

Inheritance

Ch 6

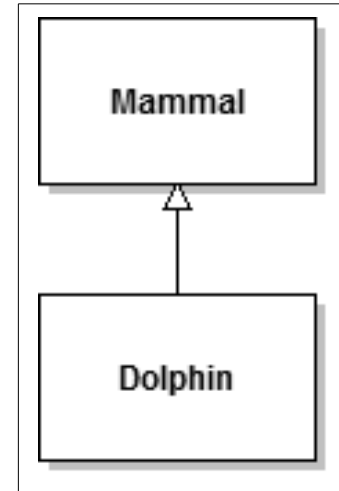
Topics

- 1) How can Java work with class inheritance?
 - 1) Creating subclasses
 - 2) Accessing the base class
 - 3) Overriding methods
 - 4) Class hierarchies
 - 5) Visibility

Creating Subclasses

Inheritance

- Inheritance:
 - Ex: A dolphin is-a mammal.
 - Dolphin inherits from mammal
(subclass) (superclass)
(derived) (base)
- Motivation:
 - Share code between base class and derived class.
 - Properties of the base are inherited by the derived.
 - ..



Book Inheritance Example

Client Code:

```
Dictionary web = new Dictionary();  
web.setPages(25);  
web.setDefinitions(2523);  
double r = web.computeRatio();
```

- Don't re-implement (or copy-and-paste) the code from Book into Dictionary.
- Makes maintaining shared Book-functionality easier.
 - Why?..

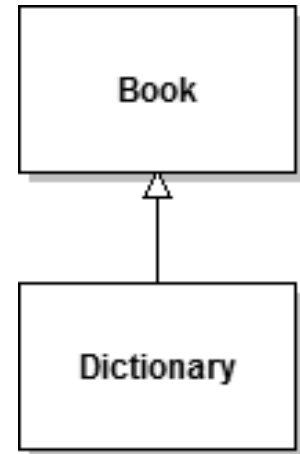
Book
- pages : int
+ setPages(numPages : int) : void + getPages() : int



Dictionary
- definitions : int
+ computeRatio() : double + setDefinitions(numDefinitions : int) : void + getDefinitions() : int

Notes on Inheritance Example

- Instantiating Dictionary does not..
 - Dictionary object has all members from:
 - the Book class (its superclass), and
 - the Dictionary class
- Access:
 - Subclass may call/access.. of super class.
 - Ex: Dictionary code can call public functions in Book.
 - Base class cannot access members of derived class.



Polymorphism via Class Inheritance

- Polymorphic references can refer to a class, or any derived class:

```
Phone x;
```

```
x = new Phone();
```

```
// Reference to derived class
```

```
CellPhone cell = new CellPhone();
```

```
x = new CellPhone();
```

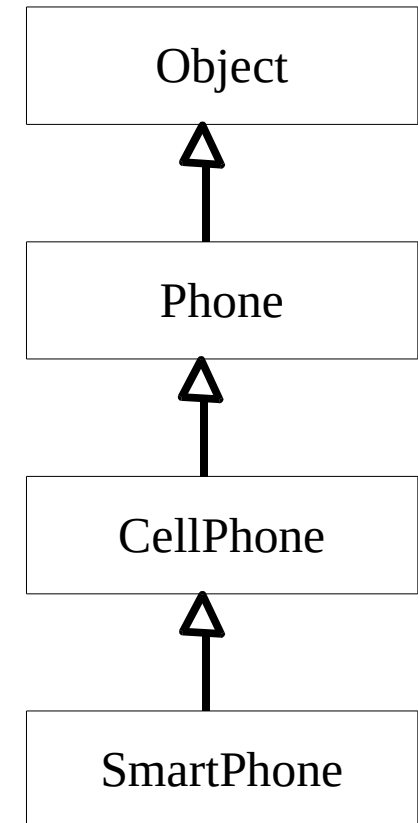
```
// Reference to derived-derived class
```

```
SmartPhone smart = new SmartPhone();
```

```
x = new SmartPhone();
```

```
// Cannot reference a base class..
```

```
SmartPhone oops = new Phone();
```



