CPSC 233: Introduction to Classes and Objects

Attributes and methods

Creating new classes

References: Dynamic memory allocation and automatic garbage collection

Information hiding and encapsulation

Constructors

Shadowing

Arrays

James Tam

What Does Object-Oriented Mean?

Procedural approach (CPSC 231)

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

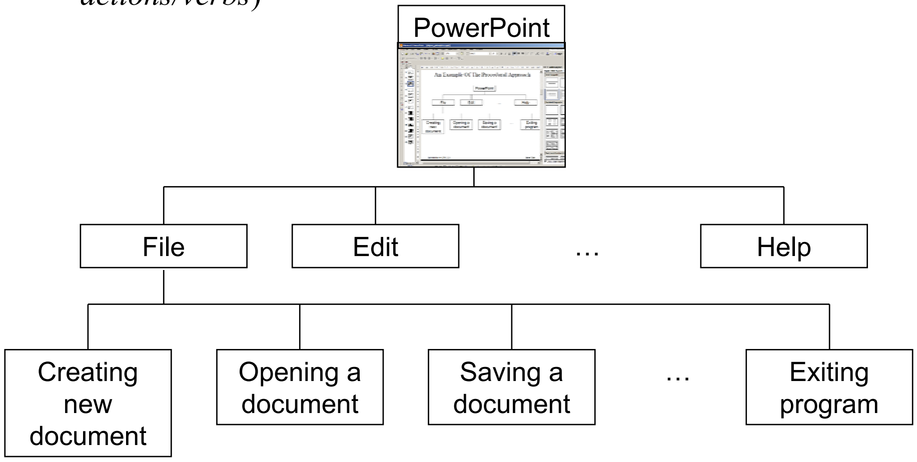
Object-Oriented approach (CPSC 233)

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

James Tam

An Example Of The Procedural Approach

- Break down the program by what it does (described with *actions/verbs*)



An Example Of The Object-Oriented Approach

- Break down the program into 'physical' components (*nouns*)

Dungeon Master

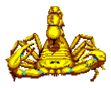
Monsters	
•Scorpion	•Mummy
•Ghost	•Screamer
•Knight	•Dragon

Weapons	
•Broadsword	•Rapier
•Longbow	•Mace

Example Objects: Monsters From Dungeon Master

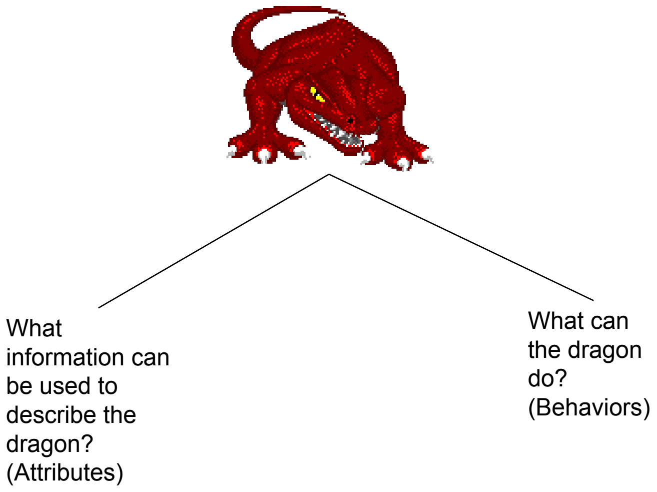
Dragon

Scorpion



Couatl

Ways Of Describing A Monster


What information can be used to describe the dragon? (Attributes)
What can the dragon do? (Behaviors)

Monsters: Attributes

Represents information about the monster:

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

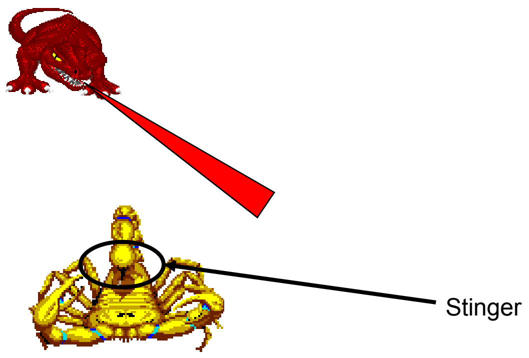
:

Monsters: Behaviours

Represents what each monster can do (verb part):

