

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)

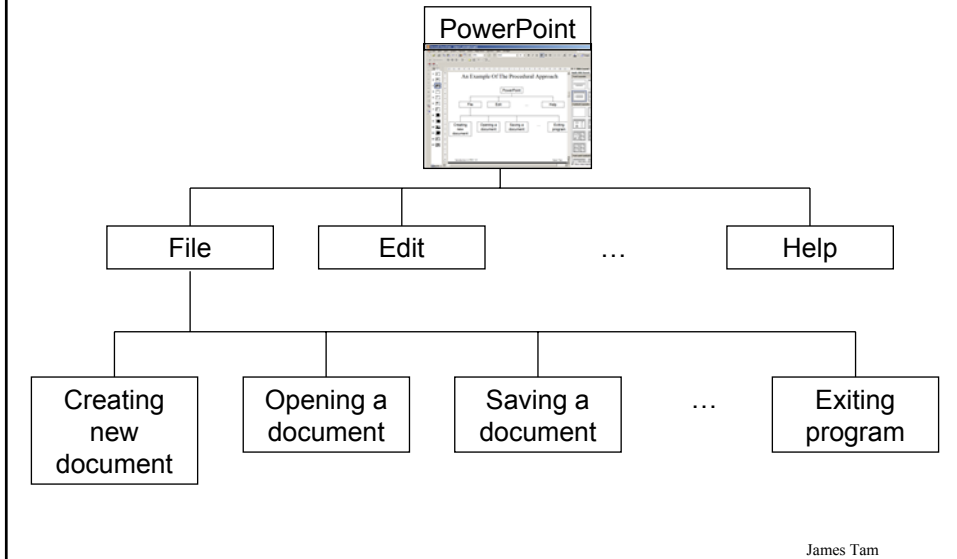
- Design and build the software in terms of actions (verbs)

Object-Oriented approach (CPSC 233)

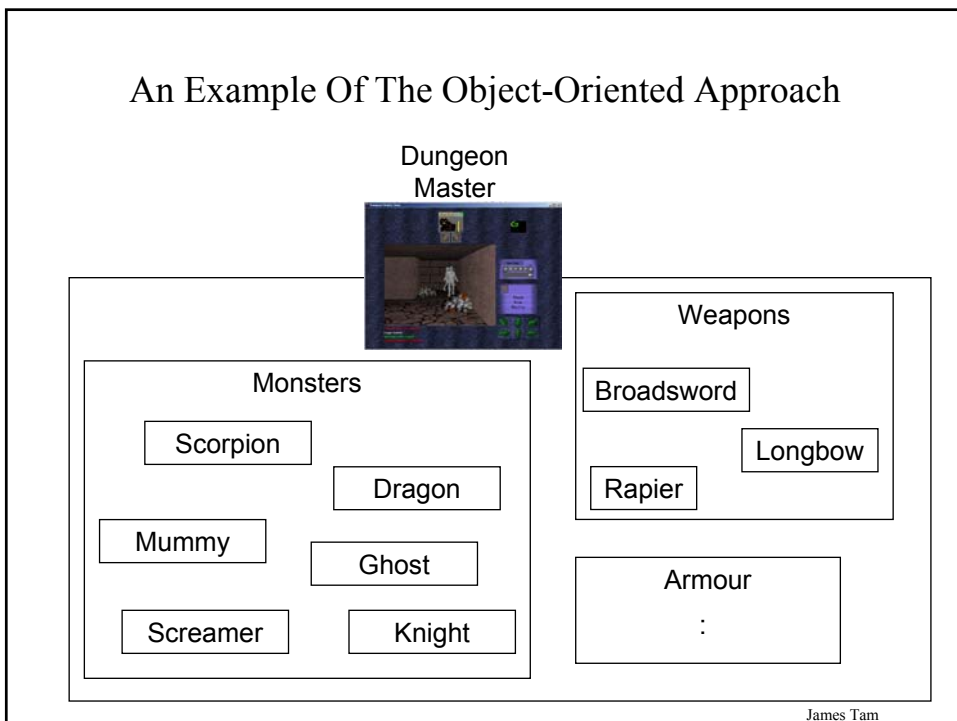
- Design and build the software in terms of things (nouns)

James Tam

An Example Of The Procedural Approach



An Example Of The Object-Oriented Approach



Example Objects: Monsters From Dungeon Master

Dragon



Scorpion



Couatl



James Tam

Ways Of Describing A Monster



What information
can be used to
describe the
dragon?

