

Advanced Java Programming

After mastering the basics of Java you will now learn more complex but important programming concepts as implemented in Java.

James Tam

Commonly Implemented Methods

- The particular methods implemented for a class will vary depending upon the application.
- However two methods that are commonly implemented for many classes:
 - toString
 - equals

James Tam

“Method: toString”

- It’s commonly written to allow easy determination of the state of a particular object (contents of important attributes).
- This method returns a string representation of the state of an object.
- It will automatically be called whenever a reference to an object is passed as a parameter is passed to the “print/println” method.
- The full example can be found online under:
</home/233/examples/advancedJava/firstExample>

James Tam

Class Person: Version 1

```
public class Person
{
    private String name;
    private int age;
    public Person () {name = "No name"; age = -1; }
    public void setName (String aName) { name = aName; }
    public String getName () { return name; }
    public void setAge (int anAge) { age = anAge; }
    public int getAge () { return age; }
}
```

James Tam

Class Person: Version 2

```
public class Person2
{
    private String name;
    private int age;
    public Person2 () {name = "No name"; age = -1; }
    public void setName (String aName) { name = aName; }
    public String getName () { return name; }
    public void setAge (int anAge) { age = anAge; }
    public int getAge () { return age; }

    public String toString ()
    {
        String temp = "";
        temp = temp + "Name: " + name + "\n";
        temp = temp + "Age: " + age + "\n";
        return temp;
    }
}
```

James Tam

The Driver Class

```
class Driver
{
    public static void main (String args [])
    {
        Person p1 = new Person ();
        Person2 p2 = new Person2 ();
        System.out.println(p1);
        System.out.println(p2);
    }
}
```

James Tam

“Method: equals”

- It's written in order to determine if two objects of the same class are in the same state (attributes have the same data values).
- The full example can be found online under:
</home/233/examples/advancedJava/secondExample>

James Tam

The Driver Class

```
public class Driver
{
    public static void main (String args [])
    {
        Person p1 = new Person ();
        Person p2 = new Person ();
        if (p1.equals(p2) == true)
            System.out.println ("Same");
        else
            System.out.println ("Different");

        p1.setName ("Foo");
        if (p1.equals(p2) == true)
            System.out.println ("Same");
        else
            System.out.println ("Different");
    }
}
```

James Tam

The Person Class

```
public class Person
{
    private String name;
    private int age;
    public Person () {name = "No name"; age = -1; }
    public void setName (String aName) { name = aName; }
    public String getName () { return name; }
    public void setAge (int anAge) { age = anAge; }
    public int getAge () { return age; }
    public boolean equals (Person aPerson)
    {
        boolean flag;
        if ((name.equals(aPerson.getName())) && (age == aPerson.getAge ()))
            flag = true;
        else
            flag = false;
        return flag;
    }
}
```

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

```
fun (p1);
```

Pass a copy
of the data

```
fun (<parameter type> <p1>)  
{  
}
```

James Tam

Passing Parameters As Reference Parameters

```
fun (p1);
```

Pass address

```
fun (<parameter type> <p1>)  
{  
  e */  
}
```

James Tam

Parameter Passing In Java: Simple Types

- All simple types are always passed by value in Java.

Type	Description
byte	8 bit signed integer
short	16 bit 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

Parameter Passing In Java: Simple Types (2)

Example The full example can be found online under:
</home/233/examples/advancedJava/thirdExample>

```
public static void main (String [] args)
{
    int num1;
    int num2;
    Swapper s = new Swapper ();
    num1 = 1;
    num2 = 2;
    System.out.println("num1=" + num1 + "\t num2=" + num2);
    s.swap(num1, num2);
    System.out.println("num1=" + num1 + "\t num2=" + num2);
}
```