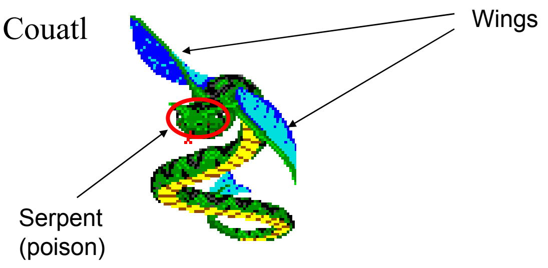
Dragon

Scorpion



Monsters: Operations

Couatl

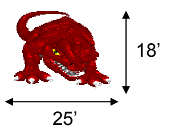


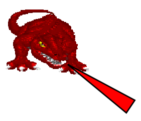
Pascal Records Vs. Java Objects

Composite type (Records)

Information (attributes)

- Information about the variable.



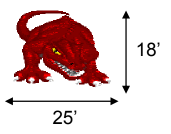


Pascal Records Vs. Java Objects

Composite type (Objects)

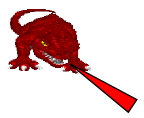
Information (attributes)

- 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

One Benefit Of Bundling Behaviors With Objects

It can be more logical to bundle into the definition of composite type what each instance can do rather than implementing that code elsewhere.

Non-Object-Oriented Approach

```
Type
   Dragon = record
            :
   end;
            :        :
procedure fly ();
begin
            :
end;
```

Object-Oriented Approach

```
public class Dragon
{
    private int height;
    private int weight;
    public void fly ()
    {
            :
    }
}
```

Working With Objects In Java

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

I) Defining A Java Class

Format of class definition:

```
public 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 type2> <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".

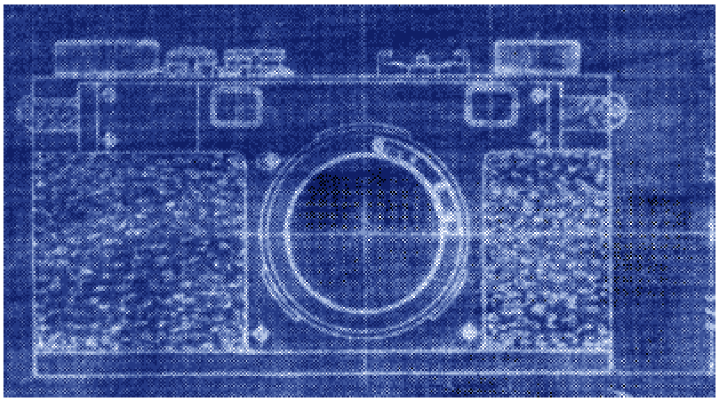
Defining A Java Class (3)

Example:

```
public class Person
{
    private int age;
    public void setAge (int newAge)
    {
        age = newAge;
    }
    public int getAge ()
    {
        return age;
    }
}
```

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:

Person jim = new Person();

Note: 'jim' is not an object of type 'Person' but a reference to an object of type 'Person'.

An Instance Is An Actual Example Of A Class

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

Declaring A Reference Vs. Instantiating An Instance

Declaring a reference to a ‘Person’

• Person jim;

Instantiating/creating an instance of a ‘Person’

• jim = new Person ();

III) Using The Parts Of A Class

Format:

<instance name>.<attribute name>;

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

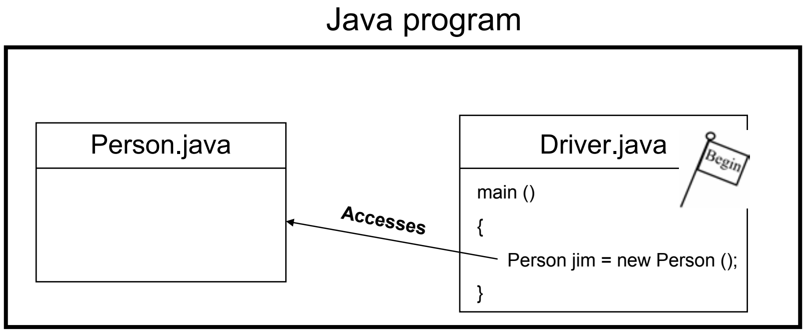
Example:

```
int newAge = 27;
Person jim = new Person ();
jim.setAge(newAge);
System.out.println(jim.getAge());
```

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

Laying Out Your Program

- The program must contain a 'Driver' class.
- The driver class is the place where the program starts running (**it contains the main method**).
- Instances of other classes can be created and used here.
- For now you should have all the classes for a particular program reside in the same directory or folder.



Putting It Altogether: First Object-Oriented Example

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

```
public class Driver
{
   public static void main (String [] args)
   {
      int newAge = 27;
      Person jim = new Person ();
      jim.setAge(newAge);
      System.out.println("Jim's current age is..." + jim.getAge());
   }
}
```

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

```
public class Person
{
   private int age;
   public void setAge (int newAge)
   {
       age = newAge;
   }
   public int getAge ()
   {
       return age;
   }
}
```

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

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 An Attribute

```
public class Person
{
   private int age;
   public void setAge (int newAge)
   {
       age = newAge;
   }
       :
}
      :
  main (String [] args)
  {
     Person jim = new Person ();
     Person joe = new Person ();
  }
```

Declared within the definition of the class

James Tam

Examples Of An Attribute

```
public class Person
{
   private int age;
   public void setAge (int newAge)
   {
       age = newAge;
   }
       :
}
      :
  main (String [] args)
  {
     Person jim = new Person ();
     Person joe = new Person ();
  }
```

But declared outside of the body of a method

James Tam

Example Of A Local Variable

```
public class Person
{
   private int age;
   public void aMethod ()
   {
        int num;                    Declared inside of
        num = 1;                    the body of a
    }                               method
       :
}
       :
   main (String [] args)
   {
      Person jim = new Person ();
      Person joe = new Person ();
      jim.aMethod();
      joe.aMethod();
   }
```

James Tam

Scope Of Local Variables

Enter into scope
- Just after declaration

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

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

James Tam

Scope Of Local Variables

Enter into scope

- Just after declaration

Exit out of scope

- When the proper enclosing brace is encountered

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

Scope of num2

Scope Of Attributes

```
public class Bar
{
       private int num1;
              :              :
       public void methodOne ()
       {
              num1 = 1;
              num2 = 2;
       }
       public void methodTwo ()
       {
              num1 = 10;
              num2 = 20;
       }
              :              :
       private int num2;
}
```

Scope of num1 & num2

Scope Of Attributes

```
public class Bar
{
        private int num1;
                :              :
        public void methodOne ()
        {
                num1 = 1;
                num2 = 2;
        }
        public void methodTwo ()
        {
                num1 = 10;
                num2 = 20;
        }
                :              :
        private int num2;
}
```

Scope of methodOne and methodTwo

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

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

Scope of aMethod

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

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

public class Driver
{
        public static void main (String [] args)
        {
               Bar b1 = new Bar ();
               Bar b2 = new Bar ();
               b1.aMethod();
        }
}
```

Outside the scope

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

```
public class Foo
{
        private int num;
        public Foo () { num = 0; }
        public void methodOne () { methodTwo(); }
        public void methodTwo () { .. }
               :          :          :
}
  :   :
  main ()
  {
        Foo f1 = new Foo ();
        Foo f2 = new Foo ();
        f1.methodOne();
   }
```

Call is inside the scope (no instance name or 'dot' needed

Call is outside the scope (instance name and 'dot' IS needed

Referring To The Attributes And Methods Of A Class: Recap

1. Outside of the methods of the class you must use the dot-operator as well as indicating what instance that you are referring to.

   e.g., f1.method();

2. Inside the methods of the class there is no need to use the dot-operator nor is there a need for an instance name.

   e.g.,

```
public class Foo
{
    public void m1 () { m2(); }
    public void m2 () { .. }
}
```

Parameter Passing: Method Definition

Format:

public *<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:

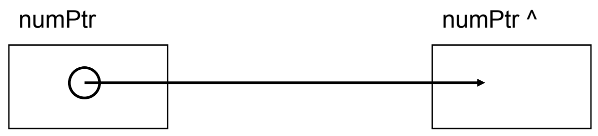
```
f.setNum(10);
```

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

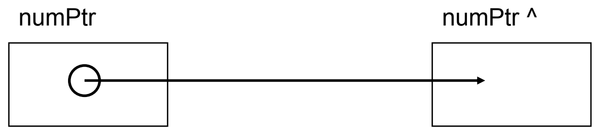
```
Foo f1 = new Foo ();
Foo f2 = new Foo ();

f1 = f2;
```

Exactly what is being copied here?

