

Advanced Java Programming

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

References: What You Should Know

- In Java when you use objects and arrays there are two things involved:
 - Reference
 - Object (or array)
- Example with an object

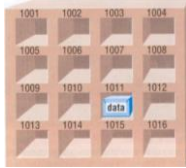
```
Person charlie; // Creates reference to object
charlie = new Person("Sheen"); // Creates object
```
- Example with an array

```
double [] salaries; // Creates reference to array
salaries = new double[100]; // Creates array
```

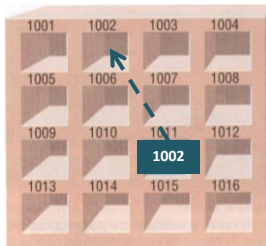
James Tam

Reminder: Variables

- Think of the 'mail box' metaphor



- A slot containing a reference (and not data) will contain the number of another 'slot' (address) in memory.

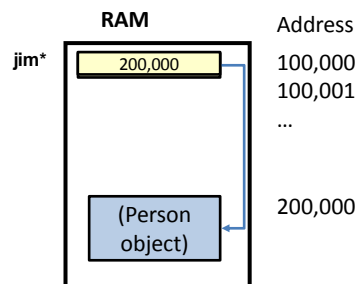


Pictures from Computers in your future by Pfaffenberger B

James Tam

References (Can) Contain Addresses

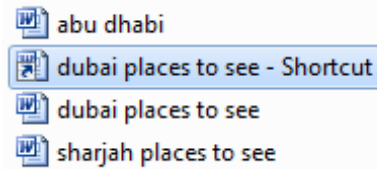
```
Person jim;
jim = new Person();
```



James Tam

Shallow Copy Vs. Deep Copies

- Shallow copy (new term, concept should be review)



A shortcut ('link' in UNIX) is similar to a shallow copy. Multiple things that refer to the same item (document)

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

James Tam

Shallow Copy Vs. Deep Copies (2)

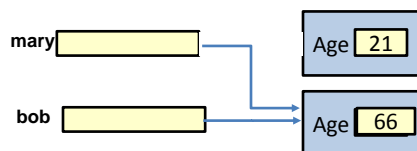
- Shallow copy, full example under:

```
/home/233/examples/advanced/1shallowDeep
```

```
Person mary = new Person(21);
Person bob = new Person(12);
System.out.println(mary.age + " " + bob.age);
    mary.age;
mary = bob; // Shallow;
bob.age = 66;
System.out.println(mary.age + " " + bob.age);
```

21 12

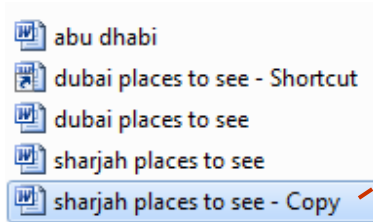
66 66



James Tam

Shallow Copy Vs. Deep Copies (3)

- Deep copy (new term, concept should be review)



Making an actual physical copy is similar to a deep copy.

- Don't copy addresses stored in the references
- Instead the data referred to by the references are copied
- After the copy each reference still refers to a different address (data variable)

James Tam

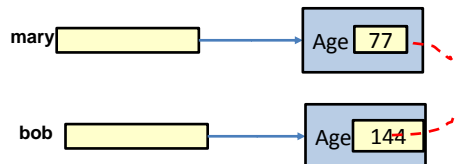
Shallow Copy Vs. Deep Copies (4)

- Deep copy, full example under:

/home/233/examples/advanced/1shallowDeep

```
// Mary still 66
bob = new Person(77);
mary.age = bob.age; // Deep
bob.age = 144;
System.out.println(mary.age + " " + bob.age);
```

77 144



James Tam

Methods Of Parameter Passing

- Pass by value
 - The data stored (the “*value*” stored) in the parameter is copied
- Pass by reference
 - Pass the address of the parameter
 - This allows references to the parameter inside the method (the method has a “*reference*” to the original parameter).

James Tam

Passing Parameters As Value Parameters

```
method (p1);
```

Pass a copy
of the data

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

James Tam

Passing Parameters As Reference Parameters

```
method (p1);
```

Pass the address of the parameter (*refer* to the parameter in the method)

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

James Tam

Which Parameter Passing Mechanism Is Used?

Passed by value

- All 'simple' built in types:
 - Integers (byte, short, int, long)
 - Floating point (float, double)
 - Character (char)
 - Boolean (boolean)

Pass by reference

- Objects
- Arrays
- (That is anything that consists of a reference and the item referenced).

Parameter Passing Example

- Full example under:
</home/233/examples/advanced/2parameters>

James Tam

Class Person

```
public class Person {  
    private int age;  
    private String name;  
  
    public Person() {  
        age = -1;  
        name = "none";  
    }  
  
    public int getAge() {  
        return(age);  
    }  
  
    public String getName() {  
        return(name);  
    }  
}
```

James Tam

Class Person (2)

```
public void setAge(int anAge) {
    age = anAge;
}

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

James Tam

Class ParameterExample

```
public class ParameterExample
{
    public void modify(Person aPerson, int aNum)
    {
        aPerson.setName("Eric Cartman");
        aPerson.setAge(10);
        aNum = 888;
        System.out.println("Person inside modify()");
        System.out.println(aPerson.getName() + " " +
            aPerson.getAge());
        System.out.println("Number inside modify()");
        System.out.println(aNum);
    }
}
```

Modifies parameters here

James Tam

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        int num = 13;
        Person aPerson = new Person();
        ParameterExample pe = new ParameterExample();

        System.out.println("Person in main() before edit");
        System.out.println(aPerson.getName() + " " +
                           aPerson.getAge());
        System.out.println("Number inside main() before edit");
        System.out.println(num);
        System.out.println("-----");
    }
}
```

```
Person in main() before edit
none -1
Number inside main() before edit
13
```

The Driver Class (2)

```
pe.modify(aPerson,num);
System.out.println("-----");
```

```
public void modify(Person aPerson, int aNum)
{
    aPerson.setName("Eric Cartman");
    aPerson.setAge(10);
    aNum = 888;
}
```

```
Person inside modify()
Eric Cartman 10
Number inside modify()
888
```

```
System.out.println("Person in main() after edit");
System.out.println(aPerson.getName() + " " +
                   aPerson.getAge());
System.out.println("Number inside main() after edit");
System.out.println(num);
```

```
    }
}
```

```
Person in main() after edit
Eric Cartman 10
Number inside main() after edit
13
```

Previous Example: Analysis

- Why did the parameter that was passed by reference change and the simple type (passed by value) did not?