What can the
dragon do?

James Tam

Monsters: Attributes

Represents information about the monster:

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

:

James Tam

Monsters: Operations

Represents what each monster can do (verb part):

Dragon



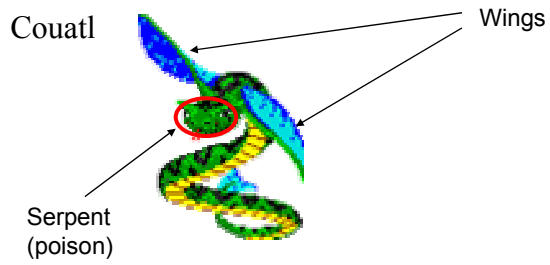
Scorpion



Stinger

James Tam

Monsters: Operations



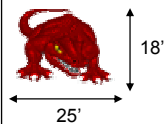
James Tam

Pascal Records Vs. Java Objects

Composite type (Records)

Information
(attributes)

• Information about the
variable.



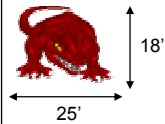
James Tam

Pascal Records Vs. Java Objects

Composite type (Objects)

Information (attributes)

- Information about the variable.



Operations (methods¹)

- What the variable "can do"



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

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

James Tam

I) Defining A Java Class

Format of class definition:

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

James Tam

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".

James Tam

Defining A Java Class (3)

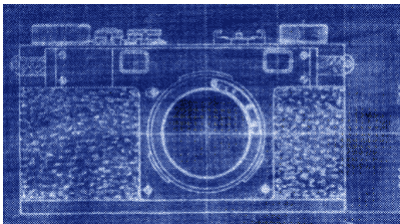
Example:

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

James Tam

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.



James Tam

II) Creating/Instantiating Instances Of A Class

Format:

```
public <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”.

James Tam

An Instance Is An Actual Example Of A Class

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



James Tam

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();
```

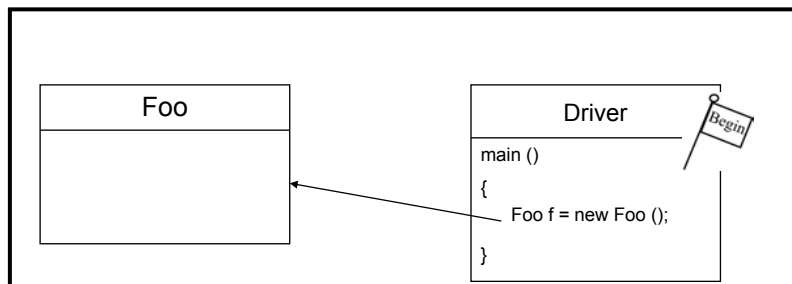
Note: In order to use the dot-operator "." the instance field or method cannot have a private level of access

James Tam

Laying Out Your Program

- The program must contain a “driver” class
- The driver class is the place where the program starts running (contains the main method)
- Instances of other classes can be created here

Java program



James Tam

Putting It Altogether: First Object-Oriented Example

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

```
class Driver
{
    public static void main (String [] args)
    {
        Foo f = new Foo ();
        f.greet();
    }
}
```

James Tam

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

```
public class Foo
{
    public Foo ()
    {
        System.out.println("Creating an actual instance of class Foo.");
    }

    public void greet ()
    {
        System.out.println("Hello!");
    }
}
```

James Tam

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 Driver¹
- For now avoid:
 - Defining instance fields / attributes for the Driver¹
 - 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)

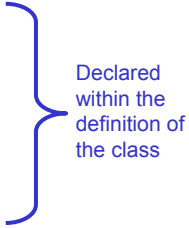
- Declared inside the body of a class definition but outside the body of any class methods.
- Typically there is a separate attribute for each instance of a class and it lasts for the life of the class.

