

# Java Programming

Arthur Hoskey, Ph.D.  
Farmingdale State College  
Computer Systems Department

- Ch 7 Arrays – Quick Review
- Ch 8 Classes and Objects: A Deeper look  
(Most topics we have already seen)
- Ch. 9 Inheritance

## Today's Lecture

- Declare an 8 element integer array called Num:

*Arrays are reference types*

```
int[] Num = new int[8];
```

*8 element array*

*Brackets signify an array*

***Can you resize an array?***

## Ch 7 Review - Arrays

- Technically, NO. You can allocate new memory for the array though:

```
int[] Num; // Num variable
```

```
Num = new int[8]; // Allocate 8 elements
```

```
Num = new int[16]; // Allocate 16 elements
```

***What happens to the data in the original array?***

## Ch 7 Review - Arrays

# Arrays of reference types

```
class Person
{
    private String m_Name;

    public Person(String name) { m_Name = name;}

    public String GetName() { return m_Name; }
    public void SetName(String name)
    { m_Name = name; }
}
```

## Ch 7 Review - Arrays

- Create an array containing Person objects

```
// Allocate the 3 element array of Person  
Person[] group = new Person[3];
```

```
// Call new FOR EACH element of the array  
group[0] = new Person("Arthur");  
group[1] = new Person("Aidan");  
group[2] = new Person("Gareth");
```

## Ch 7 Review - Arrays

Already covered from Chapter 8:

***public/private, get/set, overloading constructors, static, this***

Need to cover:

- Finalize
- Composition

## Chapter 8 Topics

## **finalize()**

- Method called when an object is **actually garbage collected**.
- An object may become a candidate for garbage collection (unreferenced) but it will not actually have its finalize() method called until the garbage collector gets around to reclaiming that area of memory (could happen immediately or it could take a long time).

finalize() can be used for “clean up” code although it may not be the best choice due to the uncertainty of when it gets called.

*What should you put in finalize()???*

Code that would close an open file.

# finalize()



- **Composition: "Has-a" relationship.**
- ***Composition is a form of code reuse.***
- One class has other classes as members variables (we've already seen this).
- For example, a class called Company has Employee member variables.
- One class is "composed of" another class.
- Much better than copying and pasting code (copying and pasting code is NOT code reuse from an object-oriented design perspective).

## Composition

- Take a look at the following class:

```
public class Employee
{
    private int m_Id;

    public Employee(int newId)
    { m_Id = newId; }

    public int GetId()
    { return m_Id; }

    public void SetId(int newId)
    { m_Id = newId; }
}
```

## Composition

```
public class Company
{
    private Employee[] m_Staff;

    // Get/Set methods and constructor(s)
}
```

- A company “has an” employee.
- A company is made up of employees.
- A company is composed of employees.

## Composition

- Composition is good because we are using code that is already written.
- No need to duplicate something that has already been done.

# Composition

- Chapter 9 - Inheritance

# Inheritance

- What if we wanted to create a Manager class.
- A manager is also an employee so it also needs an id member.
- In addition, a manager has a secretary.
- For example...

## Inheritance

- Take a look at the following class:

```
public class Manager
{
    private int m_Id;
    private String m_SecretaryName;

    public Manager(int newId, String secName)
    { m_Id = newId; m_SecretaryName = secName; }

    // Id get/set methods should go here...

    // SecretaryName get/set methods should go here...
}
```

## Inheritance

- We had to duplicate the members of the Employee class to create the Manager class. Becomes a problem for big classes.
- It would be better if we could somehow use the employee class instead of copying all of its code.
- A manager is an employee.
- Managers are special types of employees.

## Inheritance



- What is inheritance?
- ***Inheritance is a form of code reuse.***
- Create a new class from an existing class.
- Use an existing class as a "base" for the new class.
- The new class adds on to the existing class.
- Again, this is much better than copying and pasting code (copying and pasting code is NOT code reuse from an object-oriented design perspective).

# Inheritance

- The new class should "inherit" from an existing class.
- Now we can use inheritance to create the Manager class.
- The Manager class will inherit from the Employee class.

## Inheritance

- How do we implement inheritance in Java?
- Revisit Employee and Manager classes.
- This time we will use inheritance to create the Manager class.