James Tam

Benefits Of Employing References

- References require a bit more complexity but provide several benefits over directly working with objects and arrays.
- Benefit 1: As you have just seen a reference contains the address of 'something' (object, array).
 - As long as the address of the object or array is retained changes made inside the method will persist.
 - Recall that functions or methods can only return zero or one things (passing out of a function after it ends).
 - Passing by reference (passing into the function just as it starts executing) allows more than one change to persist after the function has ended:
fun (reference1, reference2, reference3...etc.)

James Tam

Benefits Of Employing References (2)

- Benefit 2: If an array or object is large then it may be much more memory efficient to pass a reference instead.
- Example:
 - References are typically 32 or 64 bits in size.
 - An array or object will almost always be larger.

```
char [] array1 = new char[1000000]; // 2 MB
```

```
class SocialNetworkUser
{
    // attribute for images
    // attribute for videos
}
```

James Tam

Modifying Simple Types (Parameters)

- Only one thing to be changed: return the updated value after the method ends)
- More than one thing to be changed:
 - Pass an array (e.g., three integers must be modified in a method then pass an array of integers with 3 elements).
 - Employ a wrapper (class).



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; }
}
```

- Also Wrapper classes are also used to provide class-like capabilities (i.e., methods) to simple variable types e.g., class **Integer**

–<http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html>

–Example useful method `parseInt(String)`: converting strings to integers

```
int num = Integer.parseInt("123"); // More on this later
```

James Tam

Arrays: Parameters And Return Values

- Full example under:

`/home/233/examples/advanced/3arrayParameters`

- **Format, method call:**

–When the method is called, passing an array as a parameter and storing a return value appears no different as other types.

–Example (`list1` and `list2` are arrays)

```
list2 = ape.oneDimensional(list1);
```

James Tam

Arrays: Parameters And Return Values (2)

- **Format, method definition:**

- Use 'square brackets' to indicate that the return value or parameter is an array.

- Each dimension requires an additional square bracket.

- One dimensional:

```
public int [] oneDimensional(int [] array1)
```

- Two dimensional:

```
public char [][] twoDimensional(char [][] array1)
```

James Tam

Array Of 'Objects'

- Although referred to as an array of objects they are actually arrays of references to objects.

- Recall for arrays 2 steps are involved to create array

```
int [] array;           // Reference to array
array = new int[3];    // Creates array of integers
```

- Recall for objects 2 steps are also required to create object

```
Person jim;           // Reference to Person object
jim = new Person();   // Creates object
```

James Tam

Array Of 'Objects' (2)

- An array of objects is actually an array of references to objects.
- So 3 steps are usually required

– Two steps are still needed to create the array

```
// Step 1: create reference to array
```

```
Person [] somePeople;
```

```
// Step 2: create array
```

```
somePeople = new Person[3];
```

• In Java after these two steps each array element be null.

```
somePeople[0].setAge(10); // Null pointer exception
```

– The third step requires traversal through array elements (as needed):
create a new object and have the array element refer to that object.

• (The third step can typically be skipped for array elements that are supposed to be 'empty')

James Tam

Array Of 'Objects' (3)

– (Step 3: creating objects continued)

```
for (i = 0; i < 3; i++)
```

```
{
```

```
    // Create object, array element refers to that object
```

```
    somePeople[i] = new Person();
```

```
    // Now that array element refers to an object, a method  
    // can be called.
```

```
    somePeople[i].setAge(i);
```

```
}
```

James Tam

Array Of Objects: Example

- Location of the full example:
 - /home/233/examples/advanced/4arrayReferences/simple

James Tam

Class Person

```
public class Person
{
    private int age;

    public Person() {
        age = 0;
    }

    public int getAge() {
        return(age);
    }

    public void setAge(int anAge) {
        age = anAge;
    }
}
```

James Tam

Driver Class

```
public class Driver
{
    public static void main(String [] args) {
        Person [] somePeople; // Reference to array
        int i;
        somePeople = new Person[3]; // Create array

        for (i = 0; i < 3; i++) {
            // Create object, each element refers to a newly
            // created object
            somePeople[i] = new Person();

            somePeople[i].setAge(i);
            System.out.println("Age: " +
                               somePeople[i].getAge());
        }
    }
}
```

James Tam

Design Example

- Suppose we wanted to simulate a 2D universe in the form of a numbered grid ('World')

```
class World
{
    private [][] Tardis;
}
```

- Each cell in the grid was either an empty void or contained the object that traveled the grid ('Tardis')¹