Java References

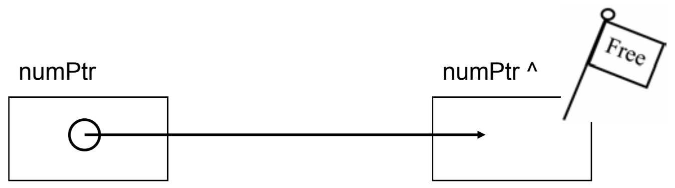
- It is a pointer that cannot be de-referenced by the programmer
- Automatically garbage collected when no longer needed

Garbage Collection And Pointers: Pascal Example



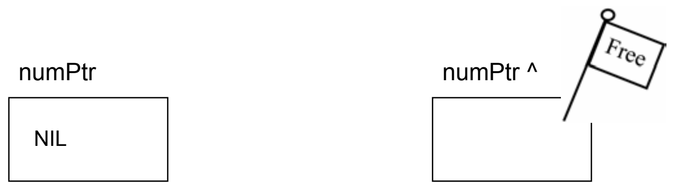
Garbage Collection And Pointers: Pascal Example

```
dispose(numPtr);
```



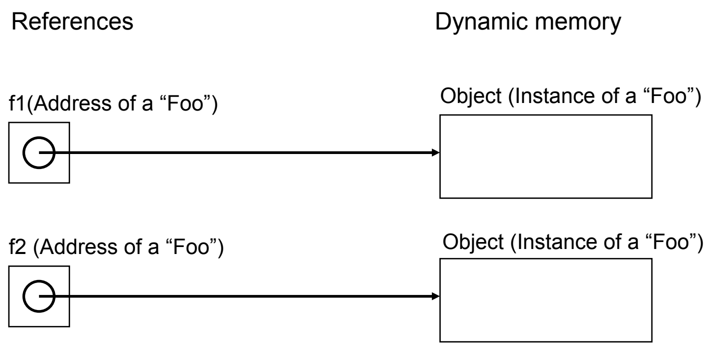
Garbage Collection And Pointers: Pascal Example

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



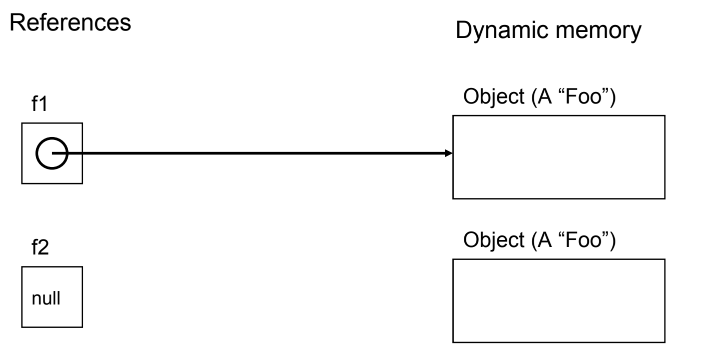
Automatic Garbage Collection Of Java References

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



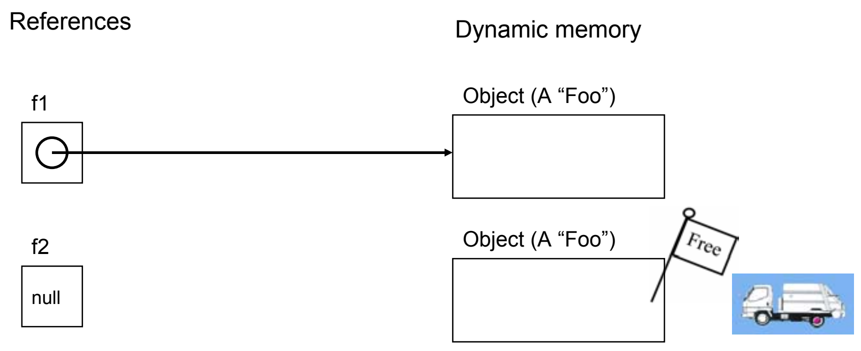
Automatic Garbage Collection Of Java References (2)