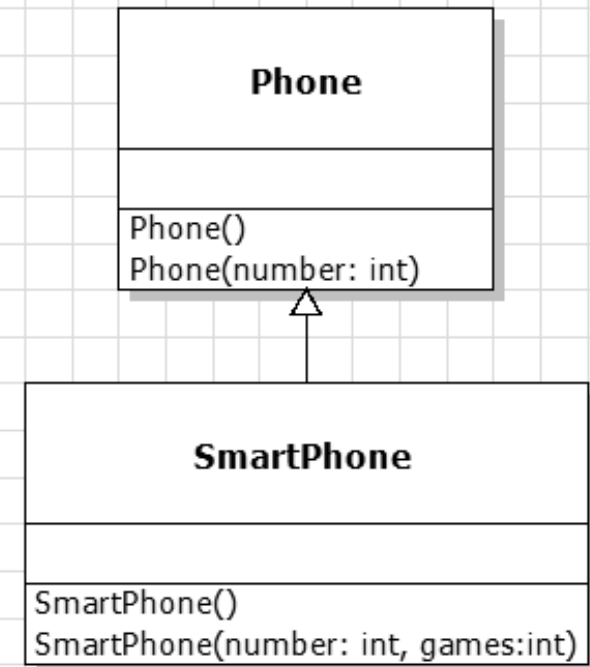
Overriding Methods

(Not over**loading**, over**riding**)

super

- `super`: refers to..
- `this`: refers to current object, not superclass.
- Subclass's constructor can “call” superclass constructor:

```
public class SmartPhone extends Phone {  
    int numGames = 0;  
    public SmartPhone () {        ..  
        super();  
    }  
    public SmartPhone (int number, int games) {  
        super(number);  
        numGames = games;    ..  
    }  
}
```



super Notes

- `super()` must be the..
 - If missing, `super();` automatically added as first line (unless using constructor chaining via `this(...)`)
- Constructor Chaining
 - Each subclass calls its superclass's constructor.
 - Creates a chain of constructor calls.
 - Ensures base-classes are..

(Except if base class calls a method which is overridden in derived class.)
 - Can chain to constructors of current class using `this()`

Chaining Constructors

- Ex: Chain constructors in current class, or super class.

```
public class Base {  
    int count = 0;  
  
    public Base() {  
        this(5);  
        // Do anything...  
    }  
  
    public Base(int count) {  
        this.count = count;  
        // Do anything...  
    }  
}
```

```
public class Derived extends Base {  
    private final double DEFAULT = 42.0;  
    private double other;  
  
    public Derived(int count) {  
        this(count, DEFAULT);  
        // Do anything...  
    }  
  
    public Derived(int count, double other)  
    {  
        super(count);  
        this.other = other;  
        // Do anything...  
    }  
}
```

= DerivedConstructor

Overriding

- Subclass can override a method of superclass if same signature as base:
 - Same name
 - Same argument # and types

```
public static void main(String[] args) {  
    Fruit apple = new Fruit("Apple");  
    System.out.println(apple.getType());  
  
    Fruit deluxe = new DeluxeFruit("Apple");  
    System.out.println(deluxe.getType());  
}
```

```
Class: class ca.cmpt213.fruit.Fruit  
Type: Apple  
Class: class ca.cmpt213.fruit.DeluxeFruit  
Type: Deluxe Apple
```

```
public class Fruit {  
    private String type;  
    public Fruit(String type) {  
        this.type = type;  
    }  
    public String getType() {  
        return type;  
    }  
}
```

```
public class DeluxeFruit extends Fruit {  
    public DeluxeFruit(String type) {  
        super(type);  
    }  
  
    @Override  
    public String getType() {  
        return "Deluxe " + super.getType();  
    }  
}
```

Overriding Details

- To override a method, derived class's method must:
 - Have identical signature
 - Not throw any extra checked exceptions (more later)
 - ..
 - Ex: Can go from protected to public, but not public to protected/private.
 - Cannot override a private, a static, or a final method.
 - Not change return type of method.
 - But you can return a subtype of original return type

final vs Overriding

- final method:..
 - In superclass:

```
public final String MCHammerSays() {  
    return "Can't touch this.";  
}
```
 - In subclass:

```
public String MCHammerSays() {  
    return "Who's MC Hammer?";  
}    ..
```
- final class:..



