

Computer Programming – CS 6011

Lecture 7: Inheritance & JavaFX

Fall 2023

Topics

- Object Oriented Programming
 - Inheritance
- JavaFX – UI Library

Object Oriented Programming


- Relies on the concept of objects and classes
- Supports code reuse //within classes
 - Generic code
 - Inheritance
- Languages
 - C++ / Kotlin / C#/...
 - Java

Code reuse

```
class Employee {  
    String firstName;  
    String lastName;  
    Employee(){  
        ....  
    }  
    public void setFirstName(String firstname) {  
    }  
    public void setLastName(String lastname) {  
    }  
    public void showDetails()  
    {  
        System.out.println("First Name: " + firstName + "Last Name: " + lastName);  
    }  
}
```

Do we have to repeat all the code for Employee?

Person interface, Employee Class and SalariedEmployee class?



```
class SalariedEmployee {  
    String firstName;  
    String lastName;  
    int salary;  
    SalariedEmployee(){  
        ....  
    }  
    public void setFirstName  
    ....  
    public void setLastName  
    ....  
    public void showDetails()
```

Related classes

- SalariedEmployee is an Employee Person
- Do we have to rewrite all the member variables and methods? Can we reuse the member variables (`firstName`, `lastName`) and the member function (`showDetails`)?

Inheritance property: *Is A*

- Inheritance describes “is a” relationships.
 - This is where one class is a more specific or refined version of another.
- The more “general” version is called the *base* or *parent* class.
- The more “specific” version is called the *derived* or *child* class.

Inheritance Syntax – `extends` Keyword

```
public class Derived extends Base {  
  
}
```

- Is A
 - Derived is a type of Base.
 - This means Derived has all the same member variables and methods of Base.
 - This happens “for free” just by writing `extends Base` when creating Derived.
 - Derived can also add more member variables or methods as needed.
 - Derived can also “Override” what the methods defined in Base do so that an object of type Derived can behave differently (in this respect) from an object of type Base.

Overriding Methods

- We can modify the way our parent class works.
- For example, though both the Person and Employee have a showDetails() method, the implementation would be very different.
- Writing a different version of showDetails() for Employee is called *overriding* the function.

```
public class SalariedEmployee extends Employee {  
    @Override  
    public void showDetails() { // Specialized code to handle driving a SportsCar. }  
}
```

- It is useful to add the @Override annotation to the method so that the compiler will make sure we didn't make spelling mistakes, and the visibility / parameters / return type are all consistent with the Base class (SalariedEmployee) version.

Overriding Methods: Printing Objects

- Parent class: Object / all classes inherit from it.
- Class Employee /* extends Object */
- The Object class provides a toString() method.

```
public String toString() { ... } // Overrides the Object toString() function.
```

Using parent constructor

- **class** SalariedEmployee **extends** Employee {
 int salary;

SalariedEmployee(String firstname, String lastname, **int** salary) {
 this.firstname=firstname; // **repeating what is in the parent class.**

this.lastname=lastname; // **can we reduce duplicate code?**

SomethingLike_CallPersonConstructor (firstname, lastname); // **maybe like calling the constructor in the parent class to set firstName and lastName in Employee;**

this.salary = salary;