```
class Tardis
{
}
}
```

¹ Tardis and "Doctor Who" © BBC

James Tam

General Description Of Program

- The 'world/universe' is largely empty.
- Only one cell contains the Tardis.
- The Tardis can randomly move from cell to cell in the grid.
- Each movement of Tardis uses up one unit of energy

James Tam


Designing The World

Class World


- Attributes?
- Methods?

Class Tardis



- Attributes?
- Methods?



CAUTION: STOP READING AHEAD



- JT's note: Normally you are supposed to read ahead so you are prepared for class.
- In this case you will get more out of the design exercise if you don't read ahead and see the answer beforehand.
- That will force you to actually think about the problem yourself (and hopefully get a better feel for some design issues).
- So for now skip reading the slides that follow this one up to the one that has a corresponding 'go' symbol all over it.
- After we have completed the design exercise in class you should go back and look through those slides (and the source code).



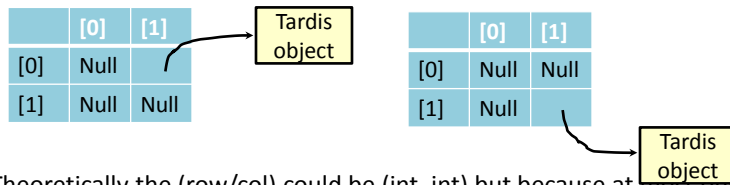
Tardis

- Attributes
 - Current energy level
- Methods:
 - Randomly generating movement:
 - Some method must reduce the energy level as the Tardis moves
 - The actual 'movement' from square to square in the grid will be a responsibility of class `World` because the grid is an attribute of the world.

World

- Attributes

- A 2D array that stores information about the 'universe'
- Most array elements will be empty (`null`)
- One element will refer to the Tardis object
- The maximum number of rows and columns
- The current location (row/column) of the Tardis
 - Needed to 'move' the Tardis from source cell to destination cell



- Theoretically the (row/col) could be (int, int) but because at most one item can be returned from a method the location will be tracked as 1D integer array (details in code):

- `World.move()` -> `Tardis.calculateCoordinates()`

James Tam

World (2)

- Methods

- Constructor(s) to create the world
- Methods that modify the world (e.g., making sure each array element is truly null: `wipe()`)
- Displaying the world: `display()`
- Changing the contents of the objects in the world (e.g., editing the world or moving objects): `move()`

James Tam

Manager

- It is responsible for things like determining how long the simulation runs.
- For very simple programs it may be a part of the `World` class (in this case it's part of the `Driver`).
- But more complex programs (e.g., need to track many pieces of information like multiple players, current scores etc. and simulation rules) may require a separate `Manager` class.
 - The `Driver` will then likely be responsible for instantiating a `Manager` object and calling some method of the manager to start the simulation.

James Tam



END SECTION: Proceed Reading



- You can continue reading ahead to the slides that follow this one.
 - JT: Thank you for your understanding and co-operation.



Source Code: Design Exercise

- Location of the full source code:
 /home/233/examples/advanced/4arrayReferences/doctor

James Tam

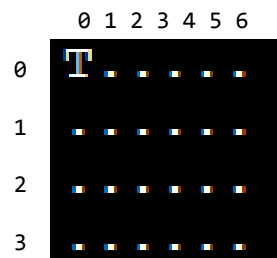
Class Tardis

```

public class Tardis
{
    private int energy;

    public Tardis(int startEnergy) {
        energy = startEnergy;
    }
    // max row and column define the size of the world
    public int[] calculateCoordinates(int maxRow, int maxColumn) {
        Random aGenerator = new Random();  e.g., = 4      e.g., = 6
        int [] newCoordinates = new int[2];
        newCoordinates[0] = aGenerator.nextInt(maxRow);      0, 1, 2, 3
        newCoordinates[1] = aGenerator.nextInt(maxColumn);
        energy--;      0, 1, 2, 3, 4, 5
        return(newCoordinates);
    }
}

```



James Tam

Class World: Attributes

```
public class World
{
    private Tardis [][] grid; // Simulated world
    private int maxRow;      // Row capacity
    private int maxColumn;  // Column capacity
    private int [] currentLocation; // (row/col) of Tardis
}
```

James Tam

Class World: Constructor

```
public World() {
    // Element 0: current row the tardis is located
    // Element 1: current column the tardis is located
    currentLocation = new int[2];

    Scanner in = new Scanner(System.in);
    System.out.print("Max rows: ");
    maxRow = in.nextInt();
    System.out.print("Max columns: ");
    maxColumn = in.nextInt();
    grid = new Tardis[maxRow][maxColumn];
    wipe(); // Empties the world, sets everything to null
    grid[0][0] = new Tardis(10); // Tardis starts top left
    currentLocation[0] = 0; // Tardis row = 0
    currentLocation[1] = 0; // Tardis col = 0
    display();
}
```

James Tam

Class World: Initialization

```

public void wipe()
{
    int r;
    int c;
    for (r = 0; r < maxRow; r++)
    {
        for (c = 0; c < maxColumn; c++)
        {
            grid[r][c] = null;
        }
    }
}

```

e.g., max = 2
 e.g., max = 3

	[0]	[1]	[2]
r = 0, c = {0,1,2}	[0] null	[1] null	[2] null
r = 1, c = {0,1,2}	[1] null	[1] null	[1] null

James Tam

Class World: Display

```

public void display()
{
    int r;
    int c;
    for (r = 0; r < maxRow; r++)
    {
        for (c = 0; c < maxColumn; c++)
        {
            if (grid[r][c] == null)
                System.out.print(".");
            else
                System.out.print("T");
        }
        System.out.println();
    }
}

```


e.g., = 4
 e.g., = 6

Move cursor to display new row on next line

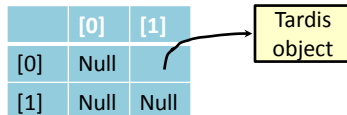
	0	1	2	3	4	5	6
0	T
1
2
3

James Tam

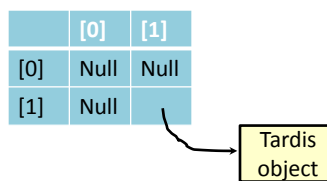
Movement

- To make it look like the Tardis has 'moved'. 
- Set the destination (row/column) to refer to the Tardis object.
- Set the source (row/column) to null

Before move



After move



James Tam

Class World: Move

```
public void move()
{
    // currentLocation 1D array stores Tardis location
    int currentRow = currentLocation[0];
    int currentColumn = currentLocation[1];

    // Keep track of where the Tardis is currently located
    int oldRow = currentRow;
    int oldColumn = currentColumn;