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;



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
```

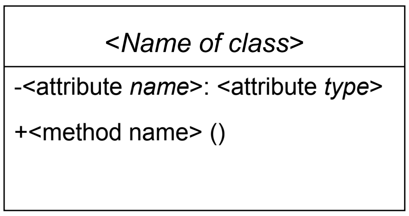
Error: Using Null References

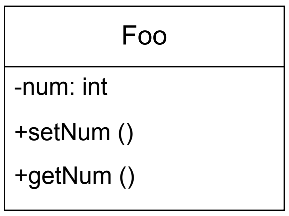
```
Foo f = null;
f.setNum(10);
```

Run-time error!

```
> java Driver
Exception in thread "main"
java.lang.NullPointerException
        at Driver.main(Driver.java:14)
```

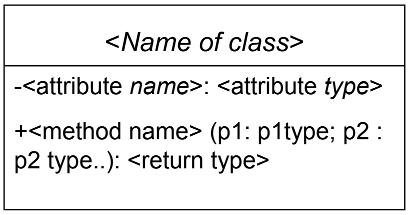
UML[2] Representation Of A Class



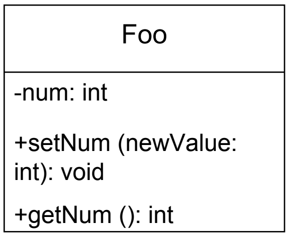


2 UML = Unified Modeling Language

Class Diagrams With Increased Details



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



2 UML = Unified Modeling Language

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)

Encapsulation

Grouping data and methods together within a class definition to allow the private attributes to be accessible only through the public methods (this allows for information to be hidden and protected because access to the data occurs only through a controlled scenario).

Illustrating The Need For Information Hiding: An Example

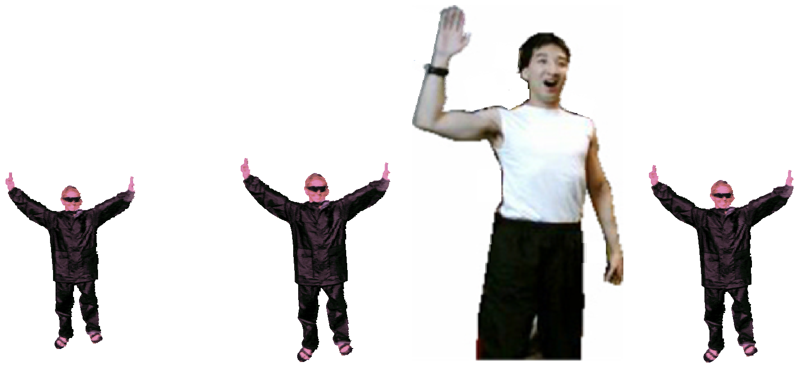
Creating a new monster: “The Critter”

Attribute: Height (must be 60” – 72”)

Illustrating The Need For Information Hiding: An Example

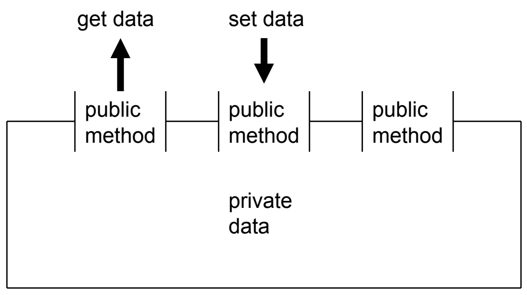
Creating a new monster: “The Critter”

Attribute: Height (must be 60” – 72”)



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 - a ‘get’ method
   - Used to determine the current value of a field
   - Example:

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

2) Mutator methods - a ‘set’ method
   - Used to set a field to a new value
   - Example:

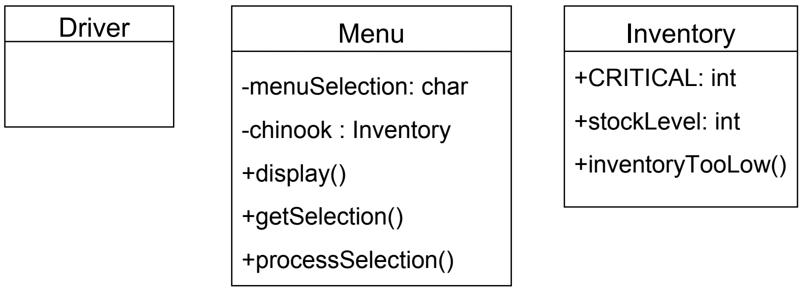
```
public void setNum (int newValue)
{
    num = newValue;
}
```

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/introductionOO/secondExample



The Driver Class

