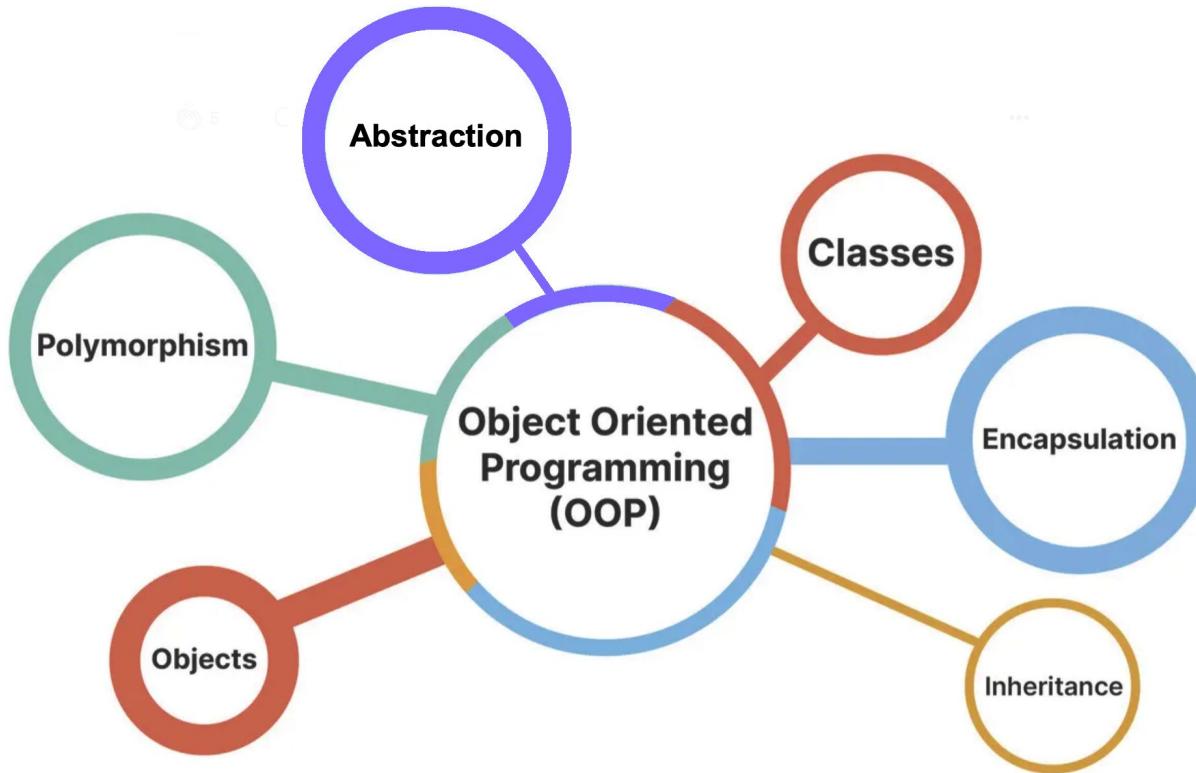


COSC 121: Computer Programming II



Today's Key Concepts

- Inheritance models a new IS-A relationship
- Implementation techniques:
 - Relationship created via `extends`
 - New visibility modifier: `protected`
 - Reference to parent object called `super`
 - The `final` modifier
 - Method `overriding`
- Adherence to encapsulation principles
- The `Object` class
- A class cannot inherit from more than one class



import vs. extends

- Difference between importing a class (e.g., Scanner class) and extending a class?
- Imported class:
 - Lets you use someone else's code
- Extended class:
 - Lets you use someone else's code

import vs. extends

- Difference between importing a class (e.g., Scanner class) and extending a class?
- Imported class:
 - Lets you use someone else's code
 - Can access info that was declared public
- Extended class:
 - Lets you use someone else's code
 - Can access info that was declared protected/public
 - Can modify it to fit your needs

The **final** modifier

- Different application contexts:
 - A `final` local variable is a **constant** as its value can never change
- Ex: `final int maxChairs = 100; // recall from prereq course`

The `final` modifier

- Different application contexts:
 - A `final` local variable is a **constant** as its value can never change
Ex: `final int maxChairs = 100; // recall from prereq course`
 - A `final` method cannot be overridden by its subclasses
Ex: `final void printReceipt() { ... }`

