

CPSC 233: Introduction to Classes And Objects, Part II

More on Java methods

Relations between classes

- Association
- Aggregation

Multiplicity

Issues associated with references

The finalize method

The static keyword

Classes and state

Debugging code

James Tam

More On Java Methods

- Method overloading and the signature of a method
- Message passing
- Implementation hiding

James Tam

More On Java Methods

- **Method overloading and the signature of a method**
- Message passing
- Implementation hiding

James Tam

Method Overloading

- Same method name but the type, number or order of the parameters is different
- Used for methods that implement similar but not identical tasks.
- Good coding style
- Example:

```
System.out.println(int)  
System.out.println(double)  
etc.
```

For more details on class System see:

<http://java.sun.com/j2se/1.4.2/docs/api/java/io/PrintStream.html>

James Tam

Method Signatures And Method Overloading

Signature consists of:

- The name of the method
- The number **and** type of parameters (Reminder: Don't distinguish methods solely by the order of the parameters).

Usage of method signatures:

- To distinguish overloaded methods (same name but the type, number or ordering of the parameters is different).

James Tam

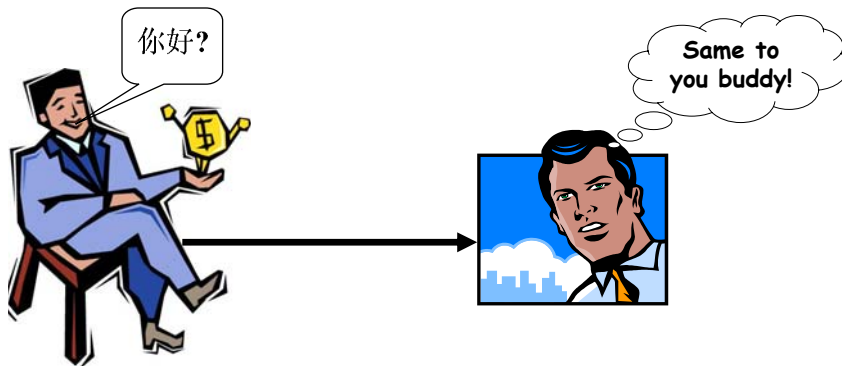
More On Java Methods

- Method overloading and the signature of a method
- **Message passing**
- Implementation hiding

James Tam

Message Passing: General Definition

A communication between a sender and a receiver



James Tam

Message Passing: Object-Oriented Definition

- A request from the source object that's sent to the destination object to apply one of its operations.
- An object invoking the methods of another object (which may be the same object or a different object)

e.g.,

```
class Bar
```

```
{
```

```
    public void aMethod ()
```

```
    {
```

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

```
        f.getNum();
```

```
    }
```

```
}
```

A message is sent
from an instance of
Bar to an instance of
Foo

James Tam

Message Passing And Program Design

Procedural approach

- Start with a function or procedure and pass the composite types to this method
- i.e., procedure (record)
- e.g.,
CD = record
 title : array [1..80] of char;
 artist : array [1..80] of char;
 price : real;
 rating : integer;
 category : char;
end;
Collection = array [1..NO] of CD;
 :
Collection tamjCollection;
 :
procedure displayCollection (tamjCollection : Collection);

James Tam

Message Passing And Program Design (2)

Object-Oriented approach:

- Start with an object and then determine which method to invoke
- i.e., object.method ()
- e.g.,
class Foo
{
 private int num;
 public void setNum (int newValue) { num = newValue; }
 public int getNum () { return num; }
}
 :
f.getNum();

James Tam

More On Java Methods

- Method overloading and the signature of a method
- Message passing
- **Implementation hiding**

James Tam

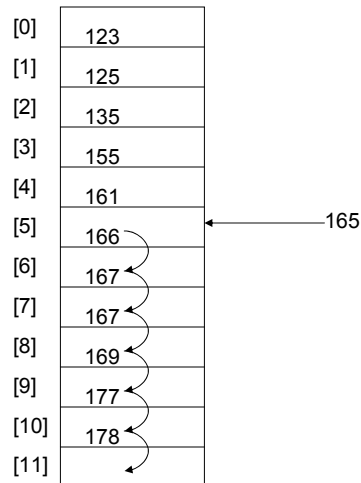
Implementation Hiding

- Allows you to use a program module (e.g., a method) without knowing how the code in the module was written (i.e., you don't care about the implementation).
- For example, a list can be implemented as either an array or as a linked list.

James Tam

Implementation Hiding (2)

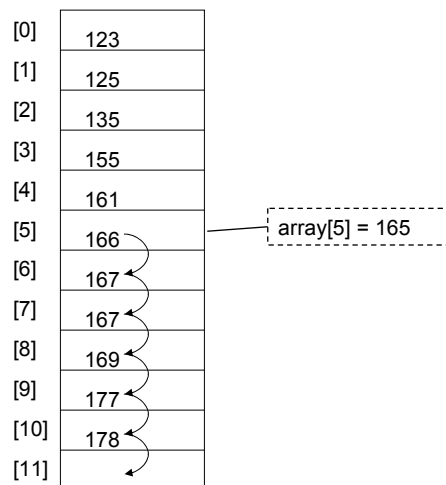
List implemented as an array (add element)



James Tam

Implementation Hiding (2)

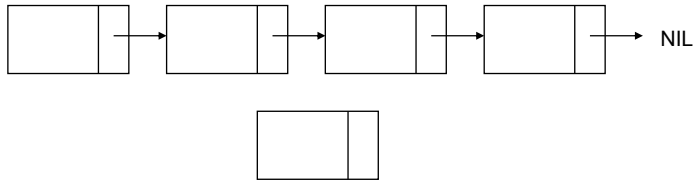
List implemented as an array (add element)



James Tam

Implementation Hiding (3)

