ÖZYEĞİN UNIVERSITY

CS 102

Object Oriented Programming

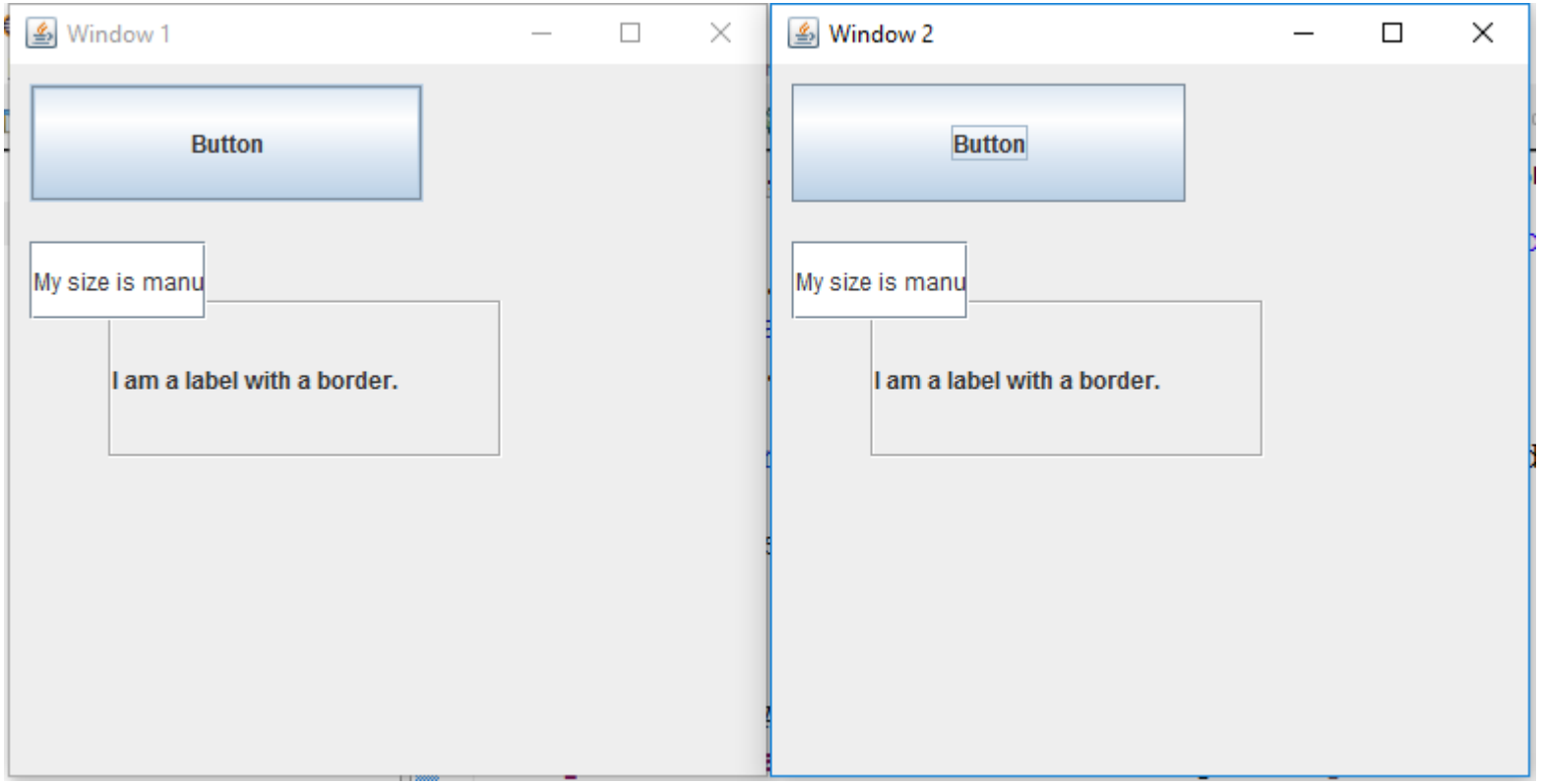
Inner Classes

Reyyan Yeniterzi

reyyan.yeniterzi@ozyegin.edu.tr

GUI with several identical frames

- Remember this GUI?



