Manager fooManager = new Manager ();
      char menuSelection;

      do
      {
         System.out.println("\n\nLIST MANAGEMENT PROGRAM:
         OPTIONS");
         System.out.println("\t(d)isplay list");
         System.out.println("\t(a)dd new element to end of list");
         System.out.println("\t(r)emove last element from the list");
         System.out.println("\t(q)uit program");
         System.out.print("Selection: ");
         menuSelection = (char) Console.in.readChar();
         Console.in.readLine();
```


The Driver Class (2)

```
      switch (menuSelection)
      {
         case 'd':
            fooManager.display();
            break;

         case 'a':
            fooManager.add();
            break;

         case 'r':
            fooManager.remove();
            break;
```

The Driver Class (3)

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

             default:
                System.out.println("Please enter one of 'd','a','r' or 'q'");
          } // End of switch.
       } while (menuSelection != 'q');
    }  // End of main.
}  // End of class Driver.
```


The Manager Class

```
import tio.*;
class Manager
{
   public final int MAX_ELEMENTS = 10;
   private Foo [] fooList;
   private int lastElement;

   public Manager ()
   {
      fooList = new Foo[MAX_ELEMENTS];
      int i;
      for (i = 0; i < MAX_ELEMENTS; i++)
      {
         fooList[i] = null;
      }
      lastElement = -1;
   }
```

The Manager Class (2)

```
public void display()
{
   int i;
   System.out.println("Displaying list");
   if (lastElement == -1)
      System.out.println("\tList is empty");
   for (i = 0; i <= lastElement; i++)
   {
      System.out.println("\tElement [" + i + "]=" + fooList[i].getNum());
   }
}
```


The Manager Class (3)

```
public void add ()
{
   int newValue;
   System.out.print("Enter an integer value for new element: ");
   newValue = Console.in.readInt();
   Console.in.readLine();
   if ((lastElement+1) < MAX_ELEMENTS)
   {
      lastElement++;
      fooList[lastElement] = new Foo(newValue);
      System.out.println("New element with a value of " + newValue + " added"
      added");
   }
   else
   {
      System.out.print("Cannot add new element: ");
      System.out.println("List already has " + MAX_ELEMENTS + "
      elements.");
   }
```

The Manager Class (4)

```
public void remove ()
{
   if (lastElement != -1)
   {
      fooList[lastElement] = null;
      lastElement--;
      System.out.println("Last element removed from list.");
   }
   else
   {
      System.out.println("List is already empty: Nothing to remove");
   }
}
```


Class Foo

```
class Foo
{
   private int num;
   public Foo () { num = 0; }
   public Foo (int newValue) { num = newValue; }
   public void setNum (int newValue) { num = newValue; }
   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