The `final` modifier

- Different application contexts:
 - A `final` local variable is a **constant** as its value can never change
Ex: `final int maxChairs = 100; // recall from prereq course`
 - A `final` method cannot be overridden by its subclasses
Ex: `final void printReceipt() { ... }`
 - A `final` class cannot be extended (useful when creating libraries)
Ex: `final class Math { ... }`



iClicker Question

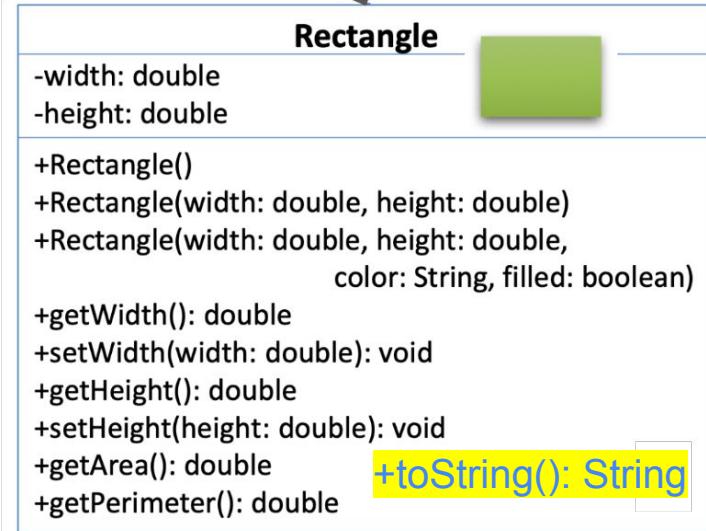
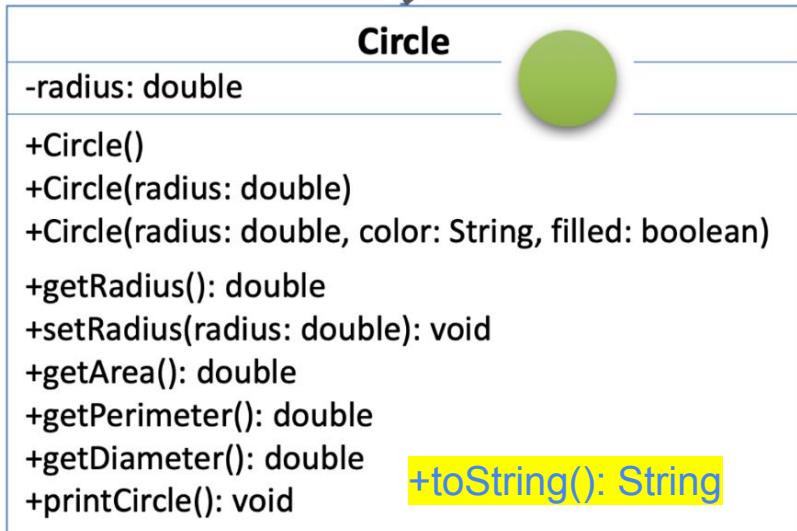
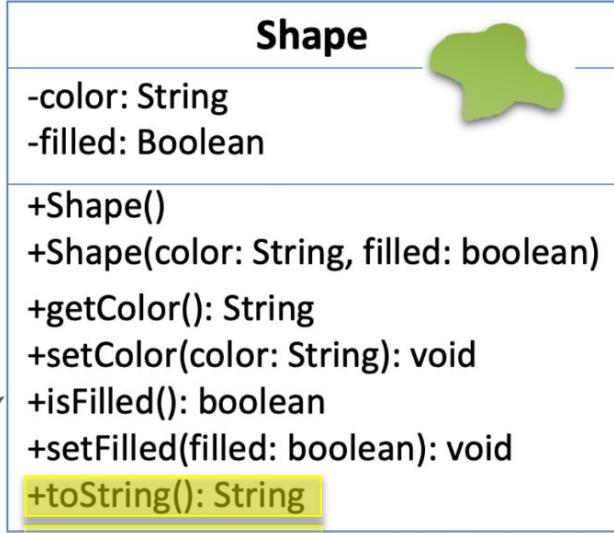
Which of the following is FALSE?

- A. final class cannot be inherited
- B. final method can be inherited
- C. final method can be overridden
- D. final variable cannot be changed

Overriding Methods

- Sometimes, parent's method is not exactly what you want for the child class
- A child class can **override** the definition of an inherited method
- Both the new method and the parent method must have:
 - The same **signature** (name and input parameters), and
 - The same return type (or subtype)
- Remember: Cannot override a **final** method

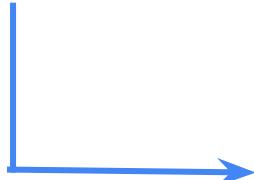
Recall Example



Overriding `toString()` in Circle

```
public class Shape {  
    private String color;  
    private boolean filled;  
    ...  
    public String toString() {  
        return "Color: " + color + ". Filled: " + filled;  
    }  
}
```

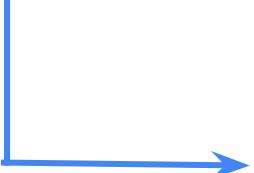
```
public class Circle extends Shape{  
    private double radius;  
    ...  
    public String toString() {  
        return "Color:" + getColor() + ". Filled: " + isFilled() +  
               ". Radius: " + radius;  
    }  
}
```