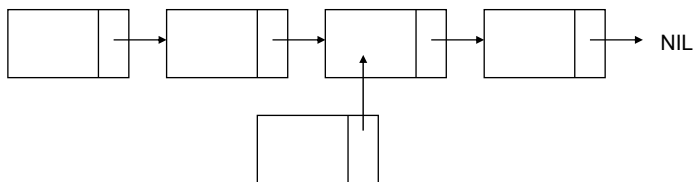
List implemented as a linked list (add element)



James Tam

Implementation Hiding (3)

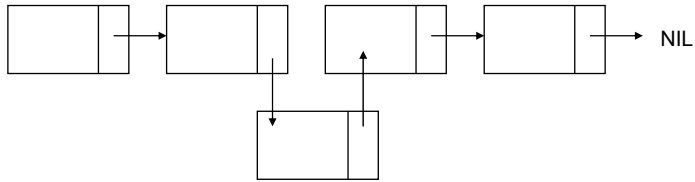
List implemented as a linked list (add element)



James Tam

Implementation Hiding (3)

List implemented as a linked list (add element)



James Tam

Implementation Hiding (4)

- Changing the implementation of the list should have a minimal impact on the rest of the program.
- Changing the implementation of a method is separate from changing the signature of a method.
- The “add” method is a black box.
- We know how to use it without being effected by the details of how it works.

add (list, newElement)

???

James Tam

Relations Between Classes

1. Association (this section of notes)
2. Aggregation (this section of notes)
3. Inheritance (next section of notes)

James Tam

Associations Between Classes

- Allows for navigation between from one class to another (you can access the public parts of the class).
- Associations occur between classes when:
 - An instance of a class is a attribute of another class
 - An instance of a class a local variable in another class's method
- Also known as a “knows-a” relation
- Association relations allows for messages to be sent

James Tam

Associations: Lights Is An Attribute Of Car

```
public class Car
{
    private Lights headLights;
    headLights = new Lights ();
    :
    public startCar () { headLights.turnOn(); }
}

public class Lights
{
    private boolean isOn;
    public void turnOn () { isOn = true;}
}
```

James Tam

Associations: Gasoline Is A Local Variable In A Method Of Class Car

```
public class Car
{
    public startCar ()
    {
        Gasoline fuel = new Gasoline ();
        fuel.burnupForIgnition();
    }
}

public class Gasoline
{
    public void burnUpForIgnition () { .. }
}
```

James Tam

Directed Associations

Unidirectional

- The association only goes in one direction
- You can only navigate from one class to the other (but not the other way around).
- e.g., You can go from an instance of Car to Lights but not from Lights to Car, or you can go from an instance of Car to Gasoline but not from Gasoline to Car (previous slides)

James Tam

Directed Associations (2)

Bidirectional

- The association goes in both directions
- You can navigate from either class to the other

•e.g.,

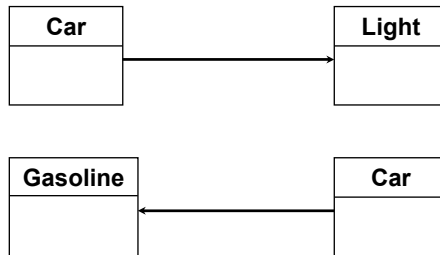
```
public class Student
{
    private Lecture [] lectureList = new Lecture [5];
    :
}

public class Lecture
{
    private Student [] studentList = new Student [250];
    :
}
```

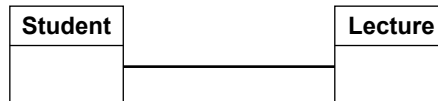
James Tam

UML Representation Of Associations

Unidirectional associations



Bidirectional associations



James Tam

Aggregation Relations Between Classes

- A stronger form of association between classes
- Can occur when a class consists of another class
 - An instance of a class an attribute of another class
- and**
- The first class is conceptually a part of the second class
- Also known as a “has-a” relation

e.g.,

```
public class Company
{
    private Department divisions [];
}
```

James Tam

Graphical Representations Of Aggregations



James Tam

Multiplicity

- It indicates the number of instances that participate in a relationship
- Also known as cardinality

Multiplicity	Description
1	Exactly one instance
n	Exactly “n” instances
n..m	Any number of instances in the inclusive range from “n” to “m”
*	Any number of instances possible

James Tam

Association Vs. Aggregation

- Aggregation is a more specific form of association (one class consists of the other)
- Navigation is not the same as aggregation! (Again: One class is an attribute field of another class **AND** the first class is a part of the second class)

Association

```
public class BankAccount
{
    private Person accountHolder;
    :
}
```

Aggregation

```
public class Person
{
    private Head myHead;
    private Heart myHeart;
    :
}
```

James Tam

Issues Associated With References

- Parameter passing
- Assignment of references and deep vs. shallow copies
- Comparisons of references

James Tam

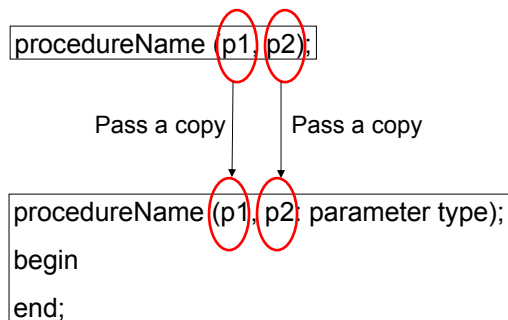
Parameter Passing Mechanisms

- Pass by value
- Pass by reference

James Tam

Passing Parameters As Value Parameters (*Pass By Value*)