GUI with several identical frames

- We have implemented our own window (frame) class and instantinated it multiple times.

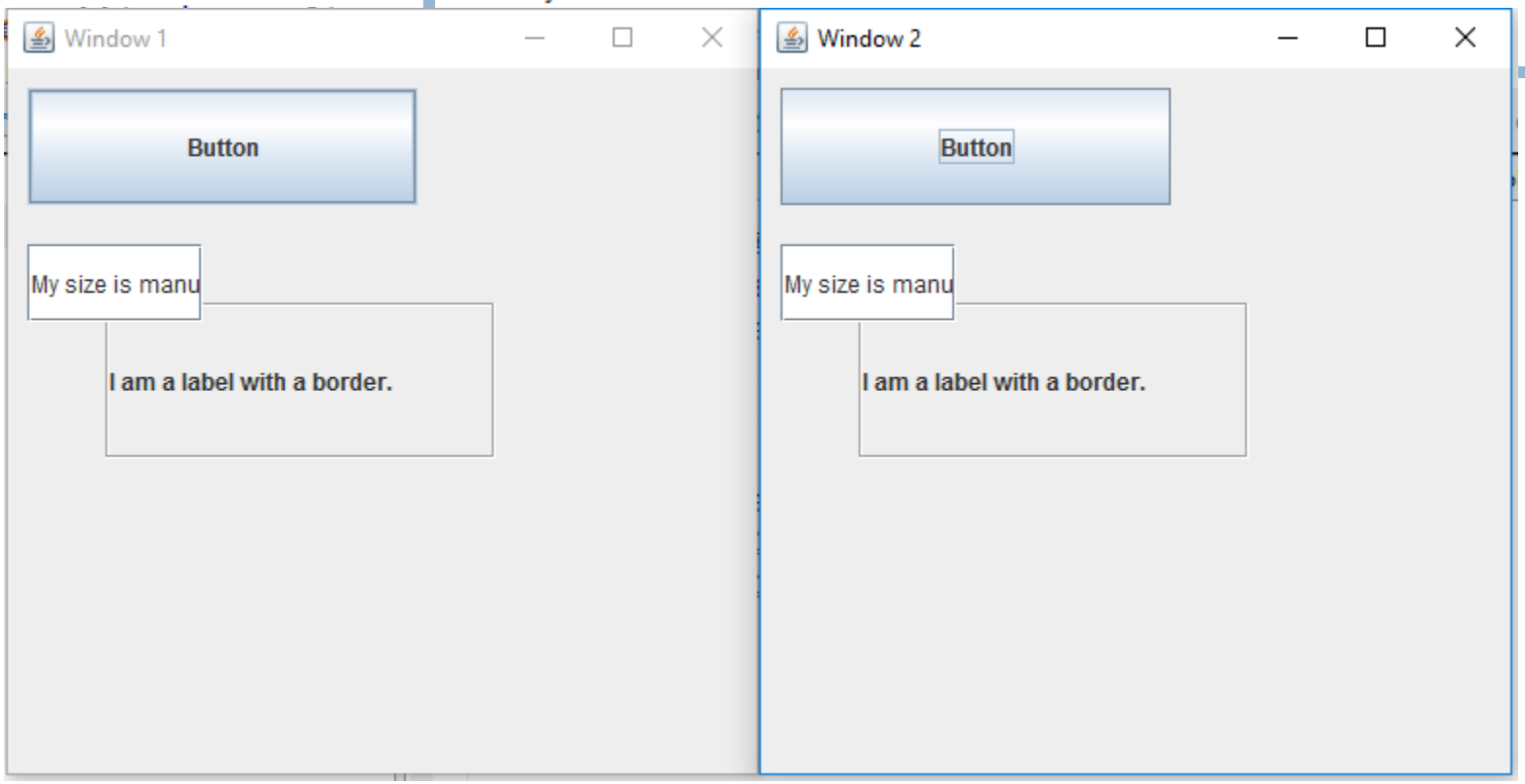
```java
class MyWindow extends JFrame {

    JPanel mainPanel;
    JButton button;
    JTextField field;
    JLabel label;

    public MyWindow() {
        super();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);

        mainPanel = new JPanel();
        add(mainPanel);
        mainPanel.setLayout(null);

        button = new JButton("Button");
        button.setBounds(10, 10, 200, 60);
        mainPanel.add(button);

        field = new JTextField("My size is manually set...");
        field.setBounds(10, 90, 90, 40);
        mainPanel.add(field);

        label = new JLabel("I am a label with a border.");
        label.setBounds(50, 120, 200, 80);
        mainPanel.add(label);
        label.setBorder(new EtchedBorder());

        setVisible(true);
    }
}
```

```java
class MyWindow extends JFrame {

    JPanel mainPanel;
    JButton button;
    JTextField field;
    JLabel label;

    public MyWindow() {
        super();
        setDefaultCloseOpe
        setSize(400, 400);

        mainPanel = new JP
        add(mainPanel);
        mainPanel.setLayou

        button = new JButton("Button");
        button.setBounds(10, 10, 200, 60);
        mainPanel.add(button);

        field = new JTextField("My size is manually set...");
        field.setBounds(10, 90, 90, 40);
        mainPanel.add(field);

        label = new JLabel("I am a label with a border.");
        label.setBounds(50, 120, 200, 80);
        mainPanel.add(label);
        label.setBorder(new EtchedBorder());

        setVisible(true);
    }
}
```

```java
public class AbsoluteLayoutWindows {
    public static void main(String[] args) {
        JFrame frame = new MyWindow();
        frame.setTitle("Window 1");
        frame.setLocation(0, 0);

        JFrame frame2 = new MyWindow();
        frame2.setTitle("Window 2");
        frame2.setLocation(400, 0);
    }
}
```

```java
class MyWindow extends JFrame {

    JPanel mainPanel;
    JButton button;
    JTextField field;
    JLabel label;

    public MyWindow() {
        super();
        setDefaultCloseOpe
        setSize(400, 400);

        mainPanel = new JP
```

