University of Kelaniya

Kesavan Selvarajah

BSc (UOK), MSc (Reading, UCSC)

Temporary Lecturer

Software Engineering Department

Faculty of Computing and Technology

skesa231@kln.ac.lk

Object-Oriented Programming

CSCI 21052 / ETEC 21062

Static Binding & Dynamic Binding

- **Static Binding** (also known as Early Binding) refers to the binding of methods and properties during compile time.
- **Dynamic Binding** (also known as Late Binding or Runtime Polymorphism) refers to the binding of methods and properties during runtime.
- In Dynamic Binding, the actual method called is determined by the type of the object at runtime, demonstrating the flexibility of polymorphism.

Static Binding & Dynamic Binding

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}
```

Static Binding & Dynamic Binding

```
public class BindingExample {
    public static void main(String[] args) {
        // Static Binding
        Animal animal = new Animal(); // Calls Animal's sound() at compile time
        animal.sound();

        // Dynamic Binding
        Animal dog = new Dog(); // Calls Dog's sound() at runtime
        dog.sound();
    }
}
```

Interfaces

- Using the keyword **interface**, you can fully abstract a class' interface from its implementation.
- That means, using interface, you can specify what a class must do, but not how it does it.
- Interfaces are syntactically similar to classes, but they lack instance variables, and, as a general rule, their methods are declared without any body.
- In practice, this means that you can define interfaces that don't make assumptions about how they are implemented.
- Once it is defined, any number of classes can implement an interface.
- Also, one class can implement any number of interfaces.

Interfaces

- Variables can be declared inside interface declarations.
- They are implicitly final and static, meaning they cannot be changed by the implementing class.
- They must also be initialized.
- All methods and variables are implicitly public.
- Here is an example of an interface definition.

```
interface MyInterface {
    void callback(int value);
}
```

Interfaces

- Once an interface has been defined, one or more classes can implement that interface.
- To implement an interface, include the **implements** clause in a class definition, and then create the methods required by the interface.

```
interface MyInterface {
    void myMethod();
}
class MyClass implements MyInterface {
    public void myMethod() {
        System.out.println("Implementation of myMethod");
    }
}
```

Interfaces

- If a class implements more than one interface, the interfaces are separated with a comma.

```
interface MyInterface {
    void myMethod();
}
interface Callback {
    void callback(int param);
}
class MyClass implements MyInterface, Callback {
    public void myMethod() {
        System.out.println("Implementation of myMethod");
    }

    public void callback(int value) {
        System.out.println("Callback with value: " + value);
    }
}
```

Interfaces

```
// Interface definition
interface Callback {
    void callback(int param);
}
// Another interface with the same method signature
interface AnotherCallback {
    void callback(int param);
}
// Class implementing multiple interfaces
class MyClass implements Callback, AnotherCallback {
    // Implementation of the common method
    public void callback(int param) {
        System.out.println("Callback with parameter: " + param);
    }
}
```

Abstract class Vs Interface

Feature	Abstract Class	Interface
Instantiation	Cannot be instantiated on its own.	Cannot be instantiated on its own.
Access Modifiers	Can have access modifiers (public, private, etc.).	All methods are implicitly public.
Constructor	Can have constructors.	Cannot have constructors.

Abstract class Vs Interface

Feature	Abstract Class	Interface
Methods	Can have both abstract and concrete methods.	Can only have abstract methods (no implementation).
Multiple Inheritance	Supports single inheritance (extends one class).	Supports multiple inheritance (implements multiple interfaces).
Abstract Keyword	Declared using the "abstract" keyword.	Declared using the "interface" keyword.

Abstract class Vs Interface

Feature	Abstract Class	Interface
Fields	Can have instance variables (fields).	Cannot have instance variables (fields without value).
Purpose	Used when a common base class is needed with shared implementation.	Used to define a contract for classes, enabling multiple inheritance-like behavior.

Nested Classes

- It is possible to define a class within another class; such classes are known as **Nested classes**.
- The scope of a nested class is bounded by the scope of its enclosing class. Thus, if class B is defined within class A, then B does not exist independently of A.
- A nested class has access to the members, including private members, of the class in which it is nested.
- However, the enclosing class does not have access to the members of the nested class.
- A nested class that is declared directly within its enclosing class scope is a member of its enclosing class.

Nested Classes