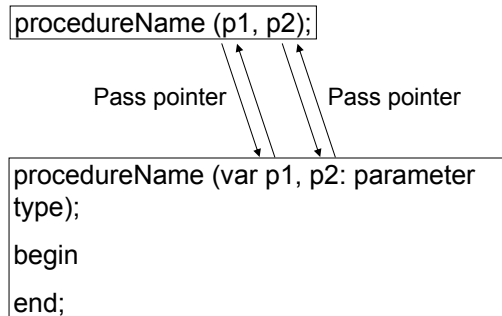
- Changes made to the parameter(s) in the procedure only changes the copy and not the original parameter(s)



James Tam

Passing Parameters As Variable Parameters (*Pass By Reference*)

- Change made to the parameter(s) in the procedure refer to the original parameter(s)



James Tam

Passing Simple Types In Java

Built-in (simple) types are **always** passed by value in Java:

- Boolean, byte, char, short, int, long, double, float

Example:

```
main ()  
{  
    int num1, num2;  
    Foo f = new Foo ();  
    num1 = 1;  
    num2 = 2;  
    System.out.println("num1=" + num1 + "\tnum2=" + num2);  
    f.swap(num1, num2);  
    System.out.println("num1=" + num1 + "\tnum2=" + num2);  
}
```

James Tam

Passing Simple Types In Java (2)

```
public class Foo
{
    public void swap (int num1, int num2)
    {
        int temp;
        temp = num1;
        num1 = num2;
        num2 = temp;
        System.out.println("num1=" + num1 + "\tnum2=" + num2);
    }
}
```

James Tam

Passing References In Java

(Reminder: References are required for variables that are arrays or objects)

Question:

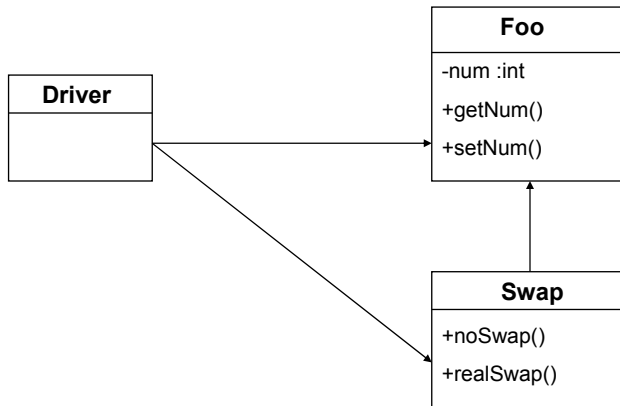
- If a reference (object or array) is passed as a parameter to a method do changes made in the method continue on after the method is finished?

Hint: If a reference is passed as a parameter into a method then a copy of the reference is what is being manipulated in the method.

James Tam

An Example Of Passing References In Java: UML Diagram

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



James Tam

An Example Of Passing References In Java: The Driver Class

```
class Driver
{
    public static void main (String [] args)
    {
        Foo f1, f2;
        Swap s1;
        f1 = new Foo ();
        f2 = new Foo ();
        s1 = new Swap ();
        f1.setNum(1);
        f2.setNum(2);
    }
}
```

James Tam

An Example Of Passing References In Java: The Driver Class (2)

```
System.out.println("Before swap:\t f1=" + f1.getNum() + "\tf2=" +
    f2.getNum());
s1.noSwap (f1, f2);
System.out.println("After noSwap\t f1=" + f1.getNum() + "\tf2=" +
    f2.getNum());
s1.realSwap (f1, f2);
System.out.println("After realSwap\t f1=" + f1.getNum() + "\tf2=" +
    f2.getNum());
    }
}
```

James Tam

An Example Of Passing References In Java: Class Foo

```
public class Foo
{
    private int num;
    public void setNum (int newNum)
    {
        num = newNum;
    }
    public int getNum ()
    {
        return num;
    }
}
```

James Tam

An Example Of Passing References In Java: Class Swap

```
public class Swap
{
    public void noSwap (Foo f1, Foo f2)
    {
        Foo temp;
        temp = f1;
        f1 = f2;
        f2 = temp;
        System.out.println("In noSwap\t f1=" + f1.getNum () + "\tf2=" +
            f2.getNum());
    }
}
```

James Tam

An Example Of Passing References In Java: Class Swap (2)

```
public void realSwap (Foo f1, Foo f2)
{
    Foo temp = new Foo ();
    temp.setNum(f1.getNum());
    f1.setNum(f2.getNum());
    f2.setNum(temp.getNum());
    System.out.println("In realSwap\t f1=" + f1.getNum () + "\tf2=" +
        f2.getNum());
}
} // End of class Swap
```

James Tam

Passing (References) To Arrays As Method Parameters

- Because the copy of an address is passed into the method, passing arrays as a parameter works the same way as passing a reference to a class as a parameter.

e.g.,

```
char [] list = new char[4];  
reference.changeOne(list);
```

```
      :      :      :  
public void changeOne (char [] list)  
{  
    char [] temp = new char[4];  
    list = temp;  
}
```

James Tam

Passing (References) To Arrays As Method Parameters (2)

e.g.,

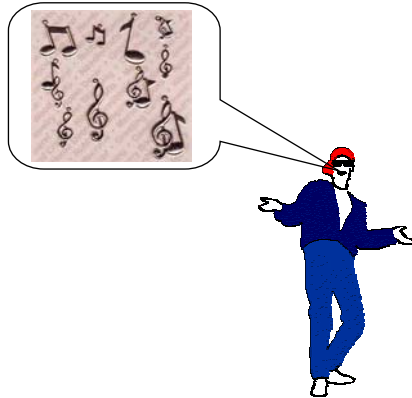
```
char [] list = new char[4];  
reference.changeTwo(list);
```

```
      :      :      :  
  
public void changeTwo (char [] list)  
{  
    list[0] = '*';  
    list[3] = '*';  
}
```

James Tam

Passing By Reference For Simple Types

- It cannot be done directly in Java
- You must use a wrapper!