Shadow Variables - a Bad Idea

- Shadow Variables:
 - Subclass declares a variable of the..

```
public class Pet {  
    private String name;  
    // ...  
}  
public class PetRock extends Pet  
{  
    private String name;  
    // ...  
}
```

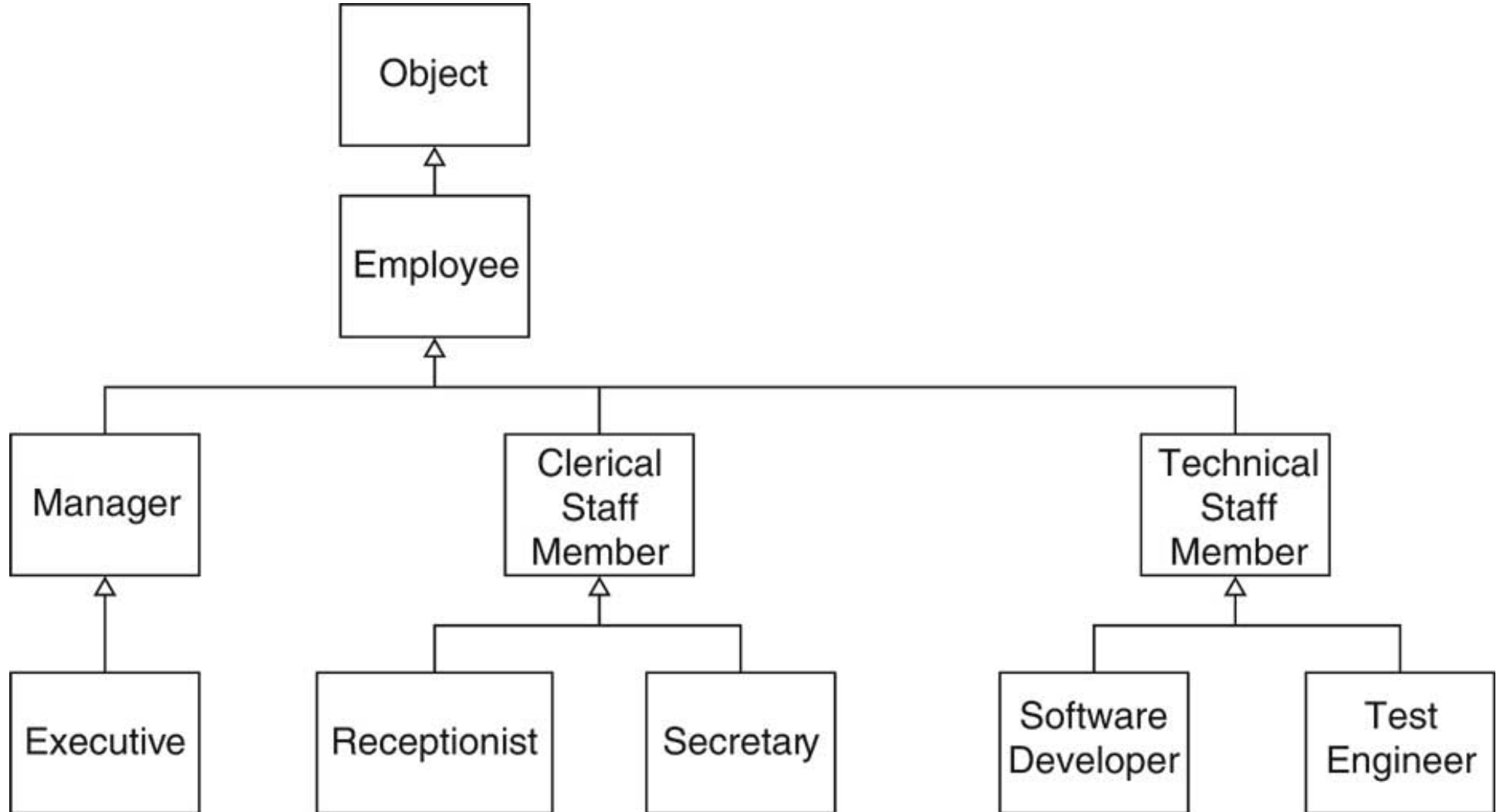
- ..
only creates confusion for programmers!
 - No good reason to use a shadow variable.
 - Pick good, unique names!