```
public class Driver
{
   public static void main (String [] args)
   {
        Menu aMenu = new Menu ();
        aMenu.processSelection();
   }
}
```

The Menu Class

```
public class Menu
{
  private char menuSelection;
  private Inventory chinook = new Inventory();
  public void display ()
  {
    System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
     System.out.println("\t(a)dd new stock to inventory");
     System.out.println("\t(c)heck if stock level is critically low");
     System.out.println("\t(d)isplay stock level");
     System.out.println("\t(q)uit program");
     System.out.println("\t(r)emove stock from inventory");
     System.out.print("Selection: ");
  }
  public void getSelection ()
  {
    menuSelection = (char) Console.in.readChar();
    Console.in.readLine();
  }
```

The Menu Class (2)

```
public void processSelection ()
{
   int amount;
  do
  {
       display();
       getSelection ();
       switch (menuSelection)
       {
         case 'A':
         case 'a':
           System.out.print("No. items to add: ");
           amount = Console.in.readInt();
           Console.in.readLine();
           chinook.stockLevel = chinook.stockLevel + amount;
           System.out.println("Inventory: " + chinook.stockLevel);
           break;
```

The Menu Class (3)

```
case 'C':
case 'c':
   if (chinook.inventoryTooLow())
     System.out.println("Stock levels critical!");
   else
     System.out.println("Stock levels okay");
 System.out.println("Inventory: " + chinook.stockLevel);
 break;

case 'D':
case 'd':
   System.out.println("Inventory: " + chinook.stockLevel);
   break;
```

The Menu Class (4)

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

case 'R':
case 'r':
    System.out.print("No. items to remove: ");
    amount = Console.in.readInt();
    Console.in.readLine();
    chinook.stockLevel = chinook.stockLevel - amount;
    System.out.println("Inventory: " + chinook.stockLevel);
    break;
```

The Menu Class (5)

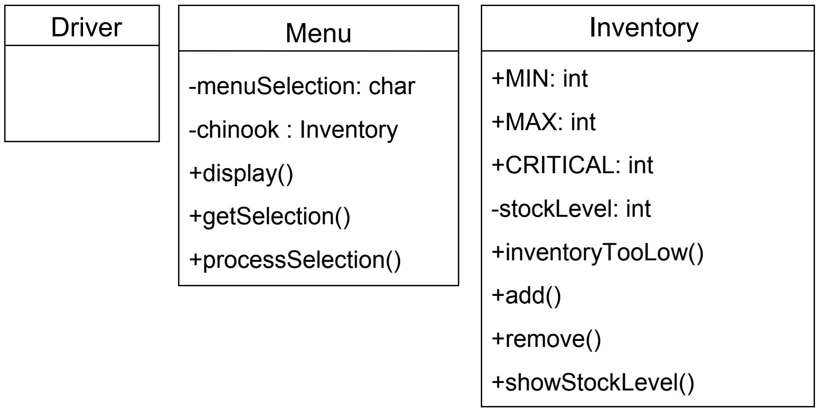
```
        default:
        System.out.println("Enter one of 'a', 'c', 'd', 'q' or 'r'");
       }
     } while ((menuSelection != 'Q') &&
             (menuSelection != 'q'));
  }
}
```

Class Inventory

```
public class Inventory
{
   public final int CRITICAL = 10;
   public int stockLevel;
   stockLevel = 0;
    }

   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/introductionOO/thirdExample



The Driver Class

```
public class Driver
{
   public static void main (String [] args)
   {
        Menu aMenu = new Menu ();
        aMenu.processSelection();
   }
}
```

Class Menu

```
public class Menu
{
  private char menuSelection;
  private Inventory chinook = new Inventory();
  public void display ()
  {
    System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
    System.out.println("\t(a)dd new stock to inventory");
    System.out.println("\t(c)heck if stock level is critically low");
    System.out.println("\t(d)isplay stock level");
    System.out.println("\t(q)uit program");
    System.out.println("\t(r)emove stock from inventory");
    System.out.print("Selection: ");
  }
  public void getSelection ()
  {
    menuSelection = (char) Console.in.readChar();
    Console.in.readLine();
  }
```

Class Menu (2)