James Tam

Passing Simple Types In Java (2)

```
public class Swapper
{
    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

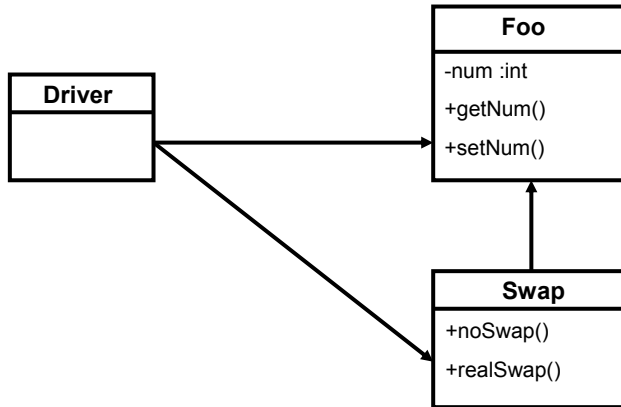
- (Reminder: References are required for variables that are arrays or objects)
- Question:
 - If a reference (object or array) is passed as a parameter to a method do changes made in the method continue on after the method is finished?

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/advancedJava/fourthExample



James Tam

An Example Of Passing References In Java: The Driver Class

```
public class Driver
{
    public static void main (String [] args)
    {
        Foo f1;
        Foo 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

References: Things To Keep In Mind

- If you refer to just the name of the reference then you are dealing with the reference (to an object, to an array).
 - E.g., `f1 = f2;`
 - This copies an address from one reference into another reference, the original objects don't change.
- If you use the dot -perator then you are dealing with the actual object.
 - E.g.,
 - `temp = f2;`
 - `temp.setNum (f1.getNum());`
 - `temp` and `f2` refer to the same object and using the dot operator changes the object which is referred to by both references.
- Other times this may be an issue
 - Assignment
 - Comparisons

James Tam

Shallow Copy Vs. Deep Copies

- Shallow copy
 - Copy the address from one reference into another reference
 - Both references point to the same dynamically allocated memory location
 - e.g.,

```
Foo f1;  
Foo 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 referred 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

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

A Previous Example Revisited: Class Sheep

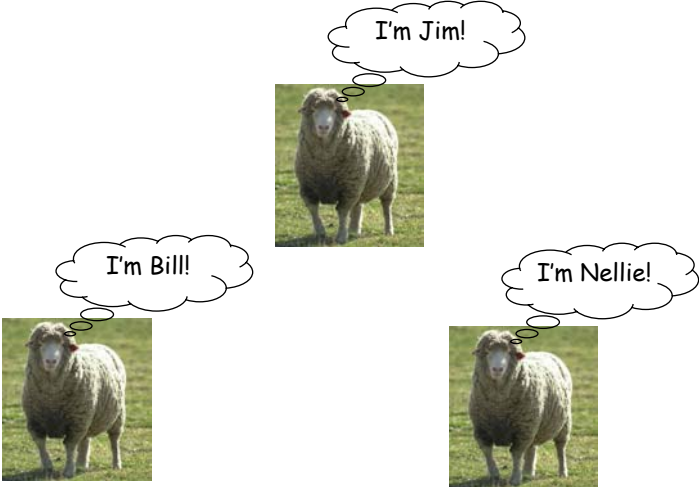
```
public class Sheep
{
    private String name;

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

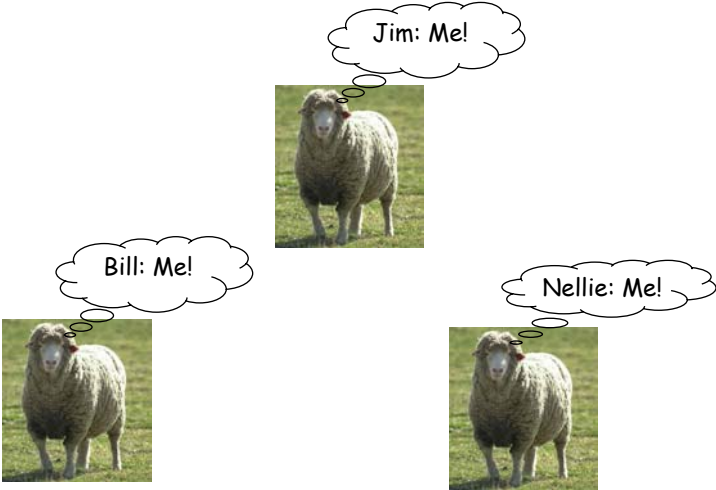
    public void setName (String newName) { name = newName; }
}
```

James Tam

We Now Have Several Sheep



Question: Who Tracks The Size Of The Herd?



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

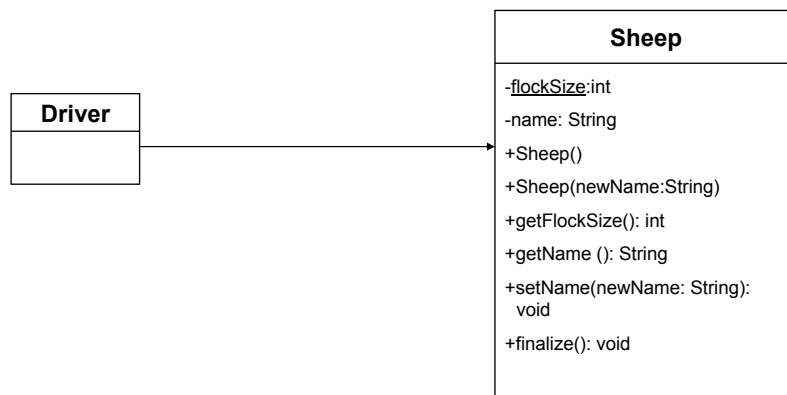
Static (Class) Methods

- 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., class 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/advancedJava/fifthExample`



James Tam

Static Data And Methods: The Driver Class

