Nested and anonymous classes.

Events and events handlers.

Nested classes

- We can define a class within another class.

- Such a class is called a nested class

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Why use nested classes?

- Logical grouping of classes

  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

- Increased encapsulation

  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.

- More readable, maintainable code

  Nesting small classes within top-level classes places the code closer to where it is used.

Nested classes

- Nested classes are divided into two categories
  - Static

    Nested classes that are declared static are simply called **static nested classes**
  - Non-static

    non static nested classes are called **inner classes**
- The nested class has access to all members of the outer class(including private members)
- The outer class has access to all members of the inner class

Static nested classes

- A static nested class is associated with its outer class
- They are accessed using the enclosing class name:

  <Outer class name>.<Nested class name>

- They have access only to the static members of the outer class
- Nested class can be *private*

Static nested classes

```
public class OuterClass {
    private String value;
    private static int count;

    void accessMemberFromTheNestedClass(){
        System.out.println(NestedStaticClass.name);
        //compilation error:
        //System.out.println(NestedStaticClass.age);
    }

    public static class NestedStaticClass {
        private static String name = "SoftAcad";
        private String age = "SoftAcad";

        public void printMemberFromOuterClass() {
            System.out.println(count);
            //compilation error:
            //System.out.println(value);
        }
    }
}
```

Access private member of the outer class

Static nested classes

```
public class OuterTest {
    public static void main(String[] args) {
        OuterClass.NestedStaticClass nsc = new OuterClass.NestedStaticClass();

        nsc.printMemberFromOuterClass();
    }
}
```

Inner classes

- Non-static nested classes are called inner classes
- An inner class is associated with an **instance** of its enclosing class
- It has direct access to enclosing class' methods and fields (including private members)
- Cannot define any static members itself.
- Objects that are instances of an inner class always exist within an instance of the outer class
- To instantiate an inner class, you must first instantiate the outer class and then:

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

Inner classes

```
public class OuterClass {
    private String value;
    private static int count;

    void useMemberFromInnerClass(){
        InnerClass inner = new InnerClass();
        System.out.println(inner.age);
    }

    public class InnerClass {
        //compilation error:
        //private static String name = "SoftAcad";
        private String age = "SoftAcad";

        public void printMemberFromOuterClass() {
            System.out.println(count);
            System.out.println(value);
        }
    }
}
```

Local and anonymous classes

There are two additional types of inner classes.

- Local inner class
  - You can declare an inner class within the body of a method.
  - The local class can access only final variables declared in the enclosing block of code.
- Anonymous inner class
  - An inner class declared within the body of a method without naming it.

Anonymous classes

```java
public class Page {
    private String title;
    private String text;

    public Page(){

    }

    public Page(String title, String text) {
        this.title = title;
        this.text = text;
    }
}
```

```java
public class Book {
    public void addNewPage(Page pageToAdd) {
        //...
    }
}
```

Anonymous classes

```
public class BookTest {
    public static void main(String[] args) {
        Page firstPage = new Page("Intro", "Once upon a time...");

        Book book = new Book();
        book.addNewPage(firstPage);

        book.addNewPage(new Page(){
            private boolean isReaded;

            public boolean isReaded() {
                return isReaded;
            }

            public void setReaded(boolean isReaded) {
                this.isReaded = isReaded;
            }
        });
    }
}
```

Anonymous class which extends Page

Anonymous classes

- Anonymous classes are widely used for implementing an interface.
- They can be used for implementing event listeners

```java
public interface IDVDRemoteController {
    void play();
    void eject();
    void insertDisc();
    void stop();
}
```

```java
public class Person {
    public void watchMovieOnDVD(IDVDRemoteController remoteController) {
        remoteController.play();
        //...
    }
}
```

Anonymous classes

```
public class PersonDemo {
    public static void main(String[] args) {
        Person ivan = new Person();

        ivan.watchMovieOnDVD(new IDVDRemoteController() {
            public void stop() {
                System.out.println("DVD is stoped");
            }
            public void play() {
                System.out.println("DVD is started");
            }
            public void insertDisc() {
                //...
            }
            public void eject() {
                //...
            }
        });
    }
}
```