## Inheritance

```
public class Employee
{
    private int m_Id;

    public Employee(int newId)
    { m_Id = newId; }

    public int GetId()
    { return m_Id; }

    public void SetId(int newId)
    { m_Id = newId; }
}
```

**There are no  
changes to the  
Employee class.**

**The Manager  
class will just  
add to it *without  
changing it.***

## Inheritance

```
class Manager extends Employee
{
    private String m_SecretaryName;

    public String GetSecretaryName()
    { return m_SecretaryName; }

    public void SetSecretaryName(String newSecName)
    { m_SecretaryName = newSecName; }
}
```

**Adds to the Employee  
class.**

**Inheritance**

**m\_Id is part of the class  
because of “extends”.**

We could also extend the Employee class in a different way if we want.

For example...

**Inheritance**

```
class SalaryEmployee extends Employee
{
    private double m_YearlySalary;

    public SalaryEmployee(double newSal)
    { m_YearlySalary = newSal; }

    public double GetYearlySalary()
    { return m_YearlySalary; }

    public void SetYearlySalary(double newSal)
    { m_YearlySalary = newSal; }
}
```

## Inheritance

**Also inherits from an employee but “extends” in a different way.**

## Terminology

- The new class is the "**derived**" class.
- The class that is being inherited from is the "**base**" class.
- The "**derived**" class inherits from the "**base**" class.

## Inheritance



## More Terminology

- The existing or base class is also called the "**super**" class.
- The new or derived class is also called a "**sub**" class.

## Inheritance

1. Write down some example classes that could be derived from the following base classes:

- a. Shape
- b. Loan
- c. Employee

2. Come up with your inheritance hierarchy (base and derived classes).

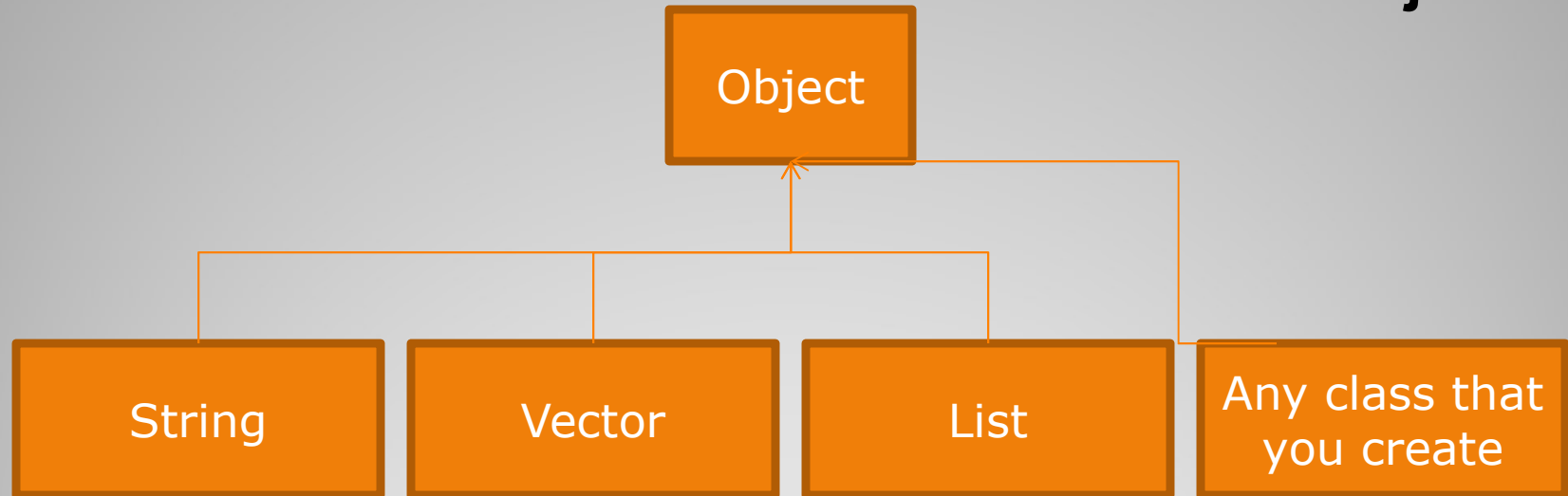
***Take attendance!!!***  
**Example Problem**

- Inheritance Examples:

Base or Superclass	Derived or Subclass
Animal	Cat, Dog, Horse, Bear, Lion
Shape	Circle, Triangle, Rectangle
Loan	CarLoan
Employee	SalaryEmployee, HourlyEmployee
Vehicle	Car, Truck

**Answers**

**All classes in Java are  
derived from Object**



**Inheritance**

- **Inheritance: "is-a" relationship**
- A derived class "is-a" type of the base class.
- **"A dog is an animal"**
- Base class are more general than derived classes.