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 Foo
{
    private int num = 0;
    public int getNum ()
    {
        return num;
    }
    : : :
}
:
Foo f1 = new Foo ();
Foo f2 = new Foo ();
```

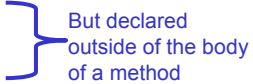


Declared within the definition of the class

James Tam

Examples Of An Attribute

```
public class Foo
{
    private int num = 0;
    public int getNum ()
    {
        return num;
    }
    : : :
}
:
Foo f1 = new Foo ();
Foo f2 = new Foo ();
```



But declared outside of the body of a method

James Tam

Examples Of A Local Variable

```
public class Foo
{
    public void greet ()
    {
        int value = 1;
        System.out.println(value + " Hello");
    }
    :
    :
    :
}
:
:
Foo f1 = new Foo ();
f1.greet();
Foo f2 = new Foo ();
f2.greet();
f2.greet()
```

Declared inside of
the body of a
method

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 num1

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

James Tam

Scope Of Attributes

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

Scope of num1 & num2

James Tam

Scope Of Methods

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

    }

    :
    :
    private int num2;
}
```

Scope of methodOne & methodTwo

James Tam

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

James Tam

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




Outside the scope

James Tam

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 () { .. }
    :      :      :
}

: :
Foo f1 = new Foo ();
f.methodOne();
```



Inside the scope

James Tam

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.

e.g.,

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

James Tam

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;
}
```

James Tam

Parameter Passing: Method Call

Format:

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

Example:

f.setNum(10);

James Tam

Methods Of Parameter Passing

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

James Tam

Passing Parameters As Value Parameters

Make a local copy of the parameter

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

Pass a copy

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

James Tam

Passing Parameters As Value Parameters

Make a local copy of the parameter

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

Pass a copy

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

James Tam

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;
```

James Tam

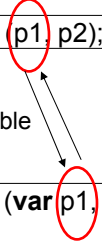
Passing Parameters As Variable Parameters

Referring to the parameter in the method refers to the original

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

Pass variable

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



James Tam

Passing Parameters As Variable Parameters

Referring to the parameter in the method refers to the original

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

Pass variable

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

James Tam

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

James Tam

Java References

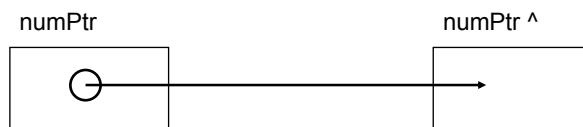
- It is a pointer that cannot be de-referenced by the programmer
- Automatically garbage collected when no longer needed

James Tam

De-Referencing Pointers: Pascal Example

```
var
  numPtr : ^ integer;

begin
  new(numPtr);
```



James Tam

De-Referencing: Java Example

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

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

```
f1.greet();
```

```
f1 = f2;
```

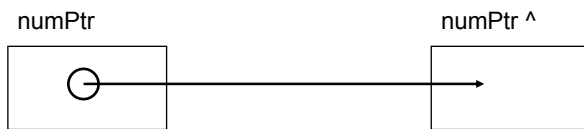
James Tam

Java References

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

James Tam

Garbage Collection And Pointers: Pascal Example



James Tam

Garbage Collection And Pointers: Pascal Example

```
dispose(numPtr);
```



James Tam

Garbage Collection And Pointers: Pascal Example

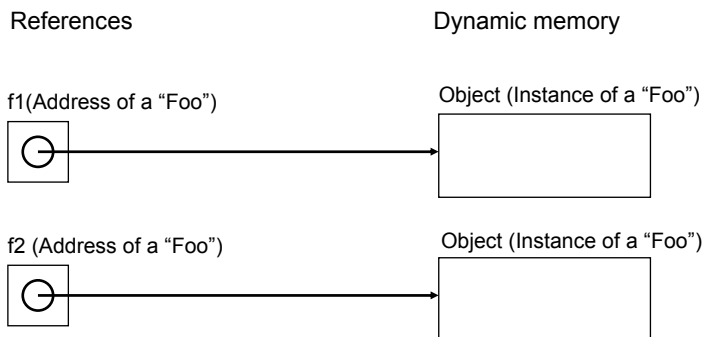
```
dispose(numPtr);  
numPtr := NIL;
```



James Tam

Automatic Garbage Collection Of Java References

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



James Tam

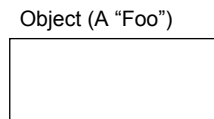
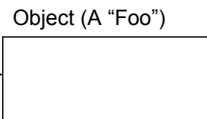
Automatic Garbage Collection Of Java References (2)

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

References



Dynamic memory



James Tam

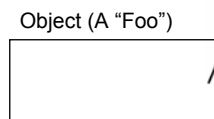
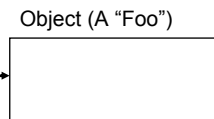
Automatic Garbage Collection Of Java References (2)

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

References



Dynamic memory



James Tam

Common Errors When Using References

- Forgetting to initialize the reference
- Using a null reference

James Tam

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
```

James Tam

Error: Using Null References

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

Run-time error!