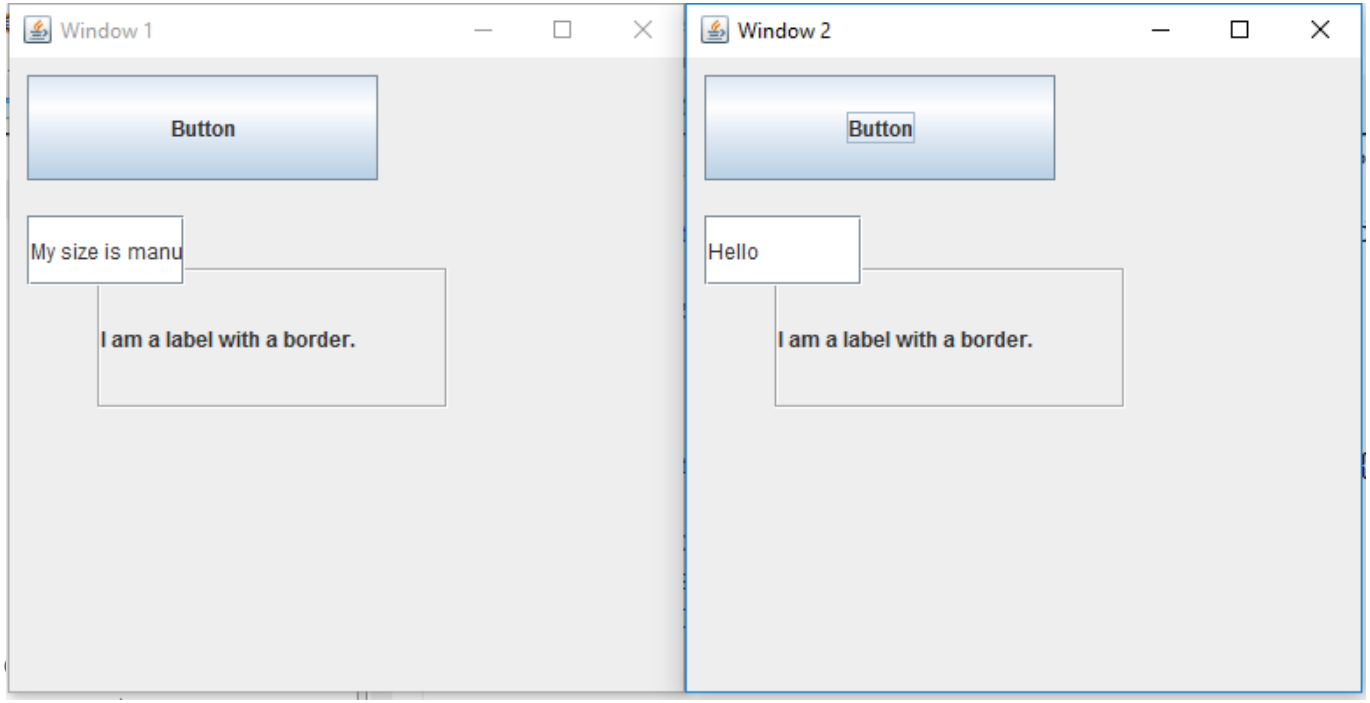
```java
public class AbsoluteLayoutWindows {
    public static void main(String[] args) {
        JFrame frame = new MyWindow();
        frame.setTitle("Window 1");
        frame.setLocation(0, 0);

        JFrame frame2 = new MyWindow();
        frame2.setTitle("Window 2");
        frame2.setLocation(400, 0);
    }
```




```java
    }
}
```