## Inheritance

- All of the members of SalaryEmployee are also members of Employee.
- This means that we can call methods of the base class (Employee) even though we didn't define them inside the derived class (SalaryEmployee).

```
SalaryEmployee se = new SalaryEmployee(100000);  
se.SetId(23);
```

- SetId() is a member of the base class so we can call it from the derived class

## Inheritance

- Does the derived class have access to the private members of the base class?

## Inheritance

- **No.** The derived class does NOT have access to the private members of the base class.
- What if we wanted to give derived classes access to members of the base class.
- Declare the members of the base class as "**protected**".

## Inheritance



- New access modifier: **protected**.
- Protected can be accessed by derived classes

## Inheritance

```
public class Employee
{
    protected int m_Id; // Using protected not private

    public Employee(int newId)
    { m_Id = newId; }

    public int GetId()
    { return m_Id; }

    public void SetId(int newId)
    { m_Id = newId; }
}
```

## Inheritance

- What access is used on a member variable if the access modifier is missing?

**Missing Access Modifier**

- What access is used on a member variable if the access modifier is missing?

## **ANSWER**

### **Package access.**

If there is no access modifier then that member is accessible from anywhere in the package (not outside of the package).

## **Missing Access Modifier**

- How do constructors work with inheritance?
- The derived class constructor calls the base class constructor.
- Use the keyword "super" to call the base class or superclass constructor.
- For example...

## Inheritance

```
public class Employee
{
    protected int m_Id;

    public Employee(int newId) // Base class constructor
    { m_Id = newId; }          // sets id member.

    public int GetId()
    { return m_Id; }

    public void SetId(int newId)
    { m_Id = newId; }
}
```

# Inheritance

```
public Manager extends Employee
```

```
{
```

```
    private String m_SecretaryName;
```

```
    public Manager(int newId, String newSec)
```

```
    {
```

```
        super(newId); // Calls base class or superclass  
                        // constructor.
```

```
        m_SecretaryName = newSec;
```

```
    }
```

```
    // Assume other methods are declared here...
```

```
}
```

Call to the base  
class constructor  
must be the first line



**super**

```
public SalaryEmployee extends Employee
{
    private double m_YearlySalary;

    public SalaryEmployee(int newId, double newSal)
    {
        super(newId); // Calls base class or superclass
                      // constructor.
        m_YearlySalary = newSal;
    }

    // Assume other methods are declared here...
}
```

**super**



- If the call to the base class constructor is missing then an implicit call will be made to the default base class constructor BEFORE the derived class constructor runs. For example:

```
class B {  
    public B() {  
        System.out.println("B running...");  
    }  
}
```

```
class D extends B {  
    public D() {  
        System.out.println("D running...");  
    }  
}
```

```
public static void main(String[] args) { D d = new D(); }
```

Output  
B running...  
D running...

## Implicit Base Constructor Call

- Can you prevent a class from being inherited from?
- Yes.
- You must declare it as "final".
- A "final" class cannot be inherited from.

**final**

```
public final class Employee {  
    // Employee members go here...  
  
}
```

```
public class Manager extends Employee {  
    // Manager members go here...  
  
}
```

- Cannot inherit from the Employee class since its final.

**final**

- You can make classes whose sole purpose is to be inherited from.
- These are called "**abstract**" classes.
- An abstract class can hold code that has common behavior or data.
- You **cannot** make an instance of an abstract class.

**abstract**

- Note: All reference types in Java are derived from the predefined class Object (Object is the base class of all classes in Java).
- A String "is an" object.

## Inheritance

- You can place an instance of a derived type into an instance of a base type.
- You CANNOT put an instance of a base type into a derived type.
- For example...

## Inheritance

- Suppose we have an Employee and a Manager.

Employee e;  
Manager m;

// Assume that new is called for both...

e = m; // This is allowed.

m = e; // This is NOT allowed!!!

## Inheritance

Employee e;

Manager m;

// Assume that new is called for both...

e = m; // Assign Manager to Employee

- This is OK because a Manager has ALL possible functionality of an Employee.

- For example:

e.SetId(111); // Employee knows SetId()

## Inheritance



```
Employee e;  
Manager m;  
// Assume that new is called for both...
```

```
m = e;    // Assign Employee to Manager
```

- This is NOT OK because a Manager has functionality that an Employee does NOT have.
- For example:  
    m.SetSecretaryName("Jane");
- The object that m is pointing to is an Employee.  
    No SetSecretaryName() or m\_SecretaryName.