```
class OuterClass {
    private int outerVariable;
    // Nested class
    class NestedClass {
        private int nestedVariable;
        // Constructor for the nested class
        public NestedClass(int nestedVariable) {
            this.nestedVariable = nestedVariable;
        }
        // Method in the nested class accessing outer class members
        public void display() {
            System.out.println("Nested Variable: " + nestedVariable);
            System.out.println("Outer Variable (accessed from nested class): " +
outerVariable);
        }
    } //continues…
```

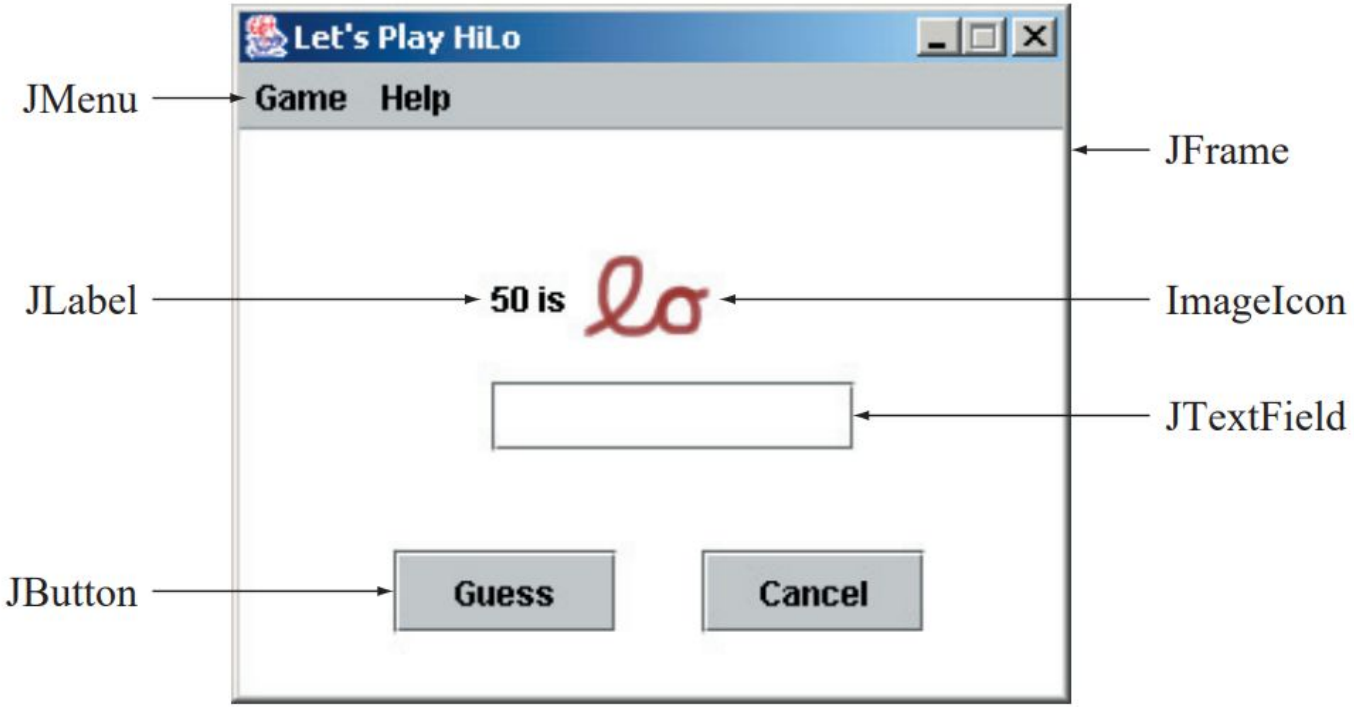
Nested Classes

```
//…Continued
    // Constructor for the outer class
    public OuterClass(int outerVariable) {
        this.outerVariable = outerVariable;
    }
    // Method in the outer class creating and using the nested class
    public void useNestedClass() {
        NestedClass nestedObj = new NestedClass(42);
        nestedObj.display();
    }
}
```

Nested Classes