    // Store new (row/col) in 1D array (currentLocation)
    currentLocation =
        grid[currentRow][currentColumn].calculateCoordinates
            (maxRow,maxColumn);
```

Recall:
Tardis.currentCoordinates()
 randomly generates a new
 (row/column) location

James Tam

Class World: Move (2)

```
// Update temporary values with current location
currentRow = currentLocation[0];
currentColumn = currentLocation[1];

// Copy tardis from the old location to the new one.
grid[currentRow][currentColumn] = grid[oldRow][oldColumn];

// Check if tardis trying to move onto same square, don't
// 'wipe' if this is the case or tardis will be lost
// (Tardis object becomes a memory leak).
if ((currentRow == oldRow) &&
    (currentColumn == oldColumn)) {
    System.out.println("Same location");
}
else {
    // 'wipe' tardis off old location
    grid[oldRow][oldColumn] = null;
}
```

James Tam

Class World: Move (3)

```
System.out.println("Tardis re-materializing");
display();
}
```

James Tam

The Driver Class (Also The “Manager”)

```
public class Driver
{
    public static void main(String [] args)
    {
        Scanner in = new Scanner(System.in);
        World aWorld = new World();
        int i;
        for (i = 0; i < 10; i++)
        {
            aWorld.move();
            System.out.println("Hit enter to continue");
            in.nextLine();
        }
        System.out.println("\n<<<Tardis is out of energy,
            end simulation>>> \n");
    }
}
```

James Tam

Universally Accessible Constants

- What you currently know
 - How to declare constants that are local to a method

```
class Driver {
    main() {
        final int A_CONST = 10;
    }
}
```

- If you need constants that are accessible throughout your program then declare them as class constants.

James Tam

Declaring Class Constants

- **Format:**

```
public class <class name>
{
    public final <type> <NAME> = <value>;
}
```

- **Example:**

```
public class Person
{
    public final int MAX_AGE = 144;
}
```

- **Note:** Because constants cannot change it is okay to set the access level to public.

James Tam

Accessing Class Constants

- **Format** (outside of the class definition)¹:

```
<class name>.<constant name>;
```

- **Example** (outside of the class definition):

```
main()
{
    System.out.println("Max life span: " + Person.MAX_AGE);
}
```

- Accessing a constant inside the methods of that class do not require the name of the class

```
public class Person {
    ...
    public void fun() { System.out.println(MAX_AGE); }
}
```

James Tam

Introducing A New Concept With..Class Sheep!

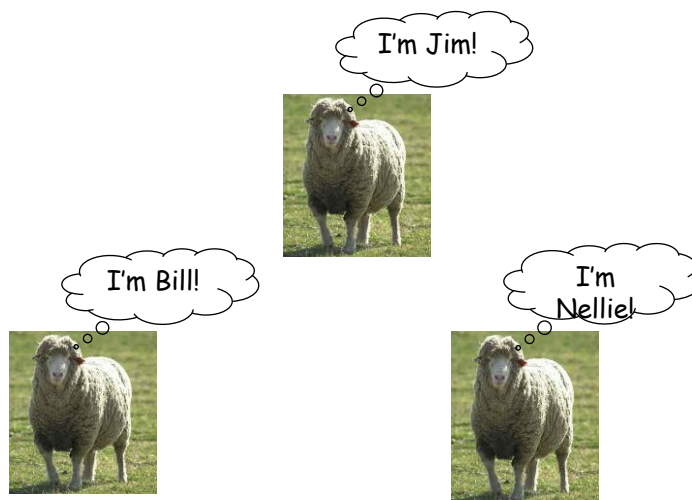
```
public class Sheep
{
    private String name;

    public Sheep()
    {
        name = "No name";
    }
    public Sheep(String aName)
    {
        setName(aName);
    }
    public String getName() { return name;}

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

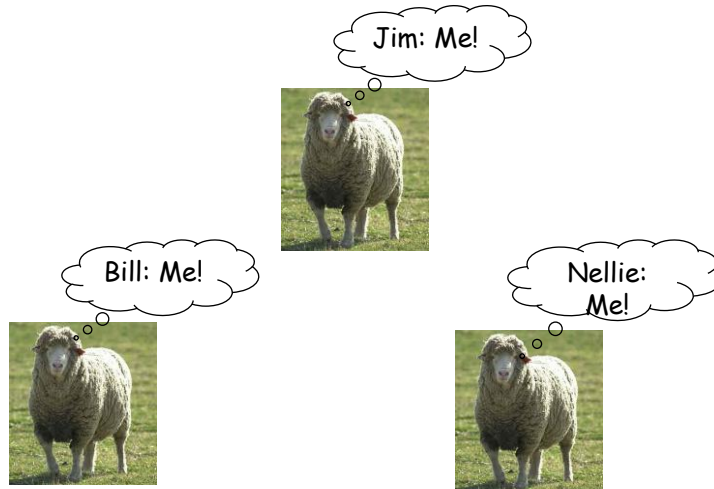
James Tam

We Create Several Sheep



James Tam

Question: Who Tracks The Size Of The Herd?

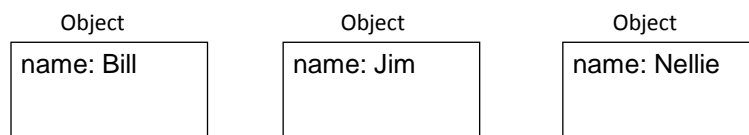


James Tam

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.