> java Driver

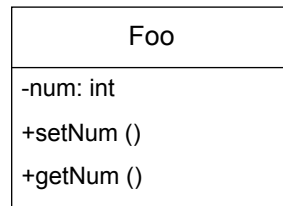
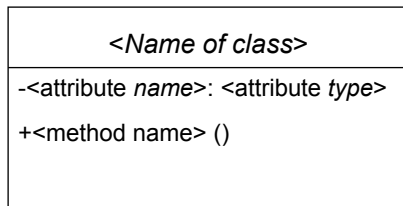
Exception in thread "main"

java.lang.NullPointerException

at Driver.main(Driver.java:14)

James Tam

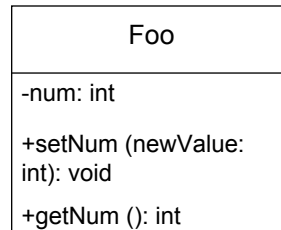
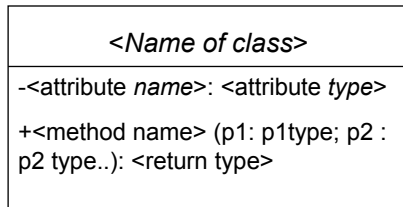
UML² Representation Of A Class



² UML = Unified Modeling Language

James Tam

Class Diagrams With Increased Details



2 UML = Unified Modeling Language

James Tam

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)



James Tam

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).

James Tam

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

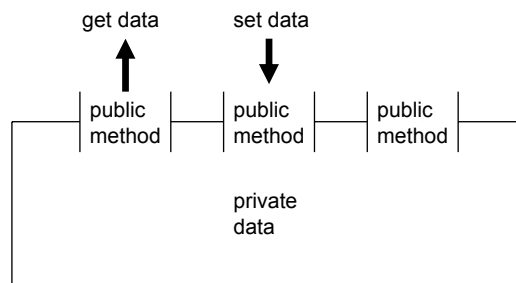
Creating a new monster: “The Critter”
Attribute: Height (must be 60” – 72”)



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



James Tam

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;
}
```

James Tam

Information Hiding: An Example

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

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

James Tam

Information Hiding: An Example (2)

```
public class Foo
{
    private int num;

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

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

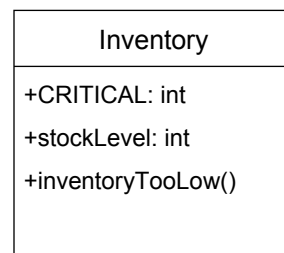
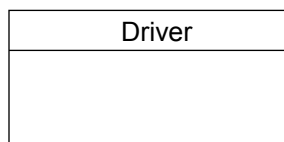
James Tam

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/thirdExample>



James Tam

The Driver Class

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

James Tam

The Driver Class (2)

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

James Tam

The Driver Class (3)

```
switch (menuSelection)
{
    case 'A':
    case 'a':
        System.out.print("No. items to add: ");
        amount = Console.in.readInt();
        Console.in.readLine();
        chinookInventory.stockLevel = chinookInventory.stockLevel +
            amount;
        System.out.println("Stock: " + chinookInventory.stockLevel);
        break;
```

James Tam

The Driver Class (4)

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

    System.out.println("Stock: " + chinookInventory.stockLevel);
    break;
```

James Tam

The Driver Class (5)

```
case 'D':
case 'd':
    System.out.println(chinookInventory.stockLevel);
    break;

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

James Tam

The Driver Class (6)

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

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

James Tam

Class Inventory

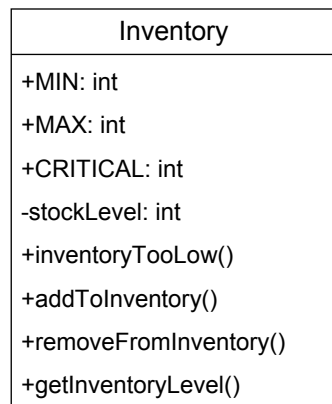
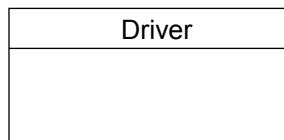
```
public class Inventory
{
    public final int CRITICAL = 10;
    public int stockLevel;

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

James Tam

Utilizing Information Hiding: An Example

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



James Tam

The Driver Class

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

James Tam

The Driver Class (2)

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

James Tam

The Driver Class (3)

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

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

James Tam

The Driver Class (4)