```
// Main class
public class Main {
    public static void main(String[] args) {
        // Creating an instance of the outer class
        OuterClass outerObj = new OuterClass(10);

        // Using the outer class method to interact with the nested class
        outerObj.useNestedClass();
    }
}
```



Various GUI objects from the **javax.swing** package.

GUI and Event Driven Programming

- In a GUI environment, there are basically two types of windows:
  - A general purpose frame
  - A special-purpose dialog
- In Java, we use a **JFrame object** for a frame window and a **JDialog object** for a dialog.
- One of the easiest ways to provide a simple GUI-based input and output is by using the JOptionPane class. For example, when we execute the statement,

```
JOptionPane.showMessageDialog(null, "I Love Java");
```

GUI and Event Driven Programming

- The first argument to the showMessageDialog method is a frame object that controls this dialog, and the second argument is the text to display.
- In the example statement, we pass null, a reserved word, meaning there is no frame object.
- If we pass null as the first argument, the dialog appears on the center of the screen.
- If we pass a frame object, then the dialog is positioned at the center of the frame.

GUI and Event Driven Programming

```
import javax.swing.*;

public class Main {
    public static void main(String[] args) {
        JFrame jFrame;
        jFrame = new JFrame();
        jFrame.setSize(400, 300);
        jFrame.setVisible(true);
        JOptionPane.showMessageDialog(jFrame, "How are you?");
        JOptionPane.showMessageDialog(null, "Good Bye");
    }
}
```

GUI and Event Driven Programming

- If we want to display multiple lines of text, we can use a special character sequence \n to separate the lines, as in,

```
JOptionPane.showMessageDialog(null,"one\ntwo\nthree");
```

- We can also use the JOptionPane class for input by using its showInputDialog method. For example, when we execute,

```
JOptionPane.showInputDialog(null, "Enter text:");
```

- To assign the name input to an input string, we write,

```
String input;
input = JOptionPane.showInputDialog(null, "Enter text:");
```

GUI and Event Driven Programming

- To input a numerical value, we need to perform the string conversion ourselves.
- To input an integer value, say, age, we can write the code as follows,

```
String str = JOptionPane.showInputDialog(null, "Enter age:");
int age = Integer.parseInt(str);
```

- If we want to use a frame window for a word processor, we need a frame window capable of allowing the user to enter, cut, and paste text; change font; print text; and so forth.
- To design such a frame window, we would define a subclass of the JFrame class and add methods and data members to implement the needed functionalities.

GUI and Event Driven Programming

```
import javax.swing.*;

class MyFrame extends JFrame {
    private static final int FRAME_WIDTH = 300;
    private static final int FRAME_HEIGHT = 200;
    private static final int FRAME_X_ORIGIN = 150;
    private static final int FRAME_Y_ORIGIN = 250;

    public MyFrame() {
        setTitle("My First Subclass");
        setSize(FRAME_WIDTH, FRAME_HEIGHT);
        setLocation(FRAME_X_ORIGIN, FRAME_Y_ORIGIN);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

GUI and Event Driven Programming

```
class Main {
    public static void main(String[] args) {
        MyFrame myFrame;
        myFrame = new MyFrame();
        myFrame.setVisible(true);
    }
}
```

GUI and Event Driven Programming

- A frame's content pane designates the area of the frame that excludes the title and menu bars and the border.
- It is the area we can use to display the content (text, image, etc.).
- We access the content pane of a frame by calling the frame's **getContentPane** method.
- We can change its background color by calling the contentpane's **setBackground** method.

GUI and Event Driven Programming

```
import javax.swing.*;
import java.awt.*;
class MyFrame extends JFrame {
    private static final int FRAME_WIDTH = 300;
    private static final int FRAME_HEIGHT = 200;
    private static final int FRAME_X_ORIGIN = 150;
    private static final int FRAME_Y_ORIGIN = 250;
    public MyFrame() {
        setTitle("My First Subclass");
        setSize(FRAME_WIDTH, FRAME_HEIGHT);
        setLocation(FRAME_X_ORIGIN, FRAME_Y_ORIGIN);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container contentPane = getContentPane();
        contentPane.setBackground(Color.BLUE);
    }
}
```

GUI Programming

- There are two key aspects involved in GUI programming.
  - Placement of GUI objects on the content pane of a frame
  - Handling of events generated by these GUI objects
- Lets develop a sample frame window that illustrates the fundamentals of GUI programming.
- To use a button in a program, we create an instance of the **javax.swing.JButton** class.
- There are two general approaches to placing buttons on a frame's content pane, one that uses a layout manager and another that does not.