Adding an Event

- We have implemented an actionListener for the button. When the button is pressed, the text inside textBox is changed to 'Hello'



Add a Button Event


```
        field = new JTextField("My size is manually set...");
        field.setBounds(10, 90, 90, 40);
        mainPanel.add(field);

        label = new JLabel("I am a label with a border.");
        label.setBounds(50, 120, 200, 80);
        mainPanel.add(label);
        label.setBorder(new EtchedBorder());

        button.addActionListener(new MyButtonHandler(field));

        setVisible(true);
    }
}

class MyButtonHandler implements ActionListener {

    JTextField T;
    MyButtonHandler(JTextField T) {
        this.T = T;
    }

    public void actionPerformed(ActionEvent e) {
        T.setText("Hello");
    }
}
```

```java
class MyButtonHandler implements ActionListener {

    JTextField T;
    MyButtonHandler(JTextField T) {
        this.T = T;
    }

    public void actionPerformed(ActionEvent e) {
        T.setText("Hello");
    }
}
```

- This class won't be commonly used by others.

```java
class MyButtonHandler implements ActionListener {

    JTextField T;
    MyButtonHandler(JTextField T) {
        this.T = T;
    }

    public void actionPerformed(ActionEvent e) {
        T.setText("Hello");
    }
}
```

- This class won't be commonly used by others.
- Therefore, it can be an inner class.

Nested Class

- Until now all of our classes were 'top-level'.
- In Java, one can define a class inside another class.
- Nested class
  - Inner class (non-static nested class)
  - Static nested class

Nested Class

- Until now all of our classes were 'top-level'.
- In Java, one can define a class inside another class.
- Nested class
  - Inner class (non-static nested class)
  - Static nested class

Inner Classes

- They are defined inside another class

```
class topLevelClass {

    //...

    class innerClass {

        //...
    }
}
```

Inner class

Outer class

Static Nested vs. Inner Classes


```
class topLevelClass {

    //...
    class innerClass {

        //...
    }

    static class staticNestedClass {

        //...
    }
}
```

Static Nested vs. Inner Classes


```
class topLevelClass {

    //...
    class innerClass {

        //...
    }

    static class staticNestedClass {

        //...
    }
}
```

A nested class needs to be static

- If the nested class has static members or functions
- If the nested class is instantinated within a static method of the outer class.

Nested Classes

- They can be declared like any other class instance or method.
- They are members of the outer class like other class instances and methods.
- They can be placed anywhere in the class.

Visibility

- They can use the usual access modifiers like public, private and protected.
  - The same visibility rules apply.
  - If it is defined private, it cannot be accessed outside the outer class.
- They are local to the outer class.
  - Another class with the same name can be defined and used outside outer class.

Access Levels

- Inner (non-static nested) classes can access all variables & methods of enclosing class (including private fields & methods)
- Static nested classes **do not** have access to all members of the enclosing class.
  - Static nested classes can access to only static variables of the outer class.