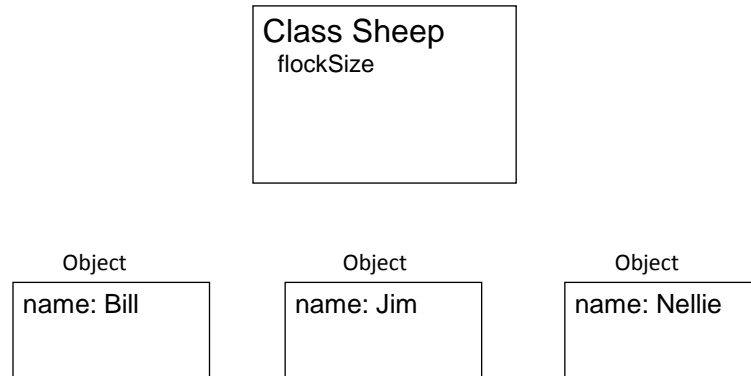
```
public class Sheep
{
    private String name;
    ...
}
```



James Tam

The Need For Static (Class Attributes)

- Static fields: One instance of the attribute exists *for the class* (not one attribute for each instance of the class)



James Tam

Static (Class) Methods

- Are associated with the class as a whole and not individual instances of the class.
 - Can be called without having an instances (because it's called through the class name not a reference/instance name).
 - Instance method:


```
Scanner in = new Scanner(System.in);
in.nextInt(); // refName.method()
```
 - Class Method:


```
double squareRoot = Math.sqrt(9); // ClassName.method()
```
- Typically implemented for classes that are never instantiated e.g., class Math.

James Tam

Accessing Static Methods/Attributes

- Inside the class definition

Format:

–*<attribute or method name>*

Example:

```
public Sheep ()  
{  
    flockSize++;  
}
```

James Tam

Accessing Static Methods/Attributes (2)

- Outside the class definition

Format:

<Class name>.<attribute or method name>

Example:

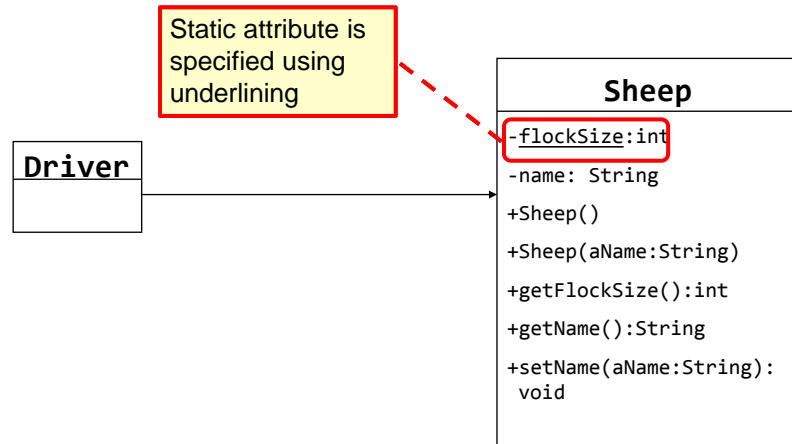
```
Sheep.getFlockSize();
```

James Tam

Static Data And Methods: UML Diagram

- Location of the online example:

~/home/233/examples/advanced/5classAttributes



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();
        System.out.println("Current count " +
            Sheep.getFlockSize());
    }
}
  
```

James Tam

Static Data And Methods: The Sheep Class

```
public class Sheep
{
    private static int flockSize = 0;
    private String name;

    public Sheep() {
        flockSize++;
        name = "No name";
    }
    public Sheep(String aName) {
        flockSize++;
        setName(aName);
    }

    public static int getFlockSize () { return flockSize; }
    public String getName() { return name;}
    public void setName(String newName) { name = newName; }
}
```

James Tam

Rules Of Thumb: Instance Vs. Class Fields

- If an attribute 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 can 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.
 - E.g., the class doesn't have any variable attributes only static constants such as class `Math`
- Otherwise the method should likely be an instance method.

James Tam

Static Vs. Final

- **Static:** Means there's one instance of the attribute for the class (not individual instances for each instance (object) of the class)
- **Final:** Means that the attribute 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 (waste) */
    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://docs.oracle.com/javase/7/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).
- Example (purely for illustration):


```
Math math1 = new Math();
Math math2 = new Math();
// What's the difference? Why bother?
math1.abs() vs. math2.abs();
```
- When in doubt *DO NOT* make attributes and methods static.

James Tam

What You Should Know: Attributes Vs. Locals

- **Attributes:** Defined inside a class definition but outside the body of a method.

```
class Person {
    private int age;
}
```

- **Locals:** Defined inside the body of a method.

```
class Person {
    public void Person(){
        Scanner in = new Scanner(System.in);
    }
}
```

James Tam

Reminder: Scope

- **Attributes**

- Declared within the body of a class but outside a method
- Accessible anywhere with the class methods

```
class Person {
    private int age;
    public Person() { age = 12; }
    ...
}
```

} Scope of attributes and methods

- **Local variables**

- Declared inside the body of a method and only accessible in that method

```
class Person {
    public Person () {
        Scanner in = new Scanner(System.in);
    }
}
```

} Scope of locals

James Tam

Scoping Rules

- When there is a reference to an identifier (e.g., variable or constant) apply the following scoping rules:

- First look for a local variable
- Then look for an attribute

SECOND:

- Look for an attribute with this name
- i.e., `private int age;`

- Example:

```
class Person {
```

```
    public setAge() {
```

```
        age = 123;
```

```
    }
```

```
}
```

FIRST:

- Look for a local variable with this name
- i.e., `int age;`

Reference to an identifier

James Tam

New Terminology: Shadowing

- There is an **attribute** and **local variable** with the same name.
- The local variable 'shadows' or hides access to the attribute (unless the 'this' reference is used – coming up later).

```
class Person
{
    private int age;
    public Person() {
        int age = 123;    // Common logic error!
    }
}
```

James Tam

Self Reference: The 'This' Reference

- From every (non-static) method of an object there exists a reference to the object (called the "this" reference) ¹

```
main(String args []) {
    Person fred = new Person();
    Person barney = new Person();
    fred.setAge(35);
}
```

This is one reason why methods must be invoked via a reference name (the contents of the reference 'fred' will be copied into the 'this' reference (so both point to the 'Fred' object).

