Java Programming

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

© 2023 Arthur Hoskey. All rights reserved.

General class topics already covered:

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

Need to cover:

- Finalize
- Aggregation
- Composition
- Inheritance

General Class Topics

© 2023 Arthur Hoskey. All rights reserved.

finalize()

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.

© 2023 Arthur Hoskey. All rights reserved.

Aggregation

- **Aggregation**: **"Has-a"** relationship.
- Aggregation 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 has another class as part of it ("has a" relationship).
- Much better than copying and pasting code (copying and pasting code is NOT code reuse from an object-oriented design perspective).

© 2023 Arthur Hoskey. All rights reserved.

- Take a look at the following class:

```
public class Student {
        private int m_Id;

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

        public int GetId()
        { return m_Id; }

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

Aggregation

© 2023 Arthur Hoskey. All rights reserved.

```
public class Course
{
        private Student[] m_Students;

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

Aggregation
Course has a Student as a member variable

- A course "has a" student.
- A course is made up of students.

Aggregation

© 2023 Arthur Hoskey. All rights reserved.

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

Aggregation

© 2023 Arthur Hoskey. All rights reserved.

Composition

- **Composition**: **"Has-a"** relationship.
- **Composition is a form of code reuse.**
- Composition is a special form of aggregation.
- With composition, if the containing object is destroyed then the contained object is also destroyed.
- For example, think about the classes building and room. A building "has a" room.
- Building and room is composition because if the building is destroyed, then the room is also destroyed.

© 2023 Arthur Hoskey. All rights reserved.

Aggregation vs Composition

- Aggregation Example: Course "has a" student. It is not composition because if the course is destroyed the student still exists. The existence of the student is not dependent on the course existing.
- Composition Example: Building "has a" room. The room only exists if the building exists.

© 2023 Arthur Hoskey. All rights reserved.

- Now on to inheritance…

Inheritance

© 2023 Arthur Hoskey. All rights reserved.

Employee Class

- Employee will be used in following examples:

```
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; }
}
```

© 2023 Arthur Hoskey. All rights reserved.

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…

© 2023 Arthur Hoskey. All rights reserved.

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…
}
```

© 2023 Arthur Hoskey. All rights reserved.

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.

© 2023 Arthur Hoskey. All rights reserved.

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).

© 2023 Arthur Hoskey. All rights reserved.

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.

© 2023 Arthur Hoskey. All rights reserved.

Inheritance

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

© 2023 Arthur Hoskey. All rights reserved.

```
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

© 2023 Arthur Hoskey. All rights reserved.

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

        public String GetSecretaryName()
        { return m_SecretaryName; }

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

Only allowed to directly inherit from 1 class

Adds to the Employee class.

m_Id is part of the class because of “extends”.

Inheritance

© 2023 Arthur Hoskey. All rights reserved.

Inheritance

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

For example…

© 2023 Arthur Hoskey. All rights reserved.

```
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; }
}
```

Also inherits from Employee but "extends" in a different way.

Inheritance

© 2023 Arthur Hoskey. All rights reserved.

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

© 2023 Arthur Hoskey. All rights reserved.

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

© 2023 Arthur Hoskey. All rights reserved.

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

© 2023 Arthur Hoskey. All rights reserved.

- Inheritance Examples:

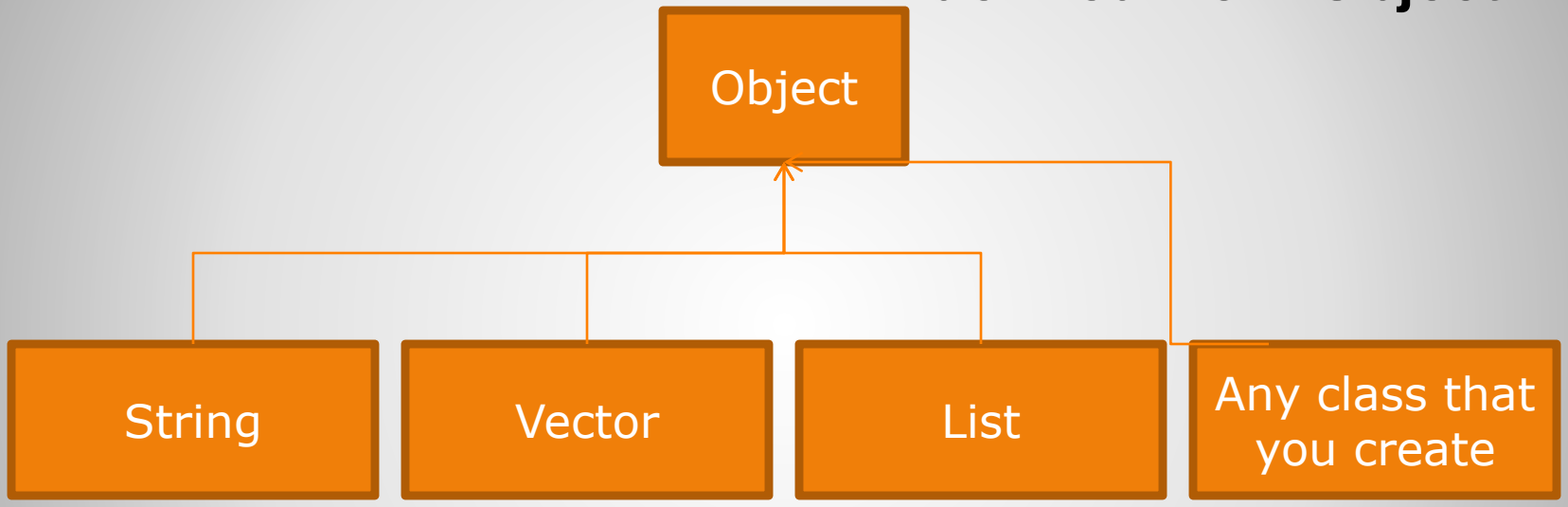
Base or Superclass	Derived or Subclass
Animal	Cat, Dog, Horse, Bear, Lion
Shape	Circle, Triangle, Rectangle
Loan	CarLoan
Employee	SalaryEmployee, HourlyEmployee
Vehicle	Car, Truck

Answers

© 2023 Arthur Hoskey. All rights reserved.

Inheritance

All classes in Java are derived from Object



© 2023 Arthur Hoskey. All rights reserved.

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.

© 2023 Arthur Hoskey. All rights reserved.

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

© 2023 Arthur Hoskey. All rights reserved.

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

Inheritance

© 2023 Arthur Hoskey. All rights reserved.

- **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

© 2023 Arthur Hoskey. All rights reserved.

Inheritance

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

© 2023 Arthur Hoskey. All rights reserved.

```
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

© 2023 Arthur Hoskey. All rights reserved.

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

Missing Access Modifier

© 2023 Arthur Hoskey. All rights reserved.

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).

© 2023 Arthur Hoskey. All rights reserved.

Inheritance

- 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…

© 2023 Arthur Hoskey. All rights reserved.

```
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

© 2023 Arthur Hoskey. All rights reserved.

```
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

© 2023 Arthur Hoskey. All rights reserved.

```
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

© 2023 Arthur Hoskey. All rights reserved.

Implict Base Constructor Call

- 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…

© 2023 Arthur Hoskey. All rights reserved.

Inheritance

- 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.

© 2023 Arthur Hoskey. All rights reserved.

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…

© 2023 Arthur Hoskey. All rights reserved.

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!!!
```

© 2023 Arthur Hoskey. All rights reserved.

```
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

© 2023 Arthur Hoskey. All rights reserved.