Access Levels


```
class outerClass {
    private int outerVar;

    class innerClass {

        public void method() {
            System.out.println(outerVar);
        }
    }

    static class staticNestedClass {

        public void method() {
            System.out.println(outerVar);
        }
    }
}
```

Cannot make a static reference to the non-static field outerVar

1 quick fix available:

Change 'outerVar' to 'static'

Press 'F2' for focus

Access Levels


```
class outerClass {
    private int outerVar;
    static private int staticVar;

    static class staticNestedClass {

        public void method() {
            System.out.println(outerVar);
            System.out.println(staticVar);
        }
    }
}
```

Static Nested Class

- Therefore, static nested class is not really an inner class.
- It is actually top-level class placed inside another class.

Access levels in Inner Classes

- Inner (non-static nested) classes can access all variables & methods of enclosing class (including private fields & methods)
- The outer class has also full access to inner class's private members and methods.

- Between inner and outer classes private access is same as public.

Access levels in Inner Classes


```
class outerClass {
    private int outerVar;

    class innerClass {
        private int innerVar;
        public void method() {
            System.out.println(outerVar);
        }
    }

    public void method() {
        innerClass i = new innerClass();
        System.out.println(i.innerVar);
    }
}
```

Why use nested classes?

- It is a way of logically grouping classes that are only used in one place
  - If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together.
  - Nesting such "helper classes" makes their package more streamlined.
- They make the outerclass more self-contained.

Source:https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Why use nested classes?

- It increases data encapsulation
  - Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private.
  - By hiding class B within class A, A's members can be declared private and B can access them.
  - In addition, B itself can be hidden from the outside world.

Source:https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Why use nested classes?

- It can lead to more readable and maintainable code. Simplifies the coding process.
  - Nesting small classes within top-level classes places the code closer to where it is used.

Source:https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

When to use nested classes?

- When nested class needs to be local to the outer class definition.
  - Nested class won't be or should not be used by other classes
- When the name of an inner class may be reused for something else outside the outer class definition

Example

- Nested classes are commonly used in GUI event handling
- Put this class inside the Window class

```
class MyButtonHandler implements ActionListener {

    JTextField T;
    MyButtonHandler(JTextField T) {
        this.T = T;
    }

    public void actionPerformed(ActionEvent e) {
        T.setText("Hello");
    }
}
```

Example


```
class MyWindow extends JFrame {
    private JPanel mainPanel;
    private JButton button;
    private JTextField field;
    private JLabel label;

    class MyButtonHandler implements ActionListener {

        private JTextField T;
        MyButtonHandler(JTextField T) {
            this.T = T;
        }

        public void actionPerformed(ActionEvent e) {
            T.setText("Hello");
        }
    }

    public MyWindow() {
        super();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);
```

Inner Class Example


```
class MyWindow extends JFrame {
    private JPanel mainPanel;
    private JButton button;
    private JTextField field;
    private JLabel label;

    class MyButtonHandler implements ActionListener {

        private JTextField T;
        MyButtonHandler(JTextField T) {
            this.T = T;
        }

        public void actionPerformed(ActionEvent e) {
            T.setText("Hello");
        }
    }

    public MyWindow() {
        super();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);
```

These are visible to the MyButtonHandler class

Inner Class Example


```
class MyWindow extends JFrame {
    private JPanel mainPanel;
    private JButton button;
    private JTextField field;
    private JLabel label;

    class MyButtonHandler implements ActionListener {

        private JTextField T;
        MyButtonHandler(JTextField T) {
            this.T = T;
        }

        public void actionPerformed(ActionEvent e) {
            T.setText("Hello");
        }
    }

    public MyWindow() {
        super();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);
```

These are visible to the MyButtonHandler class

So we don't have to pass TextField as parameter

Inner Class Example


```
class MyWindow extends JFrame {
    private JPanel mainPanel;
    private JButton button;
    private JTextField field;
    private JLabel label;

    class MyButtonHandler implements ActionListener {

        private JTextField T;
        MyButtonHandler(JTextField T) {
            this.T = T;
        }

        public void actionPerformed(ActionEvent e) {
            T.setText("Hello");
        }
    }

    public MyWindow() {
        super();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);
```

These are visible to the MyButtonHandler class

We don't need this part any more.