```
public class Person {
    private int age;
    public void setAge(int anAge) {
        age = anAge;
    }
    ...
}
```

The 'this' reference is implicitly passed as a parameter to all non-static methods. One use of 'this' is to distinguish which object's method is being invoked (in this case Fred vs. Barney)

¹ Similar to the 'self' keyword of Python except that 'this' is a syntactically enforced name.

James Tam

The 'This' Reference Is Automatically Referenced Inside (Non-Static) Methods

```
public class Person {
    private int age;
    public void setAge(int anAge) {
        // These two statements are equivalent
        age = anAge;
        this.age = anAge;
    }
}
```

James Tam

New Terminology

- Explicit parameter(s): explicitly passed (you can see them when the method is called and defined).

```
fred.setAge(10);    // 10 explicit
barney.setAge(num); // num explicit
```

```
public void setAge(int age) { ... } // age explicit
```

- Implicit parameter: implicitly passed into a method (automatically passed and cannot be explicitly passed): the 'this' reference.

```
public void setAge(int age) { ... } // 'this' is implicit
```

James Tam

Benefits Of 'This': Attributes

- Another side benefit is the this reference can make it very clear which attributes are being accessed/modified.

```
public class Person
{
    private int age;

    public void setAge(int age) {
        this.age = age;
    }
}
```

Parameter (local variable) 'age'

Attribute 'age'

James Tam

Benefits Of 'This': Parameters

- Another side benefit is the `this` reference can make it clear which object is being accessed e.g., when a class method takes as an explicit parameter an instance of that class¹

```
main (String [] args) {
    Person fred = new Person();
    Person barney = new Person();
    barney.nameBestBuddy(fred);    // JT: Explicit? Implicit?
}
// JT: What will be the output?
public void nameBestBuddy(Person aPerson) {
    println(this.name + " best friend is " + aPerson.name);
}
```

¹ JT: more on this one later – see the 'equals()' method

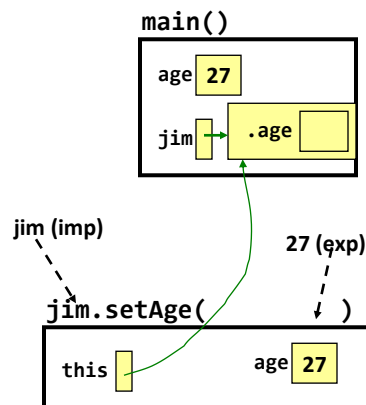
James Tam

Benefits Of 'This': Scope

- Recall: according to scoping rules, local variables are not accessible outside of that function or method (unless returned back to the caller or passed into another method).

```
main (String [] args) {
    int age = 27;
    Person jim = new Person();
    jim.setAge(age);
}
class Person {
    public void setAge(int age) {
        this.age = age;
    }
}
```

Normally the object referred to by the 'jim' reference not accessible outside of `main()` but the 'this' reference contains it's address (implicit pass by reference)



James Tam

Static Methods: No 'This' Reference

- Recall: **static methods** do not require an object to be instantiated because they are invoked via the **class name** not a reference name.

```
int result = Math.abs(-12);
```

- That means static methods do not have the implicit 'this' parameter passed in.
- Also recall I said for now avoid [for the 'Driver' class]:
 - Defining attributes for the Driver
 - Defining methods for the Driver (other than the main method)

James Tam

Static Methods: No 'This' Reference (2)

```
public class Driver
{
    private int num;
    public static void main(String [] args)
    {
        num = 123;
    }
}
```

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

- Main() must be static! Automatically called when the program runs via 'java Driver' before any other code.
- If main() were non-static it would require an object to be instantiated (which must occur inside of a method).
- But there would be no way to call that method that instantiates an object without a starting static method.
- Because main() must be static, it has no 'this' implicit parameter which in turn means that non-static attributes like 'num' cannot be accessed (although static attributes/methods accessible): `Driver.static_name` or just via `static_name`

James Tam

Mutable Vs. Immutable Types

- Mutable types

- Original memory can be modified

```
int num = 666;  
num = 777;
```

- Immutable types

- The original memory location cannot be modified
- Assigning new values will create a new memory location and leave the original untouched.

```
String s1 = "abc";  
String s2 = s1;  
s1 = "xyz";  
System.out.println(s1 + " " + s2);
```

James Tam

Mutable Vs. Immutable

- Advantage of mutable types: speed
- Advantage of immutable types: 'security'

James Tam

Mutable Advantage: Speed

- Location of full examples:
 - /home/233/examples/advanced/6mutableImmutable/speed

```
public class StringExample {
    public static void main
    (String [] args) {
        String s = "0";
        int i;
        for (i = 1; i < 100000; i++)
            s = s + i;
    }
}
```

```
public class StringBufferExample {
    public static void main
    (String [] args) {
        StringBuffer s;
        int i;
        s = new StringBuffer("0");
        for (i = 1; i < 100000; i++)
            s = s.append(i);
    }
}
```

James Tam

Immutable Advantage: Security

- Location of the full example:
 - /home/233/examples/advanced/6mutableImmutable/security

James Tam

Class SecurityExample

```
public class SecurityExample
{
    private String s;
    private StringBuffer sb;

    public SecurityExample() {
        s = new String("Original s");
        sb = new StringBuffer("Original sb");
    }

    public String getS() {
        return s;
    }

    public StringBuffer getSB() {
        return sb;
    }
}
```

James Tam

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        SecurityExample se = new SecurityExample();
        String s;
        StringBuffer sb;

        System.out.println("Originals");
        System.out.println("\t" + se.getS());
        System.out.println("\t" + se.getSB());

        s = se.getS();
        sb = se.getSB();
    }
}
```

```
Originals
Original s
Original sb
```

James Tam

The Driver Class (2)

```

sb.delete(0,sb.length());
sb.append("lolz! mucked ur data :P");
s = "lolz! mucked ur data :P";
System.out.println();

System.out.println("After modifications");
System.out.println("Values of locals");
System.out.println("\t\tString=" + s);
System.out.println("\t\tStringBuffer=" + sb);

System.out.println("\tValues of attributes");
System.out.println("\t\tString=" + se.getS());
System.out.println("\t\tStringBuffer=" + se.getSB());
}
}