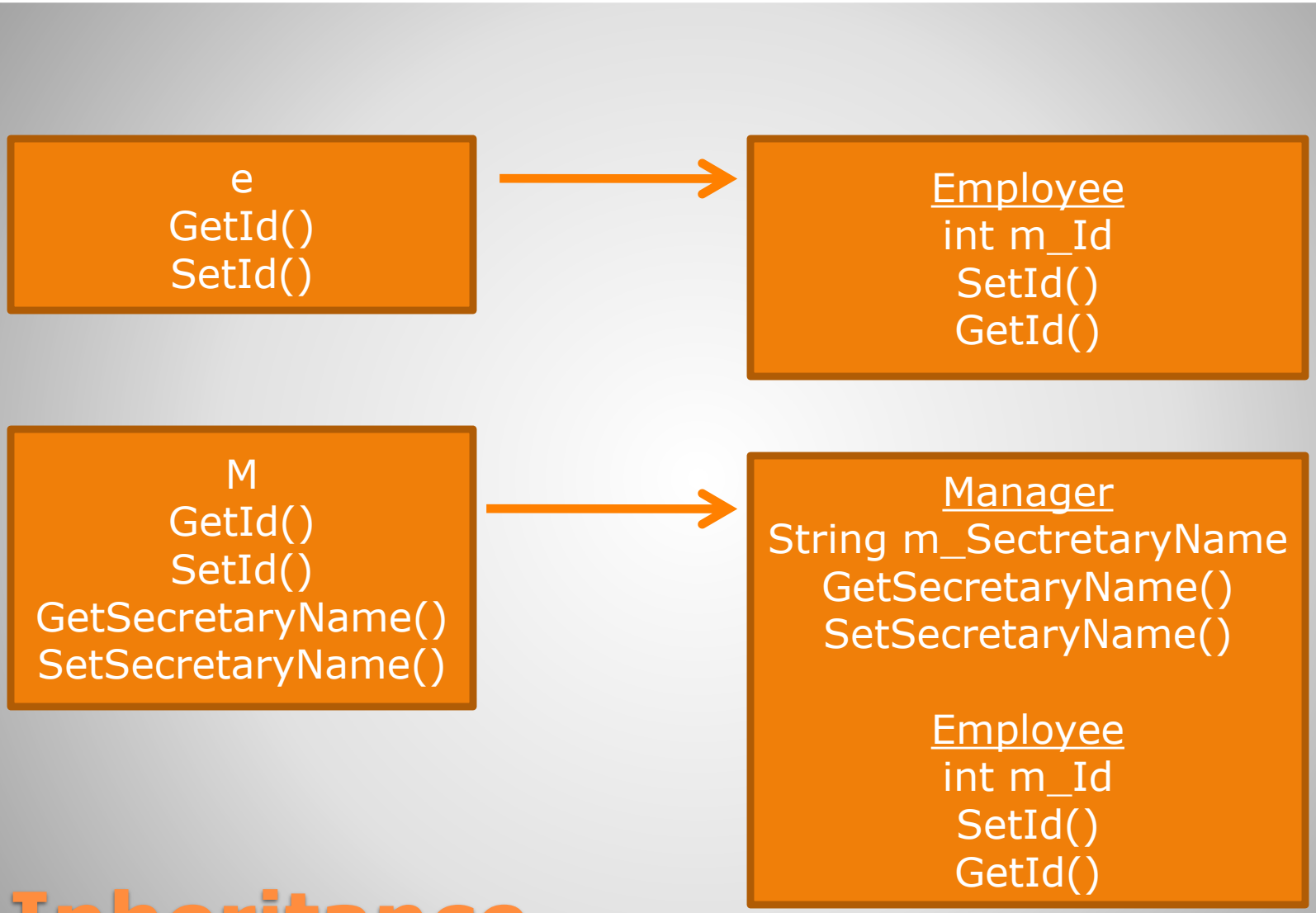
```
Employee e;
Manager m;
// Assume that new is called for both…

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 SetSectretaryName() or m_SectretaryName.

Inheritance

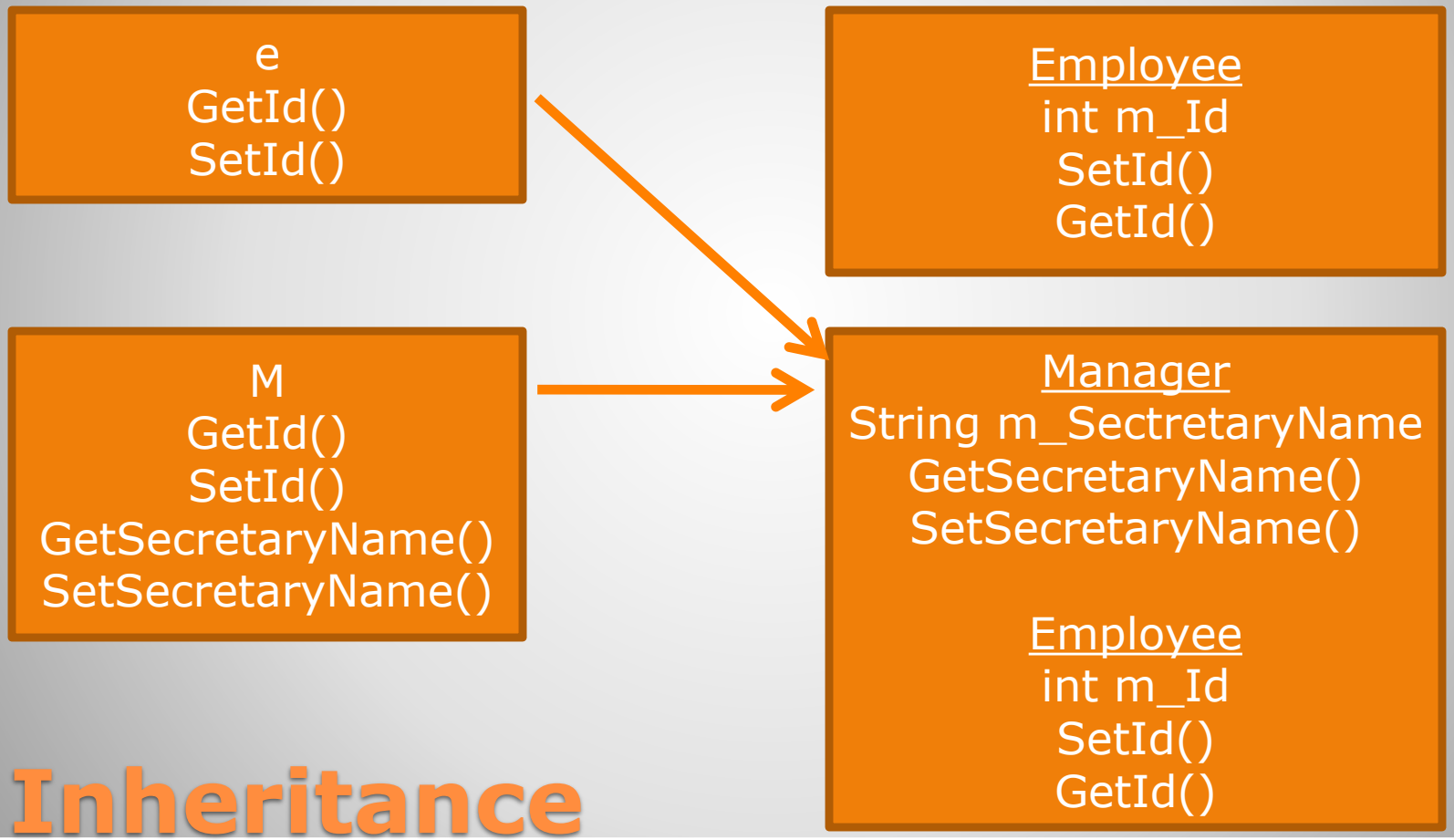
© 2023 Arthur Hoskey. All rights reserved.



Inheritance