The inner class MyButtonHandler can access all fields of MyWindow

Inner Class Example


```
class MyWindow extends JFrame {
    private JPanel mainPanel;
    private JButton button;
    private JTextField field;
    private JLabel label;

    class MyButtonHandler implements ActionListener {

        public void actionPerformed(ActionEvent e) {
            field.setText("Hello");
        }
    }

    public MyWindow() {
        super();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);

        mainPanel = new JPanel();
        add(mainPanel);

        mainPanel.setLayout(null);
```

Inner Class Example


```
class MyWindow extends JFrame {
    private JPanel mainPanel;
    private JButton button;
    private JTextField field;
    private JLabel label;

    class MyButtonHandler implements ActionListener {

        public void actionPerformed(ActionEvent e) {
            field.setText("Hello");
        }
    }

    public MyWindow() {
        super();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);

        mainPanel = new JPanel();
        add(mainPanel);

        mainPanel.setLayout(null);
```

```
class MyButtonHandler implements ActionListener {

    private String myButtonText = "MyButton";

    public void actionPerformed(ActionEvent e) {
        field.setText("Hello");
    }
}

public MyWindow() {
    super();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 400);

    mainPanel = new JPanel();
    add(mainPanel);
    mainPanel.setLayout(null);

    button = new JButton("Button");
    button.setBounds(10, 10, 200, 60);
    mainPanel.add(button);

    MyButtonHandler buttonHandler = new MyButtonHandler();
    button.addActionListener(buttonHandler);
    button.setText(buttonHandler.myButtonText);
```

```
class MyButtonHandler implements ActionListener {

    private String myButtonText = "MyButton";

    public void actionPerformed(ActionEvent e) {
        field.setText("Hello");
    }
}

public MyWindow() {
    super();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 400);

    mainPanel = new JPanel();
    add(mainPanel);
    mainPanel.setLayout(null);

    button = new JButton("Button");
    button.setBounds(10, 10, 200, 60);
    mainPanel.add(button);

    MyButtonHandler buttonHandler = new MyButtonHandler();
    button.addActionListener(buttonHandler);
    button.setText(buttonHandler.myButtonText);
```

Outer class can access the private variables and methods through an inner class object

Inner Classes

- Inner class instance is associated with an instance of outer class
- The inner class instance is tied to outer class instance at the moment or instantianition.
  - An inner class instance exists within an outer class instance
  - It cannot be changed afterwards

Inner Classes

- Inner class instance is associated with an instance of outer class
- The inner class instance is tied to outer class instance at the moment or instantianition.
  - An inner class instance exists within an outer class instance
  - It cannot be changed afterwards
- Therefore, in order to instantiate an inner class, an outer class instance needs to be instantiated first. Then the inner class object will be insantiated within outer class object.

Inner Classes

- If the inner class is public, an inner class object can be instantinated outside the outer class.
- It must be instantinated using the outer class object.

```
OuterClass outerObj = new OuterClass();
OuterClass.InnerClass innerObj =
                outerObj.new InnerClass();
```

Inner Classes

- If the inner class is public, an inner class object can be instantinated outside the outer class.
- It must be instantinated using the outer class object.

```
class outerClass {

    public class innerClass {

        //...
    }
}

public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();
          outerClass.innerClass i = o.new innerClass();
      }
}
```

Inner Classes

- If the inner class is public, an inner class object can be instantinated outside the outer class.
- It must be instantinated using the outer class object.

```
class outerClass {

    public class innerClass {

        //...
    }
}

public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();
          outerClass.innerClass i = o.new innerClass();
      }
}
```

The outer class object needs to come before the `new` keyword.

Inner Classes


```
class outerClass {
    private int outerVar;

    class innerClass {
        private int innerVar;
        public void methodIn() {
            System.out.println(outerVar);
        }
    }
    public void methodOut() {
        innerClass i = new innerClass();
        System.out.println(i.innerVar);
    }
}
public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();

          outerClass.innerClass i = o.new innerClass();
          i.methodIn();
      }
}
```

- Inner class object can call functions of the inner class

Static Nested Classes

- Unlike inner classes, they don't need to a outer class object during instantiation.
- They are associated with the outer class, not the objects of the outer class.