```
public 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.setName("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 = 0;
    private String name;

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

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

James Tam

Static Data And Methods: The Sheep Class (2)

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

public String getName () { return name; }

public void setName (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

Rules Of Thumb: Instance Vs. Class Fields

- If an attribute field can differ between instances of a class:
 - The field probably should be an instance field (non-static)
- If the attribute field relates to the class (rather to a particular instance) 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 (e.g., the method can be run when there are no instances) then it probably should be a static method.
- If it never makes sense to instantiate an instance of a class then the method should probably be a static method.
- Otherwise the method should likely be an instance method.

James Tam

Static Vs. Final

- **Static:** Means there's one instance of the field for the class (not individual instances of the field 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

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/1.5.0/docs/api/java/lang/Math.html>

James Tam

Should A Class Be Entirely Static?

- Generally it should be avoided if possible because it often bypasses many of the benefits of the Object-Oriented approach.
- Usually purely static classes (cannot be instantiated) have only methods and no data (maybe some constants).
- When in doubt do not make attributes and methods static.

James Tam

A Common Error With Static Methods

- Recall: The “this” reference is an implicit parameter that is automatically passed into the method calls (you’ve seen so far).

• e.g.,

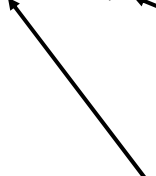
• `Foo f = new Foo ();`

• `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 (because they are not associated with any instances).

```
public 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

Recursion

- What is recursion: a method that calls itself either directly or indirectly.

- Direct call

```
class Foo
{
    public void method ()
    {
        method ();
        :
    }
    :
}
```

James Tam

Recursion: Definition (2)

- Indirect call

```
class Foo
{
    public void method1 ()
    {
        method2 ();
        :
    }
    public void method2 ()
    {
        method1 ();
        :
    }
    :
}
```

James Tam

Requirements For Sensible Recursion

- Base case

- The situation under which the recursive calls stop.
- (There can be multiple base cases but in order for recursion to get set up properly there must be at least one base case).

- Recursive case

- The situation under which the method calls itself.

- Progress towards the base case

- Successive recursive calls draw the program closer towards the base case.

James Tam

Recursion: A Simple Counting Example

- Note: This example could have been implemented with similar logic using a loop.
- The full example can be found online under:
- /home/233/examples/advancedJava/sixthExample

James Tam

Recursion: A Simple Counting Example (2)

```
public class RecursiveCount
{
    public static final int LAST = 5;
    public void doCount (int num) // num starts at 1
    {
        if (num <= LAST)
        {
            System.out.print(num + " ");
            doCount(++num);
        }
        else
            return;
    }
}
```

James Tam

Recursive Example: Sum Of A Series

- There are three variants of the program which can be found in UNIX under:

/home/233/examples/advancedJava/seventhExample

- Driver.java: sum of a series of numbers from one to three.
- Driver2.java: Similar to the original example but missing the base case.
- Driver3.java: Similar to the original example but no progress is made towards the base case.

James Tam

Example 1: Sum Of Series

```
public class Driver
{
    public static int sum (int num)
    {
        if (num == 1) // Base case
            return 1;
        else // Recursive case
            return (num + sum(num-1));
    }

    public static void main (String args [])
    {
        int sum = 0;
        int last = 3;
        int total = 0;
        total = sum(last);
        System.out.println("Sum from 1-" +last + " is " + total);
    }
}
```

James Tam

Example 2: No Base Case

```
public class Driver2
{
    public static int sum (int num)
    {
        // Base case (missing in this version).