James Tam

Wrapper Class

- A class definition built around a simple type

e.g.,

```
public class IntegerWrapper
{
    private int num;
    public int getNum () { return num; }
    public void setNum (int newNum) { num = newNum; }
}
```

James Tam

Issues Associated With References

- Parameter passing
- **Assignment of references and deep vs. shallow copies**
- Comparisons of references

James Tam

Assignment Operator: Works On The Reference

```
Foo f1, f2;  
f1 = new Foo ();  
f2 = new Foo ();  
f1.setNum(1);  
f2.setNum(2);  
System.out.println("f1=" + f1.getNum() + "\tf2=" + f2.getNum());  
f1 = f2;  
f1.setNum(10);  
f2.setNum(20);  
System.out.println("f1=" + f1.getNum() + "\tf2=" + f2.getNum());
```

James Tam

Shallow Copy Vs. Deep Copies

Shallow copy

- Copy the address in one reference into another reference
- Both references point to the same dynamically allocated memory location

• e.g.,

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

James Tam

Shallow Vs. Deep Copies (2)

Deep copy

- Copy the contents of the memory location pointed to by the reference
- The references still point to separate locations in memory.

• e.g.,

```
f1 = new Foo ();  
f2 = new Foo ();  
f1.setNum(1);  
f2.setNum(f1.getNum());  
System.out.println("f1=" + f1.getNum() + "\tf2=" + f2.getNum());  
f1.setNum(10);  
f2.setNum(20);  
System.out.println("f1=" + f1.getNum() + "\tf2=" + f2.getNum());
```

James Tam

Issues Associated With References

- Parameter passing
- Assignment of references and deep vs. shallow copies
- **Comparisons of references**

James Tam

Comparison Of The References

```
f1 = new Foo ();  
f2 = new Foo ();  
f1.setNum(1);  
f2.setNum(f1.getNum());  
if (f1 == f2)  
    System.out.println("References point to same location");  
else  
    System.out.println("References point to different locations");
```

James Tam

Comparison Of The Data

```
f1 = new Foo2 ();
f2 = new Foo2 ();
f1.setNum(1);
f2.setNum(f1.getNum());
if (f1.getNum() == f2.getNum())
    System.out.println("Same data");
else
    System.out.println("Different data");
```

James Tam

Self Reference: This Reference

- From every (non-static) method of an object there exists a reference to the object (called the “this” reference)

e.g.,

```
Foo f1 = new Foo ();
Foo f2 = new Foo ();
f1.setNum(10);
```

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

James Tam

Self Reference: This Reference

- From most every method of an object there exists a pointer to the object (“this”)

e.g.,

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

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

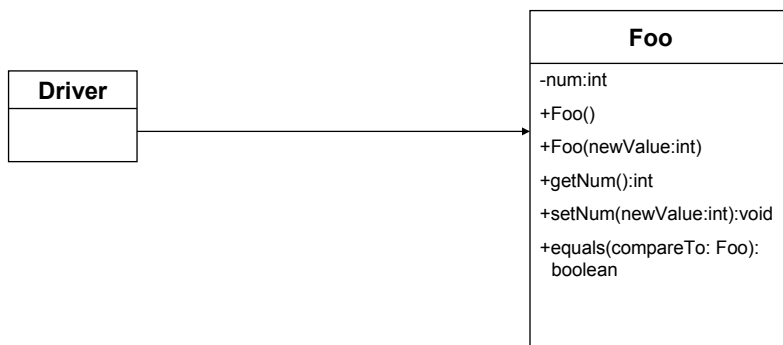
```
f1.setNum(10);
```

```
public class Foo
{
    private int num;
    public void setNum (int num)
    {
        this.num = num;
    }
    :   :
}
```

James Tam

Uses Of “This” When Checking For Equality: UML Diagram

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



James Tam

Checking For Equality: The Driver Class

```
class Driver
{
    public static void main (String [] args)
    {
        Foo f1 = new Foo(1);
        Foo f2 = new Foo(2);
        if (f1.equals(f2))
            System.out.println("Data of f1 and f2 the same.");
        else
            System.out.println("Data of f1 and f2 are not the same.");
    }
}
```

James Tam

Uses Of “This”: Checking For Equality (2)

```
public class Foo
{
    private int num;

    public Foo ()
    {
        num = 0;
    }

    public Foo (int newValue)
    {
        num = newValue;
    }

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

James Tam

Uses Of “This”: Checking For Equality (3)

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

public boolean equals (Foo compareTo)
{
    if (num == compareTo.num)
        return true;
    else
        return false;
}
}
```

James Tam

Uses Of “This”: Checking For Equality (3)

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

public boolean equals (Foo compareTo)
{
    if (this.num == compareTo.num)
        return true;
    else
        return false;
}
}
```

James Tam

Explicit Vs. Implicit Parameters

Explicit parameters

- Are the parameters enclosed within the brackets of a method call.
- e.g.,
Foo f = new Foo ();
int no = 10;
f.setNum(no);

Implicit parameters

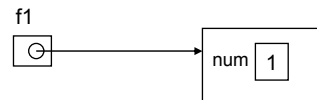
- Do not need to be explicitly passed into a method in order to be used
- The “this” reference is an explicit parameter

James Tam

The Finalize Method

Example sequence:

```
public class Foo
{
    int num;
    public Foo () { num = 1; }
    public Foo (int newValue) { num = newValue; }
    : : :
}
: :
Foo f1 = new Foo ();
```

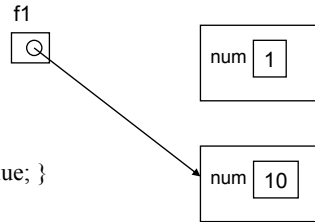


James Tam

The Finalize Method

Example sequence:

```
public class Foo
{
    int num;
    public Foo () { num = 1; }
    public Foo (int newValue) { num = newValue; }
    : : :
}
: : :
Foo f1 = new Foo ();
f1 = new Foo (10);
```

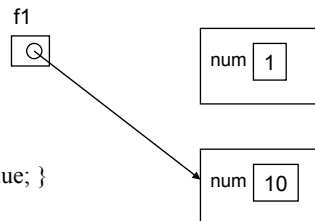


James Tam

The Finalize Method

Example sequence:

```
public class Foo
{
    int num;
    public Foo () { num = 1; }
    public Foo (int newValue) { num = newValue; }
    : : :
}
: : :
Foo f1 = new Foo ();
f1 = new Foo (10);
```



When???

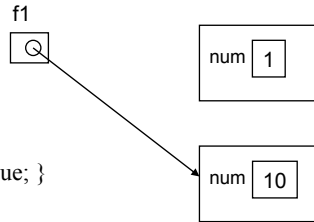


James Tam

The Finalize Method

Example sequence:

```
public class Foo
{
    int num;
    public Foo () { num = 1; }
    public Foo (int newValue) { num = newValue; }
    : : :
}
: : :
Foo f1 = new Foo ();
f1 = new Foo (10);
```



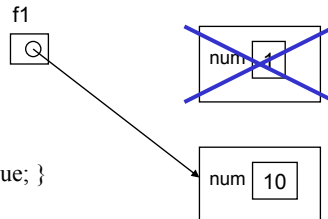
f1.finalize()

James Tam

The Finalize Method

Example sequence:

```
public class Foo
{
    int num;
    public Foo () { num = 1; }
    public Foo (int newValue) { num = newValue; }
    : : :
}
: : :
Foo f1 = new Foo ();
f1 = new Foo (10);
```



f1.finalize()

James Tam

Synopsis Of The Finalize Method

- The Java interpreter tracks what memory has been dynamically allocated.
- It also tracks when memory is no longer referenced.
- When system isn't busy, the Automatic Garbage Collector is invoked.
- If an object has a finalize method then it is invoked:
 - The finalize is a method written by the programmer to free up non-memory resources e.g., closing and deleting temporary files created by the program, network connections.
 - This method takes no arguments and returns no values.
 - Dynamic memory is **NOT** freed up by this method.
- After the finalize method finishes execution, the dynamic memory is freed up by the Automatic Garbage Collector.

James Tam

A Previous Example Revisited: Class Sheep

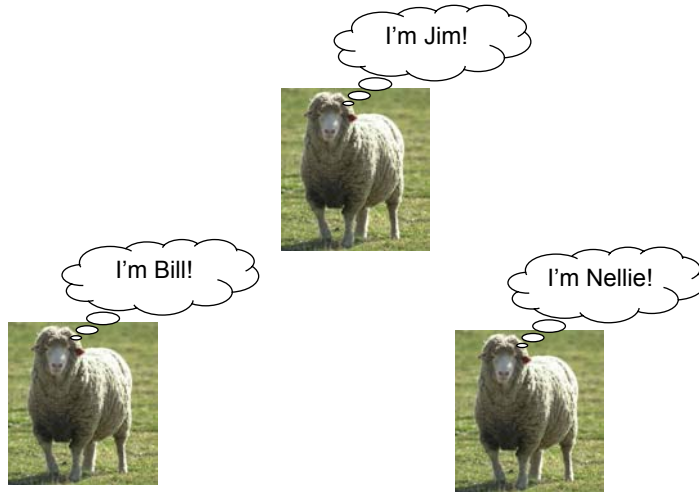
```
public class Sheep
{
    private String name;

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

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

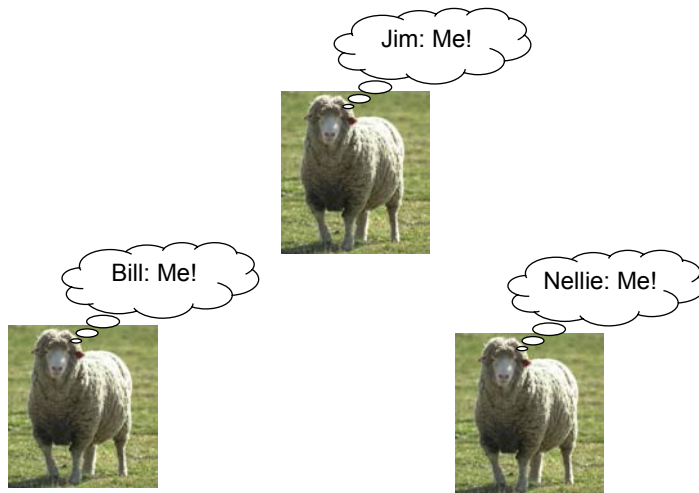
James Tam

We Now Have Several Sheep



James Tam

Question: Who Tracks The Size Of The Herd?



James Tam

Answer: None Of The Above!

- Information about all instances of a class should not be tracked by an individual object
- So far we have used instance fields
- Each *instance* of an object contains *it's own set of instance fields* which can contain information unique to the instance

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

name: Bill

name: Jim

name: Nellie

James Tam

The Need For Static (Class Fields)