Anonymous class

which implements

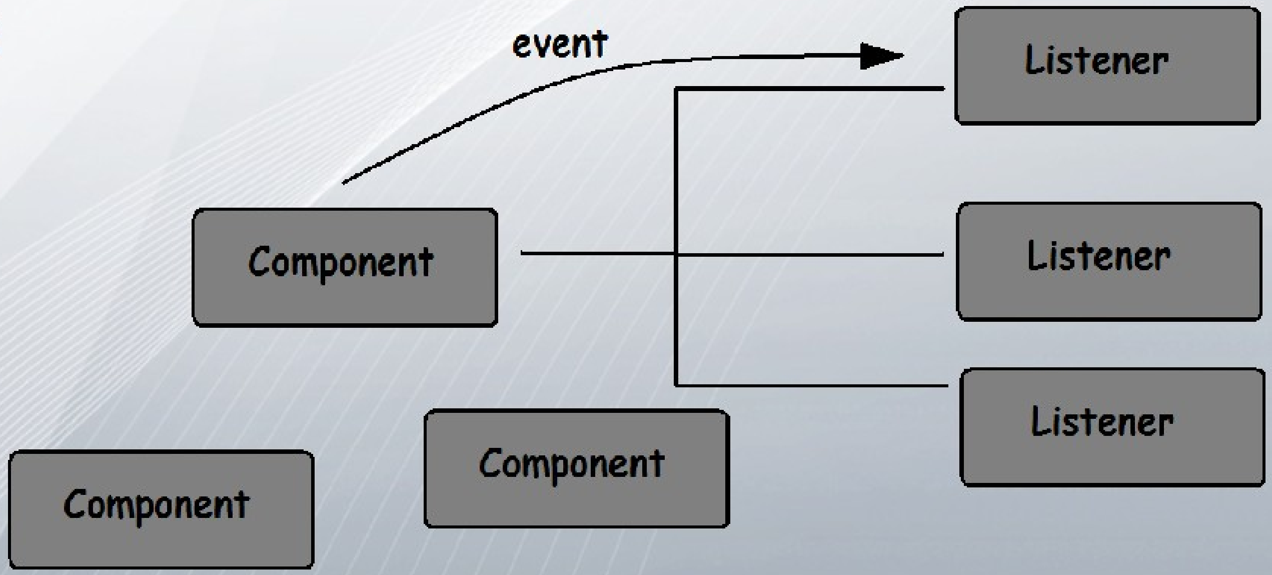
IDVDRemoteController

Events and listeners

- Components (JButton, JTextField ...) fire off events to indicate some kind of action.

The concept rely on:

- Components
- Events
- Listeners



Events and Listener

Components generate events. An event is a component's way of letting a listener know that something has happened.

- For example, the JButton fires off an ActionEvent whenever the user presses it.
- The entire point of an event is to inform a listener that something has happened to a component in the GUI.
- An event includes all of the information that a listener needs to figure out what happened and to whom it happened.
- In order to receive an ActionEvent, a listener must implement the ActionListener interface and register itself with the component.

Events and listeners

```java
public class ChristmasPanel extends JPanel {

    private JButton button;
    private JLabel message;

    public ChristmasPanel() {
        button = new JButton("Click Me");
        add(button);

        message = new JLabel("");
        add(message);

        button.addActionListener(new ChristmasButtonListener());
    }

    public class ChristmasButtonListener implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
            message.setText("MERRY CHRISTMAS AND HAPPY NEW YEAR");
        }
    }
}
```

Add listener to the button

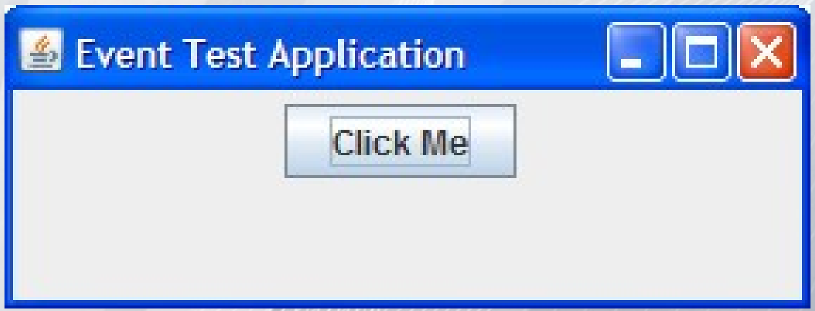
Event

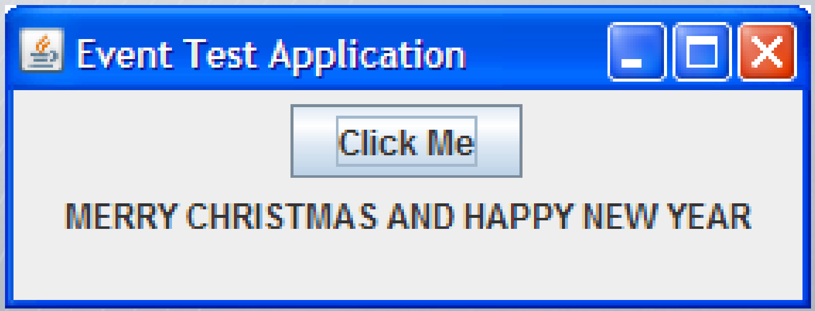
Implement an ActionListener as inner class