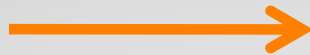
## Inheritance

e  
GetId()  
SetId()



Employee  
int m\_Id  
SetId()  
GetId()

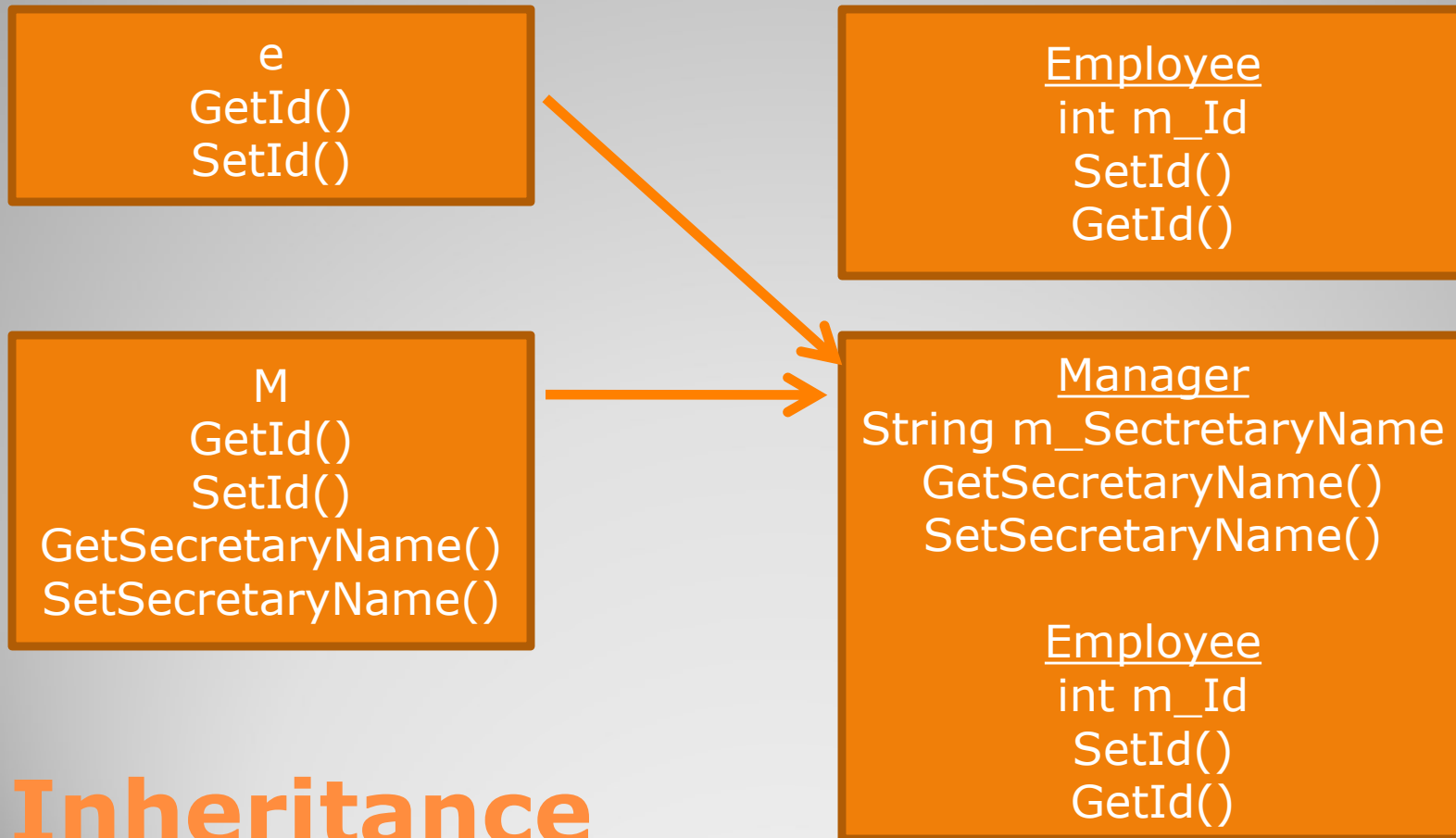
M  
GetId()  
SetId()  
GetSecretaryName()  
SetSecretaryName()



Manager  
String m\_SecretaryName  
GetSecretaryName()  
SetSecretaryName()  
  
Employee  
int m\_Id  
SetId()  
GetId()