Overriding `toString()` in Rectangle

```
public class Shape {  
    private String color;  
    private boolean filled;  
    ...  
    public String toString() {  
        return "Color: " + color + ". Filled: " + filled;  
    }  
}
```

```
public class Rectangle extends Shape{  
    private double width, height;
```



```
public String toString() {  
    return "Color:" + getColor() + ". Filled: " + isFilled() +  
           ". Width: " + width + "Height: " + height;  
}
```

Example: Thought and Advice

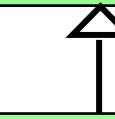
```
public class Thought {  
    public void message() {  
        System.out.println ("I feel diagonally parked");  
    }  
}  
  
public class Advice extends Thought {  
    public void message() {  
        System.out.println( "Dates are closer than they seem" );  
        super.message();  
    }  
}
```

no attributes!
no constructor!

```
graph TD; Thought[Thought] --> Advice[Advice]
```

A UML class diagram illustrating inheritance. A green rounded rectangle labeled 'Thought' has a solid black arrow pointing upwards to another green rounded rectangle labeled 'Advice'. This visualizes 'Advice' as a subclass of 'Thought'.

Thought



Advice

Example: Test Class Messages

```
public class Messages {  
    public static void main( String[] args ) {  
        Thought parked = new Thought();  
        Advice dates = new Advice();  
  
        parked.message();           // which method definition does Java call?  
        dates.message();          // which method definition does Java call?  
    }  
}
```

Thought



Advice

Example: Test Class Messages

```
public class Messages {  
    public static void main( String[] args ) {  
        Thought parked = new Thought();  
        Advice dates = new Advice();  
  
        parked.message();           // which method definition does Java call?  
        dates.message();          // which method definition does Java call?  
    }  
}
```

Output:

I feel diagonally parked

Dates are closer than they seem

I feel diagonally parked

from Thought →

from Advice →

from Advice →

(via super)

Overriding vs. Overloading

- Recall: **Overloading** allows for **flexibility** (multiple ways to call a method)
 - Ex: `public int add(int i, int j)`
`public int add(int i, int j, int k)`
- Overriding allows class to **adapt definition** of a parent method
 - Ex: In parent:
`public int mystery(int i, int j) { return i + j; }`
 - In child:
`public int mystery(int i, int j) { return i * j; }`
 - Both methods must have the same signature and return type



iClicker Question

Which example uses overriding?

- A. Left
- B. Right
- C. Neither
- D. Both

```
public class Test {  
    public static void main(String[] args) {  
        A a = new A();  
        a.p(10);  
        a.p(10.0);  
    }  
}
```

```
class B {  
    public void p(double i) {  
        System.out.println(i * 2);  
    }  
}
```

```
class A extends B {  
    public void p(double i) {  
        System.out.println(i);  
    }  
}
```

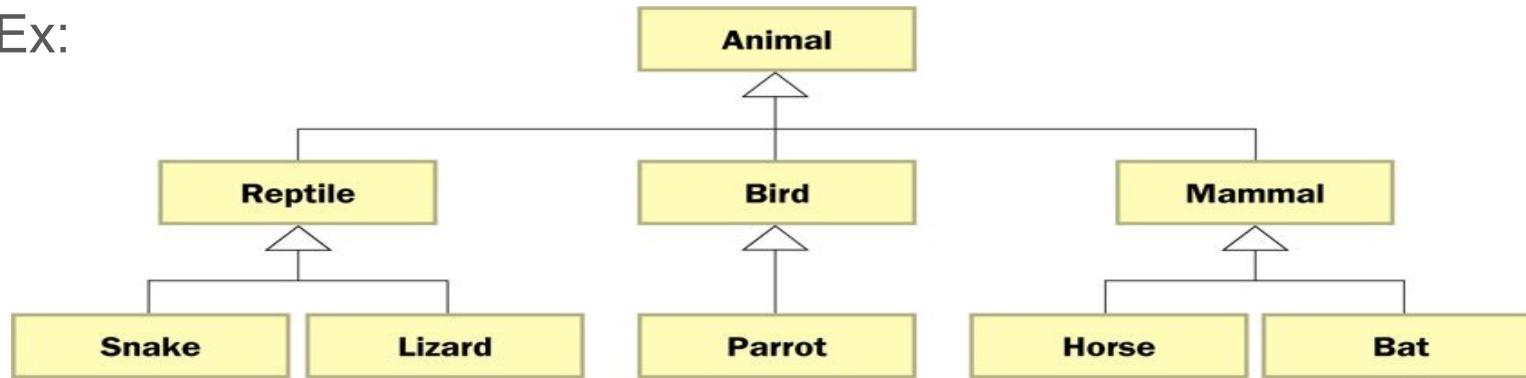
```
public class Test {  
    public static void main(String[] args) {  
        A a = new A();  
        a.p(10);  
        a.p(10.0);  
    }  
}
```

```
class B {  
    public void p(double i) {  
        System.out.println(i * 2);  
    }  
}
```

```
class A extends B {  
    public void p(int i) {  
        System.out.println(i);  
    }  
}
```