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

Class Sheep
flockSize

object

name: Bill

object

name: Jim

object

name: Nellie

James Tam

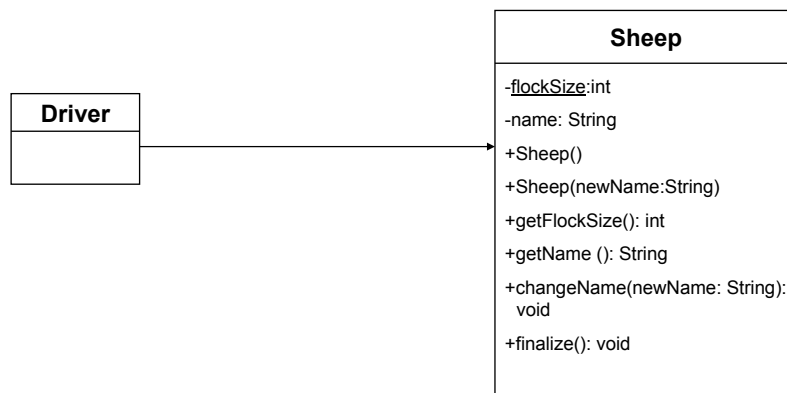
Static (Class) Methods

- Are associated with the class as a whole and not individual instances of the class
- Typically implemented for classes that are never instantiated e.g., Math
- May also be used act on the class fields

James Tam

Static Data And Methods: UML Diagram

Example (The complete example can be found in the directory /home/233/examples/advancedOO/thirdExample)



James Tam

Static Data And Methods: The Driver Class

```
class Driver
{
    public static void main (String [] args)
    {
        System.out.println();
        System.out.println("You start out with " + Sheep.getFlockSize() + " sheep");
        System.out.println("Creating flock...");
        Sheep nellie = new Sheep ("Nellie");
        Sheep bill = new Sheep("Bill");
        Sheep jim = new Sheep();
    }
}
```

James Tam

Static Data And Methods: The Driver Class (2)

```
        System.out.print("You now have " + Sheep.getFlockSize() + " sheep:");
        jim.changeName("Jim");
        System.out.print("\t"+ nellie.getName());
        System.out.print(", "+ bill.getName());
        System.out.println(", "+ jim.getName());
        System.out.println();
    }
} // End of Driver class
```

James Tam

Static Data And Methods: The Sheep Class

```
public class Sheep
{
    private static int flockSize;
    private String name;

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

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

James Tam

Static Data And Methods: The Sheep Class (2)

```
public static int getFlockSize () { return flockSize; }

public String getName () { return name; }

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

public void finalize ()
{
    System.out.print("Automatic garbage collector about to be called for ");
    System.out.println(this.name);
    flockSize--;
}
} // End of definition for class Sheep
```

James Tam

Static Vs. Final

- **Static:** Means there's one instance of the field for the class (not individual instances for each instance of the class)
- **Final:** Means that the field cannot change (it is a constant)

```
public class Foo
{
    public static final int num1= 1;
    private static int num2;           /* Rare */
    public final int num3 = 1;        /* Why bother? */
    private int num4;
    :      :
}
```

James Tam

Rules Of Thumb: Instance Vs. Class Fields

- If a attribute field can differ between instances of a class:
 - The field probably should be an instance field
- If the attribute field relates to the class or to all instances of the class
 - The field probably should be a static field of the class

James Tam

Rule Of Thumb: Instance Vs. Class Methods

- If a method should be invoked regardless of the number of instances that exist then it probably should be a static method
- Otherwise the method should likely be an instance method.

James Tam

An Example Class With A Static Implementation

```
public class Math
{
    // Public constants
    public static final double E = 2.71...
    public static final double PI = 3.14...

    // Public methods
    public static int abs (int a);
    public static long abs (long a);
    :           :
}
```

For more information about this class go to:

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Math.html>

James Tam

Should A Class Be Entirely Static?

- Generally should be avoided if possible
- Usually purely static classes (cannot be instantiated) have only methods and no data (maybe some constants)

James Tam

A Common Error With Static Methods

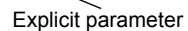
Recall: The “this” reference is an implicit parameter that is automatically passed into a method.

e.g.,

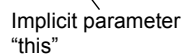
```
Foo f = new Foo ();
```

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

Explicit parameter



Implicit parameter
“this”



James Tam

A Common Error With Static Methods

Static methods have no “this” reference as an implicit parameter.

```
class Driver
{
    private int num;
    public static void main (String [] args)
    {
        num = 10;
    }
}
```

Compilation error:

Driver3.java:6: non-static variable num cannot be referenced from a static context

```
num = 10;
```

```
^
```

error

James Tam

Classes And State

- The state of an object is determined by the values of it's attributes.
- The states of objects can be modeled by State diagrams
- Not all attributes are modeled, attributes that typically are modeled in the state of an object include:
 - Attributes that can only take on a limited range of values e.g., boolean
 - Attributes that have restrictions regarding the values that it may take on. e.g., programmer defined ranges for a long may only include 1, 2, 3 or 4.

James Tam

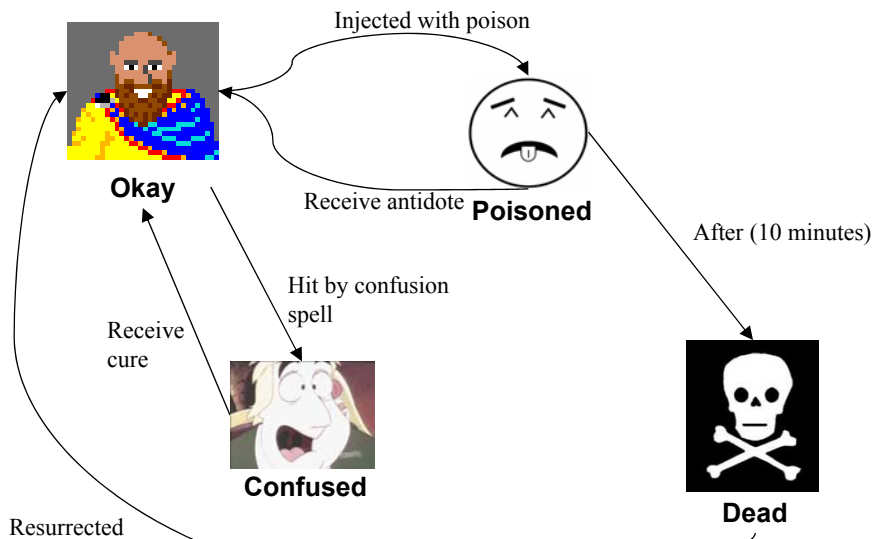
Example Class: Adventurer