GUI Programming

- The **layout manager** for a container is an object that controls the placement of the GUI objects.
- For example, the simplest layout manager called **FlowLayout** places GUI objects in the top-to-bottom, left-to-right order.
- If we do not use any layout manager, then we place GUI objects by explicitly specifying their position and size on the content pane.
- We call this approach **absolute positioning**.

GUI Programming

```
import javax.swing.*;
import java.awt.*;

class MyFrame extends JFrame {
    private static final int FRAME_WIDTH = 300;
    private static final int FRAME_HEIGHT = 200;
    private static final int FRAME_X_ORIGIN = 150;
    private static final int FRAME_Y_ORIGIN = 250;
    private final JButton cancelButton;
    private final JButton okButton;

    public MyFrame() {
        Container contentPane = getContentPane();

----------
```

GUI Programming

```
----------
//set the frame properties
        setSize(FRAME_WIDTH, FRAME_HEIGHT);
        setResizable(false);
        setTitle("Program JButtonFrame");
        setLocation(FRAME_X_ORIGIN, FRAME_Y_ORIGIN);
//set the layout manager
        contentPane.setLayout(new FlowLayout());
//create and place two buttons on the frame's content pane
        okButton = new JButton("OK");
        contentPane.add(okButton);
        cancelButton = new JButton("CANCEL");
        contentPane.add(cancelButton);
//register 'Exit upon closing' as a default close operation
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Handling Button Events

- An action such as clicking a button is called an **event**, and the mechanism to process the events **event handling**.
- A GUI object, such as a button, where the event occurs is called an event, or simply, the **event source**.
- when the user clicks on a button, the corresponding JButton object will generate an action event.
- When an event is generated, the system notifies the relevant event listener objects.
- An **event listener**, is an object that includes a method that gets executed in response to generated events.
- It is possible for a single object to be both an event source and an event listener.

Handling Button Events

- When a button is clicked or a menu item is selected, an event source will generate an **action event**.
- For the generated events to be processed, we must associate, or register, **event listeners** to the **event sources**.
- For each type of event, we have a corresponding listener.
- For example, we have action listeners for action events, window listeners for window events, mouse listeners for mouse events, and so forth.

Handling Button Events

```
import javax.swing.*;
import java.awt.*;

class MyFrame extends JFrame  implements ActionListener{
    private static final int FRAME_WIDTH = 300;
    private static final int FRAME_HEIGHT = 200;
    private static final int FRAME_X_ORIGIN = 150;
    private static final int FRAME_Y_ORIGIN = 250;

    private final JButton cancelButton;
    private final JButton okButton;

public static void main(String[] args) {
    Ch14JButtonFrameHandler frame = new
    Ch14JButtonFrameHandler(); frame.setVisible(true);
}
----------
```

Handling Button Events

```
----------
public MyFrame() {
      Container contentPane = getContentPane();
      //set the frame properties
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setResizable(false);
      setTitle("Program JButtonFrame");
      setLocation (FRAME_X_ORIGIN, FRAME_Y_ORIGIN);

      //set the layout manager
      contentPane.setLayout(new FlowLayout());

      //create and place two buttons on the frame's content pane
      okButton = new JButton("OK");
      contentPane.add(okButton);

      cancelButton = new JButton("CANCEL");
      contentPane.add(cancelButton);
----------
```

Handling Button Events

```
   ----------
         //register this frame as an action listener of the two buttons
         cancelButton.addActionListener(this);
         okButton.addActionListener(this);