Static Nested Classes


```
class outerClass {
    private int outerVar;
    static private int staticVar;

    class innerClass {
        private int innerVar;
    }

    static class staticNestedClass {

        public void method() {
            System.out.println(staticVar);
        }
    }
}

public class innerClassExample01 {
      public static void main(String[] args) {

          outerClass.staticNestedClass sn = new outerClass.staticNestedClass();

          outerClass o = new outerClass();
          outerClass.innerClass i = o.new innerClass();
      }
}
```

Static Nested Classes

- Unlike inner classes, they don't need to a outer class object during instantiation.
- They are associated with the outer class, not the objects of the outer class.
- An instance of an inner class does not contain an implicit reference to an instance of the outer class
  - Since they are not connected with outer class objects
    - They cannot reference any instance variables
    - They cannot call any instance methods.

Static Nested Classes


```
class outerClass {
    private int outerVar;
    static private int staticVar;

    class innerClass {
        private int innerVar;
        public void method() {
            methodOut();
        }
    }

    static class staticNestedClass {

        public void method() {
            methodOut();
            System.out.println(staticVar);
        }
    }

    public void methodOut() {
        System.out.println(outerVar);
    }
}
```

Static Nested Classes

- A static nested class interacts with objects of its outer class (and other classes) just like any other top-level class.
- Therefore, a static nested class is behaviorally a top-level class that has been places in another top-level class.

Nested Inner Classes

- An inner class can have another inner class inside

```
class outerClass {

    public class innerClass {

        public class innerinnerClass {

            //...
        }
    }
}

public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();
          outerClass.innerClass i = o.new innerClass();
          outerClass.innerClass.innerinnerClass ii = i.new innerinnerClass();
      }
}
```

Scope Resolution

- Assume that we have the following main...

```
public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();

          outerClass.innerClass i = o.new innerClass();
          i.methodIn();
      }
}
```

Scope Resolution

- What is the output?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(var);
        }
    }
}
```

Scope Resolution

- What is the output?

3

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(var);
        }
    }
}
```

Scope Resolution

- What is the output?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(this.var);
            System.out.println(var);
        }
    }
}
```

Scope Resolution

- What is the output?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(this.var);
            System.out.println(var);
        }
    }
}
```

```
2
3
```

Scope Resolution

- How can we access `outerClass`'s `var`?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(this.var);
            System.out.println(var);
        }
    }
}
```

Scope Resolution

- How can we access `outerClass`'s `var`?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(this.var);
            System.out.println(var);
        }
    }
}
```

```
outerClass.this.var
```

Scope Resolution

- What is the output?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(outerClass.this.var);
            System.out.println(this.var);
            System.out.println(var);
        }
    }
}
```

Scope Resolution

- What is the output?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(outerClass.this.var);
            System.out.println(this.var);
            System.out.println(var);
        }
    }
}
```

```
1
2
3
```

Scope Resolution

- Assume that we have the following main...

```
public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();

          outerClass.innerClass i = o.new innerClass();
          i.methodIn();

          o.methodOut();
      }
}
```

Scope Resolution

- What is the output?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(outerClass.this.var);
            System.out.println(this.var);
            System.out.println(var);
        }
    }
    public void methodOut() {
        innerClass i = new innerClass();
        System.out.println(var);
        System.out.println(i.var);
    }
}
```

Scope Resolution

- What is the output?

```
class outerClass {
    private int var = 1;

    class innerClass {
        private int var = 2;
        public void methodIn() {
            int var = 3;
            System.out.println(outerClass.this.var);
            System.out.println(this.var);
            System.out.println(var);
        }
    }
    public void methodOut() {
        innerClass i = new innerClass();
        System.out.println(var);
        System.out.println(i.var);
    }
}
```

```
1
2
3
1
2
```

Inner Classes – Method Resolution

- A method is called inside the inner class
  - Method is resolved within the inner class
  - If there is not such a method inside the inner class, then method can be resolved within the corresponding outer class
  - If there is not such a method inside the outer class, and if there are multiple levels of nested clases, keep on looking

Inner Classes – Method Resolution

- Both the inner and outer class have ***method()***
  - ***method()*** or **this.*method()***

    the method inside the inner class will be called

    (**this** inside the inner class refers to inner class object)

  - ***OuterClassName*.this.*method()***

    the method inside the outer class will be called

Inner Classes – Method Resolution