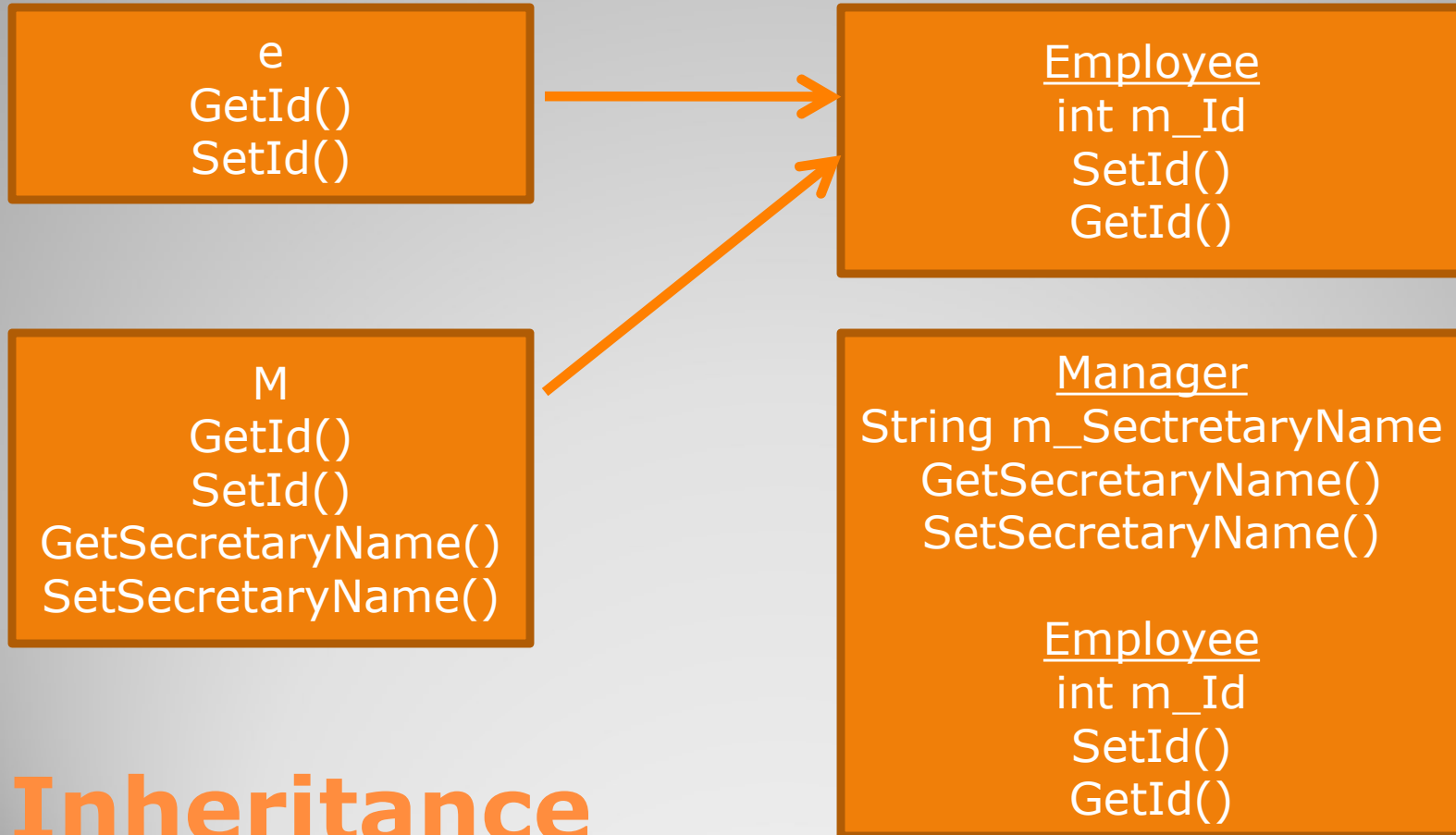
# Inheritance

- `e = m; // OK. Underlying object has GetId() and SetId()`



# Inheritance

- `m = e; // BAD. Underlying object doesn't know GetSecretaryName() etc...`



- Java collection classes are all defined to accept objects.
- Since every reference type directly or indirectly inherits from the Object class then they can all be used.
- Collections: List, Vector  
Any reference type (even user defined) can be stored in a List or Vector.

## Inheritance

- What is inheritance?
- A form of code reuse.
- Create a new class from an existing class.
- Use an existing class as a "base" for the new class.
- The new class adds on to the existing class.

## Inheritance - REVIEW