```
case 'D':
case 'd':
    System.out.println(chinookInventory.getInventoryLevel());
    break;

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

James Tam

The Driver Class (5)

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

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

James Tam

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)");
        }
    }
}
```

James Tam

Class Inventory (2)

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

James Tam

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;
    }
}
```

James Tam

Class Inventory (4)

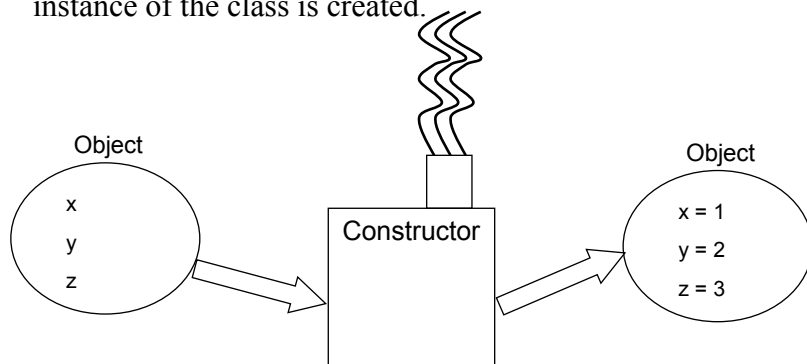
```
public boolean inventoryTooLow ()
{
    if (stockLevel < CRITICAL)
        return true;
    else
        return false;
}

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

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();`

James Tam

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";  
}
```

James Tam

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)
```

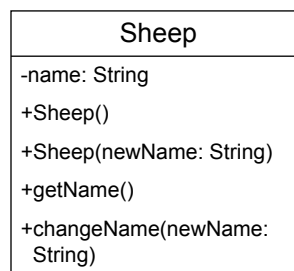
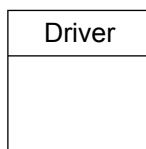
Things to avoid when overloading constructors

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

James Tam

Constructors: An Example

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



James Tam

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();
    }
}
```

James Tam

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();
    }
}
```

James Tam

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;
    }
}
```

James Tam

Class Sheep (2)

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

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

James Tam

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;
    }
}
```

James Tam

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

James Tam

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

James Tam

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

James Tam

Declaring A Reference To An Array

Format:

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

Example:

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

James Tam

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];
```

James Tam

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]);
}
```

James Tam

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

James Tam

Using A Null Reference

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

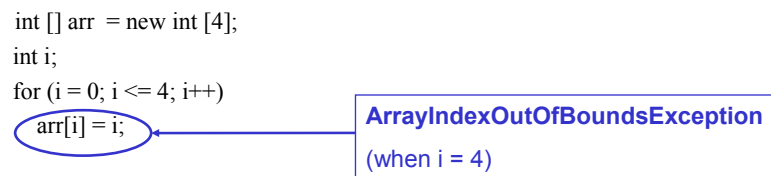


NullPointerException

James Tam

Exceeding The Array Bounds

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



ArrayIndexOutOfBoundsException
(when i = 4)

James Tam

Arrays Of Objects (References)

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

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

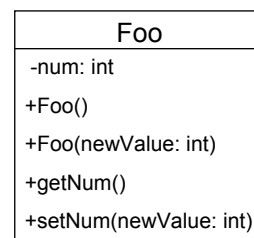
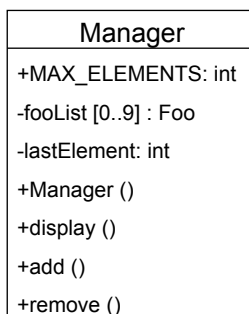
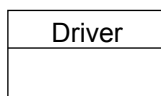
- The elements are initialized to null by default

`arr[0].setNum(1);` → **NullPointerException**

James Tam

Arrays Of References To Objects: An Example

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



James Tam

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();
```

James Tam

The Driver Class (2)

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

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

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

James Tam

The Driver Class (3)

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

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

James Tam

The Manager Class

```
import tio.*;
public 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;
    }
}
```

James Tam

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());
    }
}
```

James Tam

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.");
    }
}
```

James Tam

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");
    }
}
```

James Tam

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; }
}
```

James Tam

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 an object
- What is information hiding and what are the benefits of this approach in the design of 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