```
public class Adventurer
{
    private boolean okay;
    private boolean poisoned;
    private boolean confused;
    private boolean dead;
    :
}
```



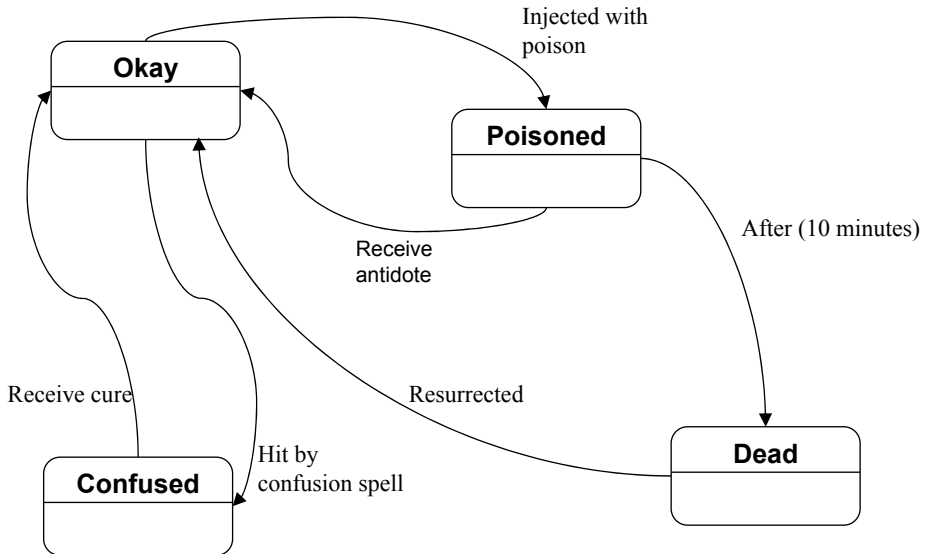
James Tam

Class Adventurer: The Set Of States



James Tam

Class Adventurer: State Diagram



James Tam

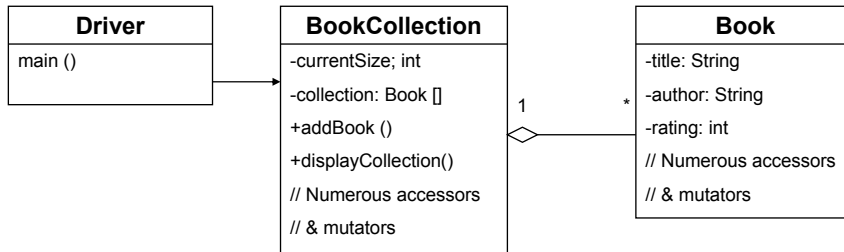
Determining The State Of Objects

- Determining the value of the attributes of a object (state) can be a useful debugging tool.

James Tam

An Example Of Determining Object's State: UML Diagram

Example (The complete example can be found in the directory
/home/233/examples/advancedOO/fourthExample



James Tam

An Example Of Determining Object's State: The Driver Class

```
class Driver
{
    public static void main (String [] args)
    {
        BookCollection tamjCollection = new BookCollection ();
        tamjCollection.displayCollection();
    }
}
```

James Tam

An Example Of Determining Object's State: The BookCollection Class

```
public class BookCollection
{
    public final static int MAX_SIZE = 4;
    private int currentSize;
    private Book [] collection;

    public BookCollection ()
    {
        int i;
        currentSize = 0;
        collection = new Book [MAX_SIZE];
        for (i = 0; i < MAX_SIZE; i++)
        {
            addBook();
        }
    }
}
```

James Tam

An Example Of Determining Object's State: The BookCollection Class (2)

```
public int getCurrentSize () { return currentSize; }

public void setCurrentSize (int newSize) { currentSize = newSize; }

public void addBook ()
{
    if ((currentSize+1) < MAX_SIZE)
    {
        Book b = new Book ();
        b.setAllFields();
        collection[currentSize] = b;
        currentSize++;
    }
}
```

James Tam

An Example Of Determining Object's State: The BookCollection Class (3)

```
public void displayCollection ()
{
    int i, no;
    System.out.println("\nDISPLAYING COLLECTION");
    no = 1;
    for (i = 0; i < currentSize; i++)
    {
        System.out.println("\tBook #" + no);
        System.out.println("\tTitle: " + collection[i].getTitle());
        System.out.println("\tAuthor: " + collection[i].getAuthor());
        System.out.println("\tRating: " + collection[i].getRating());
        System.out.println();
        no++;
    }
}
} // End of the BookCollection class
```

James Tam

An Example Of Determining Object's State: The Book Class

```
public class Book
{
    private String title;
    private String author;
    private int rating;
    public Book ()
    {
        title = "No title given";
        author = "No author listed";
        rating = -1;
    }
    public Book (String newTitle, String newAuthor, int newRating)
    {
        title = newTitle;
        author = newAuthor;
        rating = newRating;
    }
}
```

James Tam

An Example Of Determining Object's State: The Book Class (2)

```
public String getTitle ()
{
    return title;
}

public void setTitle (String newTitle)
{
    title = newTitle;
}

public String getAuthor ()
{
    return author;
}

public void setAuthor (String newAuthor)
{
    author = newAuthor;
}
```

James Tam

An Example Of Determining Object's State: The Book Class (3)