```java
class outerClass {
    private int outerVar = 10;
    class innerClass {
        private int innerVar = 5;
        public void method() {
            System.out.println(innerVar);
        }
        public void method1() {
            method();
            outerClass.this.method();
        }
    }
    public void method() {
        System.out.println(outerVar);
    }
    public void methodOut() {
        System.out.println(outerVar);
    }
}
public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();
          outerClass.innerClass i = o.new innerClass();
          i.method1();
      }
}
```

- What is the output?

Inner Classes – Method Resolution


```
class outerClass {
    private int outerVar = 10;
    class innerClass {
        private int innerVar = 5;
        public void method() {
            System.out.println(innerVar);
        }
        public void method1() {
            method();
            outerClass.this.method();
        }
    }
    public void method() {
        System.out.println(outerVar);
    }
    public void methodOut() {
        System.out.println(outerVar);
    }
}
public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();
          outerClass.innerClass i = o.new innerClass();
          i.method1();
      }
}
```

- What is the output?

```
5
10
```

Inner Classes

- A class can have multiple inner classes.
- Inner classes can access each other's private members as long as the its objst is used as the calling object.

Compiling Java classes

- Compiling a java class creates a *classname.class* file
- Inner classes compile to seperate .class files with a $ symbol in their names.
  - *outerClassName.class*
  - *outerClassName$innerClassName.class*

Inheritance and Inner Classes

- A class derived from the outer class inherits the inner class as well.
- The drived class cannot override the inner class
- Both outer and inner class can be derived classes.

Anonymous Classes

Anonymous Inner Classes

- These are inner classes without the class name.
- They are defined at the same place an instance is created.
  - The class definition is embedded inside the new operator expression.
  - Can be inside a method
- Use them when you only need to create instances of inner class in one location (only one object)

Anonymous Inner Classes

- These classes don't have any constructors.
- They are either
  - Derived from a class or
  - Implements an interface

Anonymous Inner Classes

- Assume that we have the following class

```
class class1 {
    void method () {
        System.out.println("1");
    }
}
```

Anonymous Inner Classes

- Implement an anonymous inner class derived from `class1`

```
class class1 {
    void method () {
        System.out.println("1");
    }
}
class outerClass {

    public void methodOut() {
        class1 c = new class1(){
            void method () {
                System.out.println("2");
            }
        };
        c.method();
    }
}
```

Anonymous Inner Classes

- Implement an anonymous inner class derived from `class1`

```
class class1 {
    void method () {
        System.out.println("1");
    }
}
class outerClass {

    public void methodOut() {
        class1 c = new class1(){
            void method () {
                System.out.println("2");
            }
        };
        c.method();
    }
}
```

No class name is given. Only the name of the superclass or interface is used for referencing

Anonymous Inner Classes

- What is the output?

```
class class1 {
    void method () {
        System.out.println("1");
    }
}
class outerClass {

    public void methodOut() {
        class1 c = new class1(){
            void method () {
                System.out.println("2");
            }
        };
        c.method();
    }
}
public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();
          o.methodOut();
      }
}
```

Anonymous Inner Classes

- What is the output?

```
class class1 {
    void method () {
        System.out.println("1");
    }
}
class outerClass {

    public void methodOut() {
        class1 c = new class1(){
            void method () {
                System.out.println("2");
            }
        };
        c.method();
    }
}
public class innerClassExample01 {
      public static void main(String[] args) {
          outerClass o = new outerClass();
          o.methodOut();
      }
}
```

2

Anonymous Inner Classes

- Defining anonymous inner classes

```
button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            field.setText("Hello");
        }
    });
```

Anonymous Inner Classes

- Defining anonymous inner classes

```
button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            field.setText("Hello");
        }
    });
```

Anonymous Inner Classes

- Defining anonymous inner classes

```
button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            field.setText("Hello");
        }
    });
```

- Like inner classes, anonymous inner classes have full access to the fields and methods of the outer class

Summary: Nested Class

- Non-Static Nested (Inner) Class
  - Like any other ordiary class, just defines inside another class
  - Has full access to outer class variables
  - Anonymous Class:
    - Useful for one time instatiated objects
- Static Nested Class
  - Static inner class
  - Acts like an outer class

Any Questions ?