Events and listeners

```
public class ChristmasPanelTest {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Test Application");
        frame.setSize(800, 600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        ChristmasPanel p = new ChristmasPanel();
        frame.add(p);
        frame.setVisible(true);
    }
}
```





MERRY CHRISTMAS AND HAPPY NEW YEAR

Events and Listener

- There are many events: XXXEvent

  (`ActionEvent`, `MouseEvent`, `KeyEvent` …)

- For each event there is appropriate listener: XXXListener

  (`ActionListener`, `MouseListener`, `KeyListener ...`)

- Each component can fire specific events and can add the appropriate listerners for them via method:

  addXXXListener(XXXListener l)

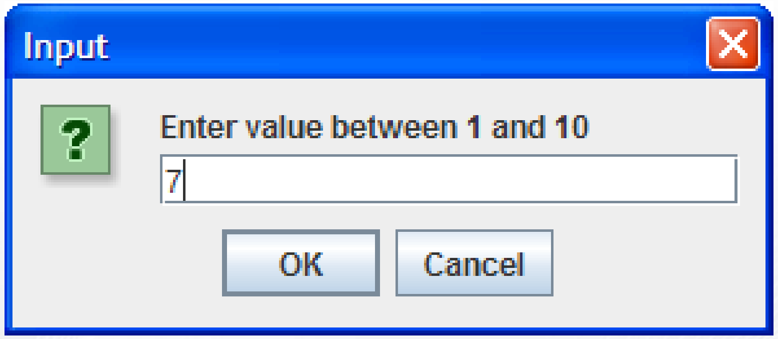
  (`addActionListener(ActionListener l)`,

  `addMouseListener(MouseListener l)`,

  `addKeyListener(KeyListener l)` …)

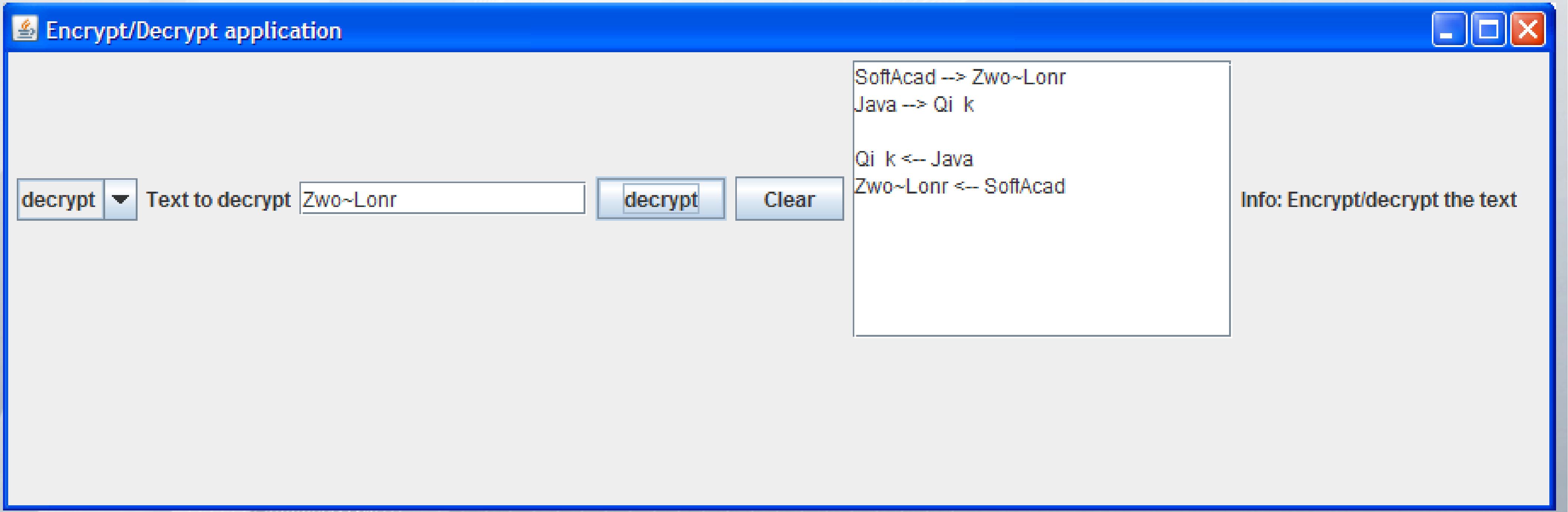
Exercise

Read the task from the pdf file in moodle.



Enter value between 1 and 10

7



Zwo~Lonr

```
SoftAcad --> Zwo~Lonr
Java --> Qi  k

Qi  k <-- Java
Zwo~Lonr <-- SoftAcad
```

Info: Encrypt/decrypt the text