```
public int getRating ()
{
    return rating;
}

public void setRating (int newRating)
{
    if ((newRating >= 1) && (newRating <= 5))
        rating = newRating;
    else
        System.out.println("The rating must be a value between 1 and 5
            (inclusive)");
}
```

James Tam

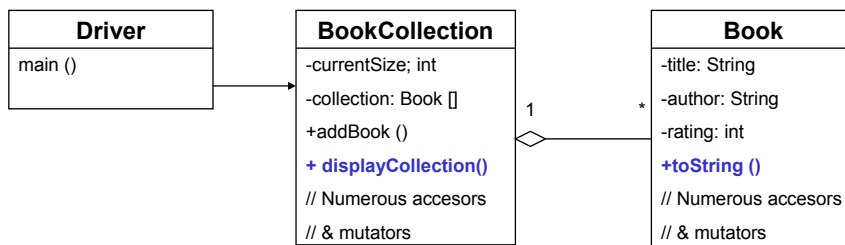
An Example Of Determining Object's State: The Book Class (4)

```
public void setAllFields ()
{
    System.out.print("Enter the title of the book: ");
    title = Console.in.readLine();
    System.out.print("Enter the author of the book: ");
    author = Console.in.readLine();
    do
    {
        System.out.print("How would you rate the book (1 = worst, 5 = best): ");
        rating = Console.in.readInt();
        if ((rating < 1) || (rating > 5))
            System.out.println("Rating must be a value between 1 and 5");
    } while ((rating < 1) || (rating > 5));
    Console.in.readLine();
    System.out.println();
}
} // End of class Book
```

James Tam

An Revised Example Of Determining Object's State: UML Diagram

Example (The complete example can be found in the directory
/home/233/examples/advancedOO/fifthExample



James Tam

A Revised Example Of Determining Object's State: The Driver Class

```
class Driver
{
    public static void main (String [] args)
    {
        BookCollection tamjCollection = new BookCollection ();
        tamjCollection.displayCollection();
    }
}
```

James Tam

A Revised Example Of Determining Object's State: The BookCollection Class

```
public class BookCollection
{
    public final static int MAX_SIZE = 4;
    private int currentSize;
    private Book [] collection;

    public BookCollection ()
    {
        int i;
        currentSize = 0;
        collection = new Book [MAX_SIZE];
        for (i = 0; i < MAX_SIZE; i++)
        {
            addBook();
        }
    }
}
```

James Tam

A Revised Example Of Determining Object's State: The BookCollection Class (2)

```
public int getCurrentSize ()
{
    return currentSize;
}

public void setCurrentSize (int newSize)
{
    currentSize = newSize;
}

public void addBook ()
{
    Book b = new Book ();
    b.setAllFields();
    collection[currentSize] = b;
    currentSize++;
}
```

James Tam

A Revised Example Of Determining Object's State: The BookCollection Class (3)

```
public void displayCollection ()
{
    int i, no;
    System.out.println("\nDISPLAYING COLLECTION");
    no = 1;
    for (i = 0; i < MAX_SIZE; i++)
    {
        System.out.println("\tBook #" + no);
        System.out.println(collection[i]);
        System.out.println();
        no++;
    }
}
} // End of class BookCollection
```

James Tam

A Revised Example Of Determining Object's State: The Book Class

```
public class Book
{
    private String title;
    private String author;
    private int rating;

    public Book ()
    {
        title = "No title given";
        author = "No author listed";
        rating = -1;
    }

    public Book (String newTitle, String newAuthor, int newRating)
    {
        title = newTitle;
        author = newAuthor;
        rating = newRating;
    }
}
```

James Tam

A Revised Example Of Determining Object's State: The Book Class (2)

```
public String getTitle ()
{
    return title;
}

public void setTitle (String newTitle)
{
    title = newTitle;
}

public String getAuthor ()
{
    return author;
}

public String getTitle ()
{
    return title;
}
```

James Tam

A Revised Example Of Determining Object's State: The Book Class (3)

```
public void setTitle (String newTitle)
{
    title = newTitle;
}

public String getAuthor ()
{
    return author;
}

public void setRating (int newRating)
{
    if ((newRating >= 1) && (newRating <= 5))
        rating = newRating;
    else
        System.out.println("The rating must be a value between 1 and 5
        (inclusive);");
}
```

James Tam

A Revised Example Of Determining Object's State: The Book Class (4)

```
public void setAllFields ()
{
    System.out.print("Enter the title of the book: ");
    title = Console.in.readLine();
    System.out.print("Enter the author of the book: ");
    author = Console.in.readLine();
    do
    {
        System.out.print("How would you rate the book (1 = worst, 5 = best): ");
        rating = Console.in.readInt();
        if ((rating < 1) || (rating > 5))
            System.out.println("Rating must be a value between 1 and 5");
    } while ((rating < 1) || (rating > 5));
    Console.in.readLine();
    System.out.println();
}
```

James Tam

A Revised Example Of Determining Object's State: The Book Class (5)

```
public String toString ()
{
    String temp = new String ();
    temp = temp + "\tTitle: " + title + "\n";
    temp = temp + "\tAuthor: " + author + "\n";
    temp = temp + "\tRating: " + rating + "\n";
    return temp;
}
} // End of class Book
```

James Tam

You Should Now Know

- New terminology and concepts relevant to methods: message passing, method signatures, overloading of methods
- What is implementation hiding and what is the benefit of employing it
- Two types of relationships that can exist between classes: associations and aggregation
- Some specific issues and problems associated with Java references
 - The parameter passing mechanism that is employed for different types in Java
 - How does the assignment and comparison of references work in Java
 - What is the "this" reference: how does it work and when is used

James Tam

You Should Now Know (2)

- More advanced concepts in the Java garbage collection process: the finalize method and how it fits into the garbage collection of references
- What is the difference between static and instance methods, and static and instance attributes and when to should each one be employed
- Classes and states:
 - What is meant by the state of an instance of a class
 - Debugging programs by examining the state of instances