}

Accessing super class members: *Super* Keyword

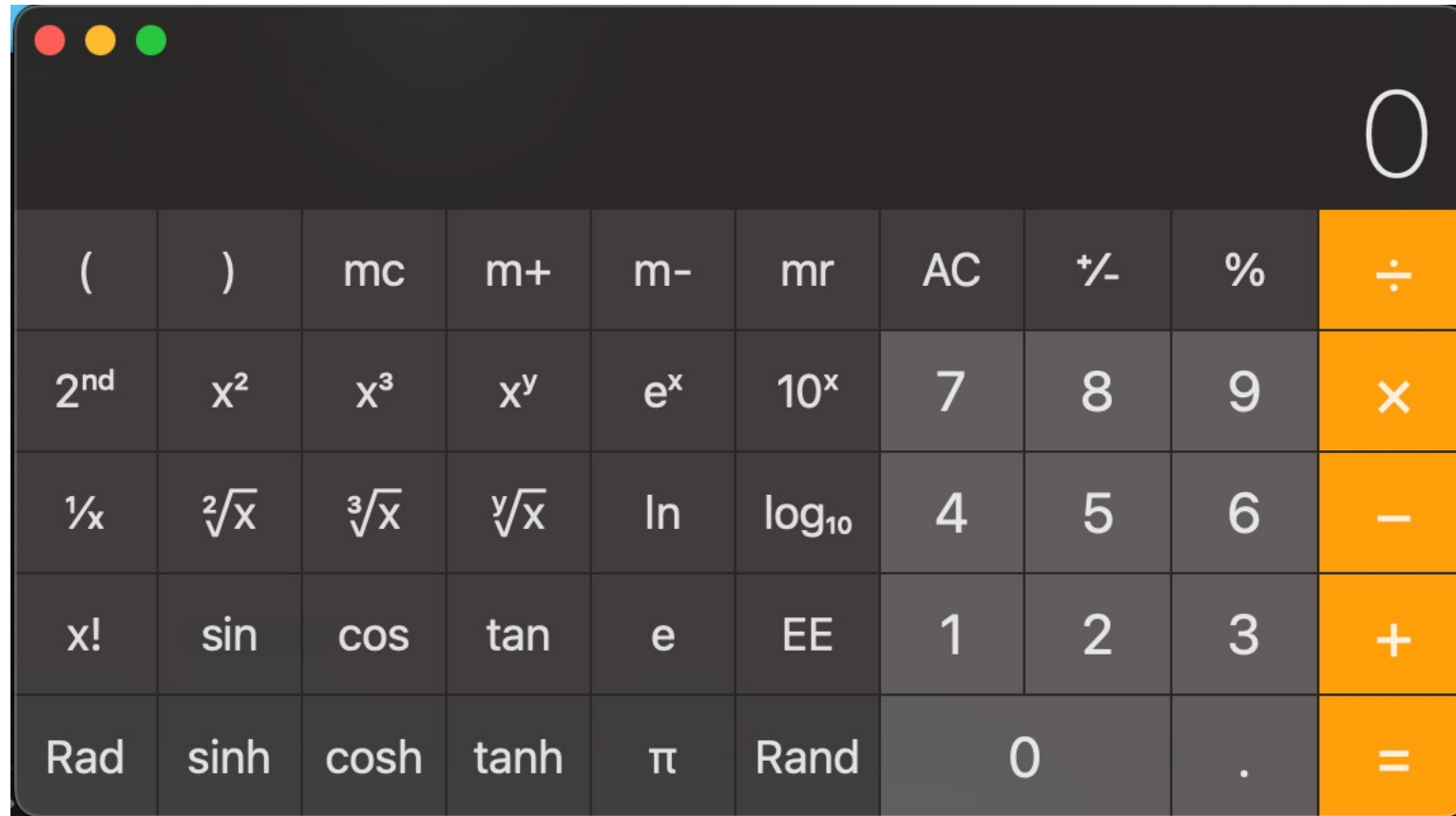
- If a base class has private members (For example, Exception's message is private), inheriting classes can't refer to them by name.
- Mostly we call the public methods of the parent class.
- However, if we have overridden the method, how do we call the parent's version?
- `super.method()` calls the parent class version of a method. This is usually done in the code of an overriding method which needs to use the parent's method to set some things up.
- We can also call our parent's constructor (again, because it will set up some things for us) before we finish constructing the child class.
 - `super(constructor params)`

Using super keyword

- **class** SalariedEmployee **extends** Employee {
 int salary;

```
SalariedEmployee(String firstname, String lastname, int salary) {  
    super (firstname, lastname)  
    this.salary = salary;  
}
```

Graphical Users Interface - GUI



JavaFX

- JavaFX is a relative new GUI library for Java that is (sort of) part of the Java standard library.
- It replaces an older library called “Swing” that’s been around a LONG time.
- Most stuff lives in the `javafx.*` package and its subpackages.
- Like many GUI libraries, it makes extensive use of the OOP concepts we’re covering this week.
- Specifically, it uses lots of interfaces and inheritance.

Major JavaFX Concepts

- The applications main window has a ***Stage***.
- The *stage* can display one ***scene*** at a time.
- A *scene* contains a “parent” object.
- A parent is a class that can contain child *nodes*.
- To make a GUI, you create nodes, add them to the parents (which are themselves nodes, which can be placed in parents). Then you create a *scene* from the top level parent, and put the scene on the stage.

JavaFX Nodes (Widgets)

- Common ones include:
 - Button
 - Label
 - Slider
 - TextBox
 - Circle, Line, Rectangle, and other shapes
 - Parents (which are containers for more nodes)

Parents

- Most other parents control layout of their children:
 - VBox stacks children vertically
 - HBox places children horizontally
 - Border lets you assign elements to the top, bottom, left, right, and center
 - Grid lets you add elements on a... grid.
 - Nested of some or all the above

Adding a Child to a Parent

- `someParent.getChildren().add(childToAdd);`
- Once a child is added to a parent, it will be drawn onto the stage by the JavaFX library.
- Some of the layouts (Parents) have dedicated methods for adding components to particular spots (BorderPane has `setCenter`, `setRight`, etc methods).

Design Tips

- Use combinations of the built-in parents
- Trying to position widgets manually is hard, brittle, and error prone
- Sometimes it makes sense to adjust a widget's position directly. You can use the `setTranslateX` and `setTranslateY` methods to do this.
 - This is how we can move our `AudioComponents` around the canvas.

What happens when you press a button?

```
Button button1 = new Button();
```

Responding to User Input

- When a user does something (clicks, drags, moves the mouse, presses a key, etc) our code will be notified by *events*.
- Nodes allow you to attach “listeners”.
- There are a variety of Listener interfaces that have one (or a few methods) that will be called automatically by JavaFX when an event occurs.
- For example:
 - `myButton.setOnAction(someListenerObject); // Or`
 - `myButton.setOnAction(e -> handlePress(e, myButton)); // Or`
 - `myButton.setOnAction(new EventHandler<ActionEvent>() {
 @Override public void handle(ActionEvent e) {
 label.setText("Accepted"); } });`

Listeners

```
public class ButtonListener implements EventHandler {  
    @Override  
    public void handle(Event event) {  
        System.out.println("In Listener");  
    }  
}
```

How is this related to OOP?

- Inheritance
- Interfaces

Tuesday Assignment(s)

- Code Reviews: WebServer Refactoring
- Assignment – Synthesizer GUI, Part 1