         //register 'Exit upon closing' as a default close operation
         setDefaultCloseOperation(EXIT_ON_CLOSE);
   }

   public void actionPerformed(ActionEvent event) {
         JButton clickedButton = (JButton) event.getSource();
         String buttonText = clickedButton.getText();
         setTitle("You clicked " + buttonText);
   }
}
   ----------
```

Text-Related GUI Components

- A **JTextField** object allows the user to enter a single line of text, while a **JLabel** object is for displaying uneditable text.
- A **JTextArea** object allows the user to enter multiple lines of text.
- It can also be used for displaying multiple lines of uneditable text.
- Like a **JButton** object, an instance of **JTextField** generates an action event.
- A **JTextField** object generates an action event when the user presses the Enter key while the object is active.
- The **JLabel** class is not limited to the display of text. We can also use it to display an image.

GUI Components in action

Main.java

```
public class Main {
    public static void main(String[] args) {
        MyFrame myFrame = new MyFrame();
        myFrame.setVisible(true);
    }
}
```

#1 JLabel

MyFrame.java

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public temp() {
        getContentPane().setLayout(null);
        setTitle("Name and Age");
        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        JLabel nameLabel = new JLabel("Name:");
        nameLabel.setFont(new Font("Arial", Font.PLAIN, 15));
        nameLabel.setSize(300, 30);
        nameLabel.setLocation(50, 20);
        add(nameLabel);
    }
}
```

#2 JTextField

MyFrame.java

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public temp() {
        getContentPane().setLayout(null);
        setTitle("Name and Age");
        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //......

        JTextField nameTextField = new JTextField();
        nameTextField.setFont(new Font("Arial", Font.PLAIN, 15));
        nameTextField.setSize(300, 30);
        nameTextField.setLocation(200, 20);
        add(nameTextField);
    }
}
```

#3 JTextArea

MyFrame.java

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public MyFrame() {
        getContentPane().setLayout(null);
        setTitle("Name and Age");
        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //......

        JTextArea resultsText = new JTextArea();
        resultsText.setFont(new Font("Arial", Font.PLAIN, 15));
        resultsText.setSize(300, 100);
        resultsText.setLocation(150, 230);
        resultsText.setLineWrap(true);
        add(resultsText);
    }
}
```

#4 JButton

MyFrame.java

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public temp() {
        getContentPane().setLayout(null);
        setTitle("Name and Age");
        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //......

        JButton submitButton = new JButton("Submit");
        submitButton.setFont(new Font("Arial", Font.PLAIN, 15));
        submitButton.setSize(100, 20);
        submitButton.setLocation(150, 150);
        add(submitButton);
    }
}
```

#5 JCheckBox

MyFrame.java

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public MyFrame() {
        getContentPane().setLayout(null);
        setTitle("Name and Age");
        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //......

        JCheckBox checkBox = new JCheckBox("Accept Terms.");
        checkBox.setFont(new Font("Arial", Font.PLAIN, 15));
        checkBox.setSize(250, 20);
        checkBox.setLocation(145, 190);
        add(checkBox);
    }
}
```

#6 JRadioButton

MyFrame.java

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public MyFrame() {
        getContentPane().setLayout(null);
        setTitle("Name and Age");
        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //......

        JRadioButton male = new JRadioButton("Male");
        male.setFont(new Font("Arial", Font.PLAIN, 15));
        male.setSelected(true);
        male.setSize(75, 20);
        male.setLocation(200, 100);
        add(male);

.....
```

#6 JRadioButton

MyFrame.java

```
.....

        JRadioButton female = new JRadioButton("Female");
        female.setFont(new Font("Arial", Font.PLAIN, 15));
        female.setSelected(false);
        female.setSize(80, 20);
        female.setLocation(275, 100);
        add(female);

        ButtonGroup gender = new ButtonGroup();
        gender.add(male);
        gender.add(female);

    }
}
```

#5 JComboBox

MyFrame.java

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public MyFrame() {
        getContentPane().setLayout(null);
        setTitle("Name and Age");
        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //......

        String [] course={"ICT","ET","CS"};
        JComboBox courseCombo = new JComboBox(course);
        courseCombo.setSize(90,20);
        courseCombo.setLocation(200,70);
        add(courseCombo);
    }
}
```

Thank you