        // Recursive case
        return (num + sum(num-1));
    }
}
```

James Tam

Example 3: No Progress Towards The Base Case

```
public class Driver3
{
    public static int sum (int num)
    {
        if (num == 1) // Base case
            return 1;
        else // Recursive case
            return (num + sum(num));
    }
}
```

James Tam

Recursion And The System Stack

- The system stack: used to store local memory for method calls (local variables, parameters, return values).
- Implementing recursion may use up too much of the system stack.
- Operating systems react differently to this error
 - e.g., overflowing the stack in UNIX results in a segmentation fault.
- Some programming languages may also deal with this type of error
 - e.g., Java StackOverflowError

James Tam

Types Of Recursion

- Tail recursion:
 - A recursive call is the last statement in the recursive method.
 - This form of recursion can easily be replaced with a loop.
- Non-tail recursion:
 - The last statement in the method is not a recursive call (excludes return statements).
 - This form of recursion is very difficult (read: impossible) to replace with a loop.
- The full examples (tail and non-tail recursion) can be found in:
 - /home/233/examples/advancedJava/eighthExample

James Tam

Example: Tail Recursion

```
// On the first call to the method, num = 1
public static void tail (int num)
{
    if (num <= 3)
    {
        System.out.print(num + " ");
        tail(num+1);
    }
    return;
}
```

James Tam

Example: Non-Tail Recursion

```
// On the first call to the method, num = 1
public static void nonTail (int num)
{
    if (num < 3)
    {
        nonTail(num+1);
    }
    System.out.print(num + " ");
    return;
}
```

James Tam

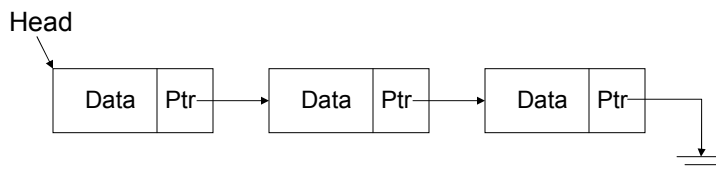
When To Use Recursion

- Recall: recursive method calls that employ tail recursion can typically be replaced with a loop.
- The use of recursion adds overhead (allocating and de-allocating memory on the stack).
 - If an alternative solution to recursion can be implemented then the alternative probably should be chosen.
- There will be times that only a recursive solution will work.
 - Typically this involves 'backtracking'.
 - The repetition of the recursive call is similar to a loop.
 - However in cases where a loop will not work, after the base case has been reached and the recursion 'unwinds' (each method returns back to it's caller), additional actions must occur.
 - This additional unwinding backtracks along each recursive call and performs a necessary task (thus the last statement is not a recursive call, non-tail recursion).

James Tam

Examples Of When To Use Recursion

- Displaying a linked list in reverse order.



James Tam

Examples Of When To Use Recursion (2)

- Finding the exit from a maze.

#	#	#	#	#	#	#	#	#	#
#		#	#	#	#	#	#	#	#
#		#	#	#	#	#	#	#	#
		#	#	#	#	#	#	#	#
#		#	#	#	#		#	#	#
#		#	#	#	#		#	#	#
#		#	#	#		S			#
#		#	#	#		#	#	#	#
#						#	#	#	#
#	#	#	#	#	#	#	#	#	#

Suppose that the order for checking directions for the exit is: North, West, East, South

James Tam

After This Section You Should Now Know

- Two useful methods that should be implemented for almost every class: toString and equals
- What is the difference between pass by value vs. pass by reference
- The difference between references and objects
- Issues associated with assignment and comparison of objects vs. references
- The difference between a deep vs. a shallow copy
- What is a static method and attribute, when is appropriate for something to be static and when is it inappropriate (bad style)
- How to write and trace recursive methods
- The requirements for sensible recursion

James Tam

After This Section You Should Now Know (2)

- The difference between tail and non-tail recursion
- What is the system stack and the role that it plays in recursion
- When should recursion be used and when an alternate should be used