Class Hierarchies

Multiple Inheritance

- Single Inheritance:
A class may inherit from..
 - Ex: A Car is a Vehicle.
 - Java uses this approach.
- Multiple Inheritance:
A class may inherit from many superclasses.
 - Ex: A TA is both a Student and a Teacher.
 - ..
 - Impossible in Java (specifically forbidden).
- Use.. to get some benefits of multiple inheritance using only single inheritance.

Inheritance Hierarchy



Object

- All Java classes ultimately derive from the Object class.
 - If a class does not extend another a class,..
 - If a class extends some other class, its superclass must ultimately derive from Object.
- Object's public methods are inherited by all classes.
 - `boolean equals(Object obj)` `// Is this same as obj`
 - `String toString()` `// Express as a string.`
 - `Object clone()` `// Return a copy of this obj.`
 - `int hashCode()` `// For hashing collections`
- Object has an implements for each, but a class may.. with a more meaningful implementation.

Abstract Class

Abstract Classes

- Abstract class: (basic idea)
 - Un-implemented method.
Concrete derived classes must..
 - Classes with abstract methods must be abstract.
 - Abstract class cannot be instantiated:
it's incomplete; not concrete.
- Make a class abstract:
`public abstract class Plant { ... }`
- Make a method abstract:
`public abstract void doSomethingAmazing();`

Abstract Class Example

```
abstract class GraphicObject {  
    int x, y;                ..  
    ...  
    void moveTo(int newX, int newY) {  
        ...  
    }  
    abstract void draw();  
    abstract void resize();  ..  
}
```

Abstract class...

Abstract method has no
implementation.

```
class Circle extends GraphicObject {  
    @Override  
    void draw() {  
        ...  
    }  
    @Override  
    void resize() {  
        ...  
    }  
}
```

draw() and resize() must be..

Abstract Class vs Interface

Similarities

Abstract class:

- Force derived concrete class to..
- Supports constants

Java interfaces:

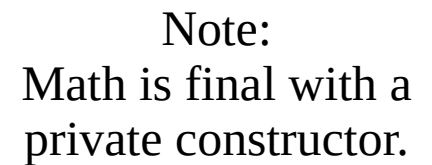
Differences

- (non-abstract)
- (non-constant fields)
- Extend classes
- In UML, abstract classes shown in *italics*.
 - Sometimes decorated with {abstract}
- Class can implement..

In Java 8, interfaces can have default (“defender”) methods, but these can only call other methods of the interface.

Abstract Questions

- Can a method be both abstract and final?
 -
- Can an abstract class have a static method?
 -
- Can a method be both abstract and static?
 -
- Can a class be both final and abstract?
 -



Note:
Math is final with a
private constructor.

Visibility

Indirect Access to Private Base Members

- Subclass **cannot** access superclass's private members.
- Can access a non-private method of the superclass, which..

```
public class Parent {  
    private int amountWine = 100;  
    protected void homeAlone() {  
        drinkWine();    // Call a private method.  
    }  
  
    private void drinkWine() {  
        amountWine--;  
    }  
}  
  
class Child extends Parent {  
    public void goodTimes() {  
        homeAlone();    //..  
        drinkWine();    //..  
    }  
}
```

protected

- protected
 - allows..
Crates a “protected” interface.
 - unrelated classes cannot access the protected members.
- Not a great idea:
 - you have no control over which classes extend your class in the future.
 - Create a “protected” interface to expose just those things that only derived classes will need (“template method”) Often better to use public interface.

Class Member Visibility

- Visibility Modifies and member accessibility:
 - public: anywhere
 - protected: in the class, package, and derived classes
 - default:
 - default is without any modifiers; called package-private
 - private:

	Inside Own Class	Inside Same Package	Inside Inherited Classes	Rest of the world
public	Visible	Visible	Visible	Visible
protected	Visible	Visible	Visible	
<i>“default” no modifier</i>	Visible	Visible		
private	Visible			

Summary

- Inheritance (is-a) used to create subclasses
- Child uses super in constructor
- Child overrides methods of parents to change behaviour
- Class hierarchies all start from Object, and each class may have at most one parent.
- Visibility modifiers affect inheritance