```

```

Values of locals
String=lolz! mucked ur data :P
StringBuffer=lolz! mucked ur data :P

```

```

Values of attributes
String=Original s
StringBuffer=lolz! mucked ur data :P

```

James Tam

Automatic Garbage Collection Of Java References

- Dynamically allocated memory is automatically freed up when it is no longer referenced (Foo = a class) e.g., Foo f1 = new Foo();
–Foo f2 = new Foo();

References

f1(Address of a "Foo")



Dynamic memory

Object (Instance of a "Foo")



f2 (Address of a "Foo")



Object (Instance of a "Foo")



James Tam

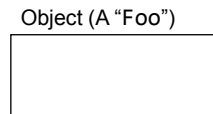
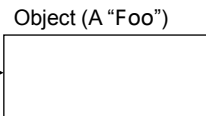
Automatic Garbage Collection Of Java References (2)

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

References



Dynamic memory



James Tam

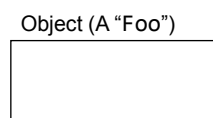
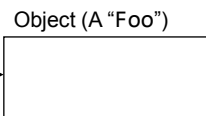
Automatic Garbage Collection Of Java References (3)

- Dynamically allocated memory is automatically freed up when it is no longer referenced e.g., `f2 = null`; (recall that a `null` reference means that the reference refers to nothing, it doesn't contain an address).

References



Dynamic memory



James Tam

Caution: Not All Languages Provide Automatic Garbage Collection!

- Some languages do not provide automatic garbage collection (e.g., C, C++, Pascal).
- In this case dynamically allocated memory must be manually freed up by the programmer.
- Memory leak: memory that has been dynamically allocated (such as via the Java 'new' keyword') but has not been freed up after it's no longer needed.
 - Memory leaks are a sign of poor programming style and can result in significant slowdowns.

James Tam

The Finalize() Method

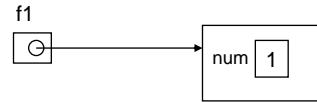
- The Java interpreter tracks what memory has been dynamically allocated via 'new'
- It also tracks when memory is no longer referenced.
- When the system isn't busy, the Automatic Garbage Collector is invoked.
- If an object has a finalize method implemented then it is invoked:
 - The finalize is a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

The Finalize() Method

- Example sequence:

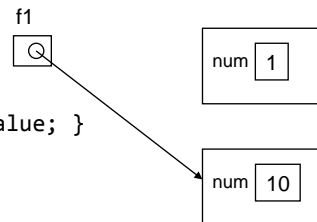
```
public class Foo
{
    int num;
    public Foo() { num = 1; }
    public Foo(int aValue) { num = aValue; }
    ...
}
...
Foo f1 = new Foo ();
```



The Finalize() Method

- Example sequence:

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

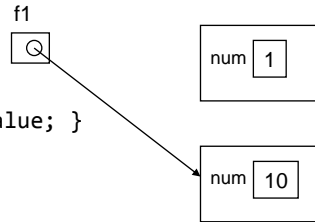


The Finalize() Method

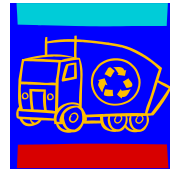
- Example sequence:

```
public class Foo
{
    int num;
    public Foo() { num = 1; }
    public Foo(int aValue) { num = aValue; }
    ...
}
```

```
Foo f1 = new Foo();
f1 = new Foo(10);
```



Don't know when

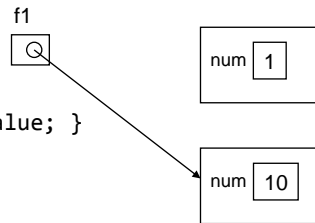


The Finalize() Method

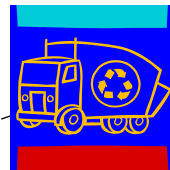
- Example sequence:

```
public class Foo
{
    int num;
    public Foo() { num = 1; }
    public Foo(int aValue) { num = aValue; }
    ...
}
```

```
Foo f1 = new Foo();
f1 = new Foo(10);
```



Don't know when



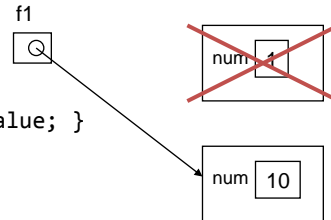
f1.finalize()

The Finalize() Method

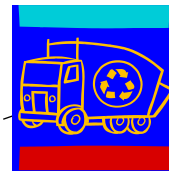
- Example sequence:

```
public class Foo
{
    int num;
    public Foo() { num = 1; }
    public Foo(int aValue) { num = aValue; }
    ...
}
```

```
Foo f1 = new Foo();
f1 = new Foo(10);
```



Don't know when



f1.finalize()

Example Application Of finalize()

- As a sheep object gets de-allocated from memory (memory is freed up because the object is no longer referenced) the `finalize()` method could update the sheep count.

```
public class Sheep
{
    private int flockSize = 0;
    public Sheep() {
        flockSize++;
    }
    ...
    public void finalize() {
        flockSize--;
    }
}
```

Displaying The Current State Of Objects

- The `toString()` method is commonly implemented to allow 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.