Class Hierarchy

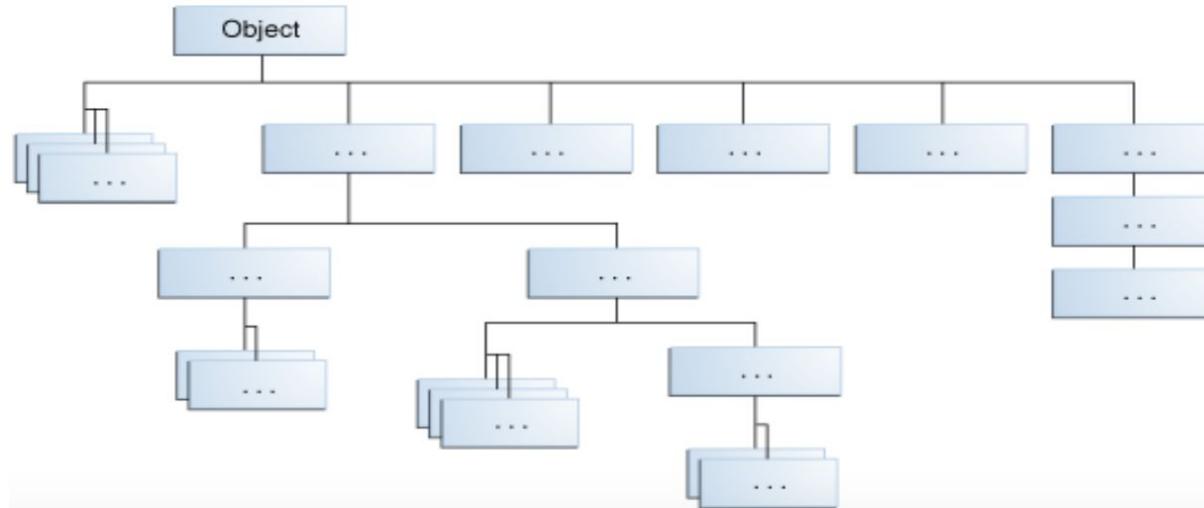
- Class hierarchy is the set of classes related through inheritance
- Ex:



- Common properties (attributes/methods) should be defined as high up in the hierarchy as is reasonable
- Inherited member is passed continually down the line
- Ex:: Horse is also an Animal

The Object Class

- In `java.lang` (always implicitly imported), there is a class called **Object**
- All classes inherit from the Object class automatically
- Even basic class definitions without `extends` are children of Object
- Object is the root of all class hierarchies



Methods in Object

- All children of Object have the following:
 - `toString()`
 - When you define `toString()` in your classes, you are overriding the inherited definition!
 - If you don't define `toString()` in your own class, you can still call the inherited definition (although hard to read)

Methods in Object

- All children of Object have the following:
 - **toString()**
 - When you define `toString()` in your classes, you are overriding the inherited definition!
 - If you don't define `toString()` in your own class, you can still call the inherited definition (although hard to read)
 - **equals()**
 - Inherited definition returns true if two `references` are aliases of the same object
 - String class overrides `equals()` by checking if two String objects have the same characters instead

Name Example

- Suppose you have defined (which also inherits equals() implicitly):

```
public class Name {  
    private String title;  
    private String firstName;  
    private String middleName;  
    private String lastName;  
  
    ...  
}
```

- Example names that to test with:
 - Dr. Bowen Hui
 - Sir Arthur Conan Doyle

When we compare if two people have the same names, should we override equals()?

What About Multiple Inheritance?

- This means a class that is derived from two or more classes
- Ex: PickupTruck **extends** Truck
PickupTruck **extends** Car
- Problem: **Collisions** – different parents may have the same attributes and/or method signatures
 - Ex: A motorcycle inherits from both a bicycle and a car
 - Motorcycles and Bikes are 2-wheeled vehicles
 - Motorcycles and Cars have engines, gas, similar speeds
 - Cars typically have 4 wheels
- Java only supports single inheritance; multiple inheritance is not allowed

Summary of Inheritance

- Inheritance models a new IS-A relationship
- Implementation techniques:
 - `extends` (vs. `import`)
 - `protected` (vs. `private, public`)
 - `super` (vs. `this`)
 - overriding (vs. overloading)
 - `final` (when to use it)
- Class hierarchy with `Object` at the top
- Adherence to encapsulation principles