```
  public void processSelection ()
  {
    int amount;
    do
    {
       display();
       getSelection ();
       switch (menuSelection)
       {
         case 'A':
         case 'a':
           System.out.print("No. items to add: ");
           amount = Console.in.readInt();
           Console.in.readLine();
           chinook.add(amount);
           System.out.println(chinook.showStockLevel());
           break;
```

Class Menu (3)

```
 case 'C':
case 'c':
   if (chinook.inventoryTooLow())
      System.out.println("Stock levels critical!");
   else
      System.out.println("Stock levels okay");
      System.out.println(chinook.showStockLevel());
   break;

 case 'D':
case 'd':
    System.out.println(chinook.showStockLevel());
    break;

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

James Tam

Class Menu (4)

```
        case 'R':
       case 'r':
           System.out.print("No. items to remove: ");
           amount = Console.in.readInt();
            Console.in.readLine();
            chinook.remove(amount);
            System.out.println(chinook.showStockLevel());
            break;

        default:
           System.out.println("Enter one of 'a', 'c', 'd', 'q' or 'r'");
      }
   } while ((menuSelection != 'Q') && (menuSelection != 'q'));
  }
}
```

James Tam

Class Inventory

```
public class Inventory
{
   public final int CRITICAL = 10;
   public final int MIN = 0;
   public final int MAX = 100;
   private int stockLevel = 0;
   public boolean inventoryTooLow ()
   {
      if (stockLevel < CRITICAL)
        return true;
      else
        return false;
   }
```

Class Inventory (2)

```
   public void add (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)");
      }
      else
      {
         stockLevel = temp;
      }
   } // End of method add
```

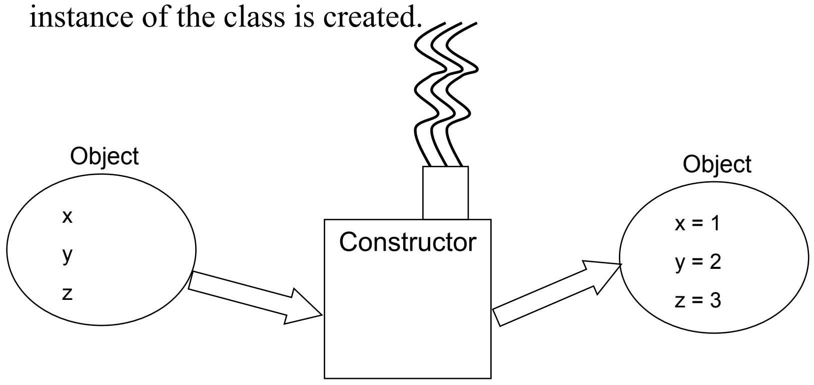
Class Inventory (3)

```
public void remove (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;
    }
}
public String showStockLevel ()
{
    return("Inventory: " + stockLevel);
}
}
```

James Tam

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.



James Tam

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

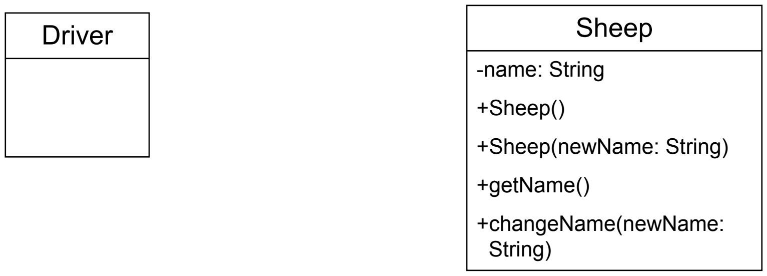
```
public Sheep ()
public Sheep (String n)
```

Things to avoid when overloading constructors

1) Distinguishing constructors solely by the order of the parameters
2) Overloading constructors but having identical implementation for each body