`toString()` Example

- Location of the full example:
 - `/home/233/examples/advanced/7toString`

Class Person

```
public class Person
{
    private int height;
    private int weight;
    private String name;

    public Person(String name, int height, int weight)
    {
        this.name = name;
        this.height = height;
        this.weight = weight;
    }
}
```

Class Person (2)

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

public int getHeight()
{
    return(height);
}

public int getWeight()
{
    return(weight);
}
```

Class Person (3)

```

public String toString()
{
    String s;
    s = "Name: " + name + "\t";
    s = s + "Height: " + height + "\t";
    s = s + "Weight: " + weight + "\t";
    return(s);
}
}

```

The Driver Class

```

public class Driver
{
    public static void main(String [] args)
    {
        Person jim = new Person("Jim",69,160);
        System.out.println("Attributes via accessors()");
        System.out.println("\t" + jim.getName() + " " +
            jim.getHeight() +
            " " + jim.getWeight());
        System.out.println("Attributes via toString()");
        System.out.println(jim);
    }
}

```

Attributes via accessors()
Jim 69 160

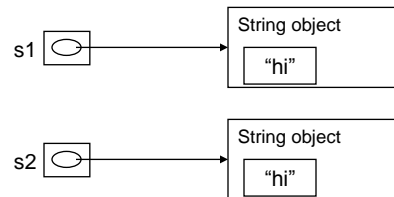
Attributes via toString()
Name: Jim Height: 69 Weight: 160

Comparing Objects

- Recall from the discussion of parameter passing (pass by reference) that a reference contains the address of an object or array.
- Using the comparison operator on the references '==' will only determine if the address (and not data) is the same.

```
String s1 = "hi";
String s2 = "hi";
```

```
if (s1 == s2)
```



Comparing Objects (2)

- Either each attribute of each object must be manually compared or else some form of `equals()` method must be implemented.
- Class `String` has two methods:
 - `compareTo()` # ABC not same as Abc
 - `compareToIgnoreCase()` # ABC same as abc

Implementing Equals()

- Location of the full example:
 - /home/233/examples/advanced/8equals

Class Person

```
public class Person {
    private int height;
    private int weight;

    public Person(int height, int weight) {
        this.height = height;
        this.weight = weight;
    }

    public int getHeight() {
        return(height);
    }

    public int getWeight() {
        return(weight);
    }
}
```

Class Person (2)

```
public void setHeight(int height) {
    this.height = height;
}

public void setWeight(int weight) {
    this.weight = weight;
}

    Implicit: Jim           Explicit: Bob
public boolean equals(Person compareTo) {
    boolean flag = true;
    if (this.height != compareTo.getHeight() ||
        this.weight != compareTo.getWeight())
        flag = false;
    return(flag);
}
}
```

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        Person jim = new Person(69,160);
        Person bob = new Person(72,175);
    }
}
```



```

new
Person(69,160);
new
Person(72,175);
System.out.println("Different data, addresses");
System.out.println("Compare data via accessors()");
if (jim.getHeight() == bob.getHeight() &&
    jim.getWeight() == bob.getWeight())
    System.out.println("\tObjects same data");
else
    System.out.println("\tNot equal");

System.out.println("Compare data via equals()");
if (jim.equals(bob) == true)
    System.out.println("\tObjects same data");
else
    System.out.println("\tNot equal");

System.out.println("Compare addresses");
if (jim == bob)
    System.out.println("\tSame address");
else
    System.out.println("\tDifferent addresses");

```

Compare data via accessors()
Not equal

Compare data via equals()
Not equal

Compare addresses
Different addresses

```

Person(72,175); # via set()
Person(72,175);
System.out.println();
System.out.println("Same data, different addresses");
jim.setHeight(72);
jim.setWeight(175);
if (jim.equals(bob) == true)
    System.out.println("\tObjects same data");
else
    System.out.println("\tNot equal");

System.out.println("Compare addresses");
if (jim == bob)
    System.out.println("\tSame address");
else
    System.out.println("\tDifferent addresses");

```

Same data, different addresses
Objects same data

Compare addresses
Different addresses

Person(72,175); # via set()

Person(72, 175);

The Driver Class (4)

```

System.out.println();
System.out.println("Same data, different addresses");
jim.setHeight(72);
jim.setWeight(175);
if (jim.equals(bob) == true)
    System.out.println("\tObjects same data");
else
    System.out.println("\tNot equal");

System.out.println("Compare addresses");
if (jim == bob)
    System.out.println("\tSame address");
else
    System.out.println("\tDifferent addresses");

```

Same data, different addresses
Objects same data

Compare addresses
Different addresses

jim = bob;

The Driver Class (5)

```

System.out.println();
System.out.println("Same addresses");
jim = bob;
if (jim == bob)
    System.out.println("\tSame address");
else
    System.out.println("\tDifferent addresses");

```

Same addresses
Same address

After This Section You Should Now Know

- References
 - How references and objects are related
 - The difference between a deep vs. shallow copy
 - How to check for if objects are identical (on a field-by-field basis and by implementing an equals() method)
 - What is the difference between comparing references vs. objects
- How the two methods of parameter passing work, what types are passed using each mechanism
- What are the benefits of employing the indirect mechanism of references-data vs. just data variables
- What is a wrapper class and what is its purpose

James Tam

After This Section You Should Now Know (2)

- How to pass arrays as parameters and return them from methods
- Arrays of 'objects'
 - Why they are really arrays of references
 - How to declare such an array, create and access elements
- How could a simple simulation be implemented using an array of references
- How to declare class constants
- Static attributes and methods
 - How to create statics
 - How to access statics
 - When something should be static vs. non-static (instance)
 - The difference between static and final

James Tam

After This Section You Should Now Know (3)

- Attributes vs. local variables
 - How is each one declared
 - What is the scope of each
 - How is each one accessed
- What are the scoping rules for locals and attributes
- What is shadowing
- Design issues
 - When should something be declared as local vs. an attribute
 - How to determine which attributes and methods should be part of which classes
- What is the 'this' reference
 - When it is and is not an implicit parameter
 - What's the difference between implicit and explicit parameters
 - What are the benefits of having a `this` parameter

James Tam

After This Section You Should Now Know (4)

- Mutable vs. immutable types
 - What is the difference
 - What is the advantage of each type
 - What is automatic garbage collection
- The `finalize()` method
 - How to define one
 - When is it called
 - What are common uses for this method
 - How is it related to automatic garbage collection
- How to display the current state of an object by implementing a `toString()` method

James Tam