© 2023 Arthur Hoskey. All rights reserved.

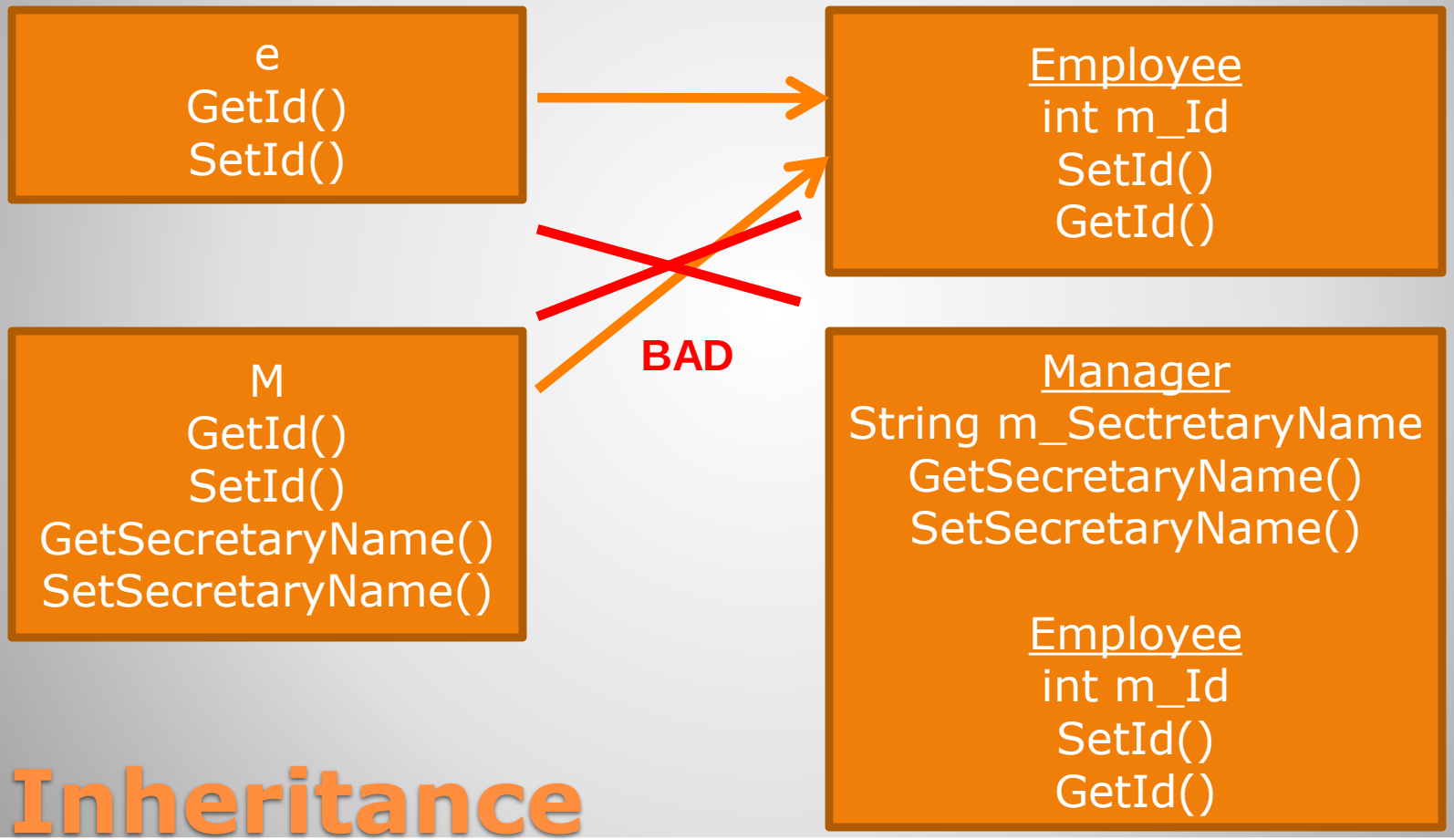
- e = m; // OK. Underlying object has GetId() and SetId()



Inheritance

© 2023 Arthur Hoskey. All rights reserved.

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



Inheritance

© 2023 Arthur Hoskey. All rights reserved.

Inheritance

- 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.

© 2023 Arthur Hoskey. All rights reserved.

Inheritance - REVIEW

- 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.

© 2023 Arthur Hoskey. All rights reserved.

- End of Slides

End of Slides

© 2023 Arthur Hoskey. All rights reserved.