Constructors: An Example

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



The Driver Class

```
public class Driver
{
   public static void main (String [] args)
   {
       Sheep nellie;
       Sheep jim;
       System.out.println("Creating flock...");
       nellie = new Sheep ("Nellie");
       jim = new Sheep();
       jim.setName("Jim");
       System.out.println("Displaying updated flock");
       System.out.println("   "+ nellie.getName());
       System.out.println("   "+ jim.getName());
   }
}
```

Class Sheep

```
public 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;
   }
    public String getName () { return name; }
   public void setName (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 careful of accidentally doing this.

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

Shadowing

Scope Rules:

1. Look for a local identifier
2. Look for an attribute

```
public class Foo
{
    // Attributes
    public void method ()
    {
        // Local variables

        num = 1;
    }
}
```

Second: Look for an attribute by that name

First: Look for a local identifier by that name

A reference to an identifier

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)
- Many utility methods exist
- 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)**
- Many utility methods exist
- 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)
- Many utility methods exist
- 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)

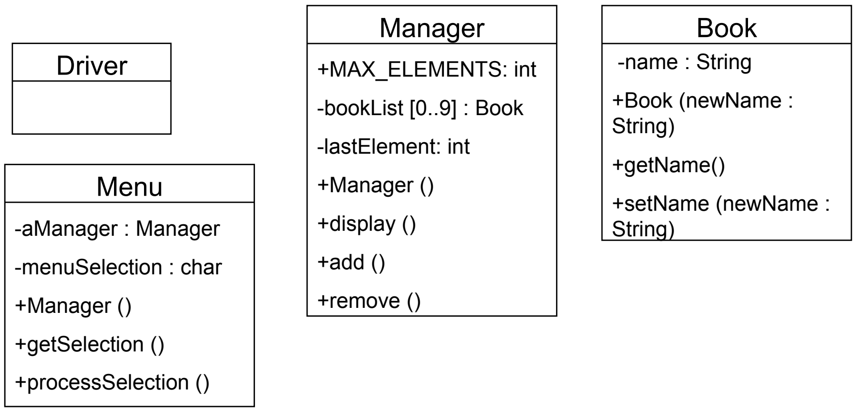
- An array of objects is actually an array of references to objects
  - e.g., Foo [] arr = new Foo [4];

- 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/introductionOO/fifthExample



The Driver Class

```
public class Driver
{
   public static void main (String [] args)
   {
       Menu aMenu = new Menu ();
       aMenu.processSelection();
   }
}
```

The Menu Class

```
public class Menu
{
   private Manager aManager;
   private char menuSelection;

   public Menu () { aManager = new Manager (); }

   public void display ()
   {
        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: ");
   }
```

The Menu Class (2)

```
    public void getSelection ()
   {
     menuSelection = (char) Console.in.readChar();
     Console.in.readLine();
   }

    public void processSelection ()
   {
     do
     {
         display();
         getSelection();
```

The Menu Class (3)

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

   case 'A':
   case 'a':
      aManager.add();
      break;

   case 'R':
   case 'r':
      aManager.remove();
      break;
```

The Menu Class (4)

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

         default:
           System.out.println("Please enter one of 'd','a','r' or 'q'");
        }
    } while ((menuSelection != 'Q') && (menuSelection != 'q'));
  }
}
```

The Manager Class

```
public class Manager
{
    public final int MAX_ELEMENTS = 10;
    private Book [] bookList;
    private int lastElement;

    public Manager ()
    {
        bookList = new Book[MAX_ELEMENTS];
        int i;
        for (i = 0; i < MAX_ELEMENTS; i++)
        {
            bookList[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("\tTitle No. " + (i+1) + ": "+ bookList[i].getName());
        }
    }
```

The Manager Class (3)

```
public void add ()
{
   String newName;
   System.out.print("Enter a title for the book: ");
   newName = Console.in.readLine();
   if ((lastElement+1) < MAX_ELEMENTS)
   {
      lastElement++;
      bookList[lastElement] = new Book(newName);
      System.out.println("Book " + newName + " added");
   }
```

The Manager Class (4)

```
      else
      {
         System.out.print("Cannot add new element: ");
         System.out.println("List already has " +
MAX_ELEMENTS + "
         elements.");
      }
 }
```

The Manager Class (5)

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

The Book Class

```
public class Book
{
   private String name;
   public Book (String newName) { name = newName; }
   public void setName (String newName) { name = newName; }
   public String getName () { return name; }
}
```

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 class
- 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