Unit 2: Event handling

- Two event handling mechanisms
- The delegation event model: events, event sources, event listeners

  Events:
  - KeyEvent class:
  - MouseEvent Class,
  - TextEvent class,
  - WindowEvent class
- Sources of Events
- Event Listener Interfaces
  - ActionListener,
  - KeyListener,
  - MouseListener,
  - WindowListener,
  - ContainerListenenr
- Using Delegation event model
  - Handling mouse events,
  - Handling keyboard events
- Adapter classes
- Inner classes

Introduction

- Event is the change in the state of object or source.
- Events are generated as result of user interaction with the graphical user interface components.
- For example,
  - clicking on a button,
  - moving the mouse,
  - entering a character through keyboard,
  - selecting an item from list,
  - scrolling the page are the activities that causes an event to happen.

Types of Event

The events can be broadly classified into two categories:

Foreground Events - Those events which require the direct interaction of user.

- They are generated as consequences of a person interacting with the graphical components in Graphical User Interface.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

Background Events -

- Those events that require the interaction of end user are known as background events.
- Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

Event Handling

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism have the code which is known as event handler that is executed when an event occurs.
- Java Uses the Delegation Event Model(DEM) to handle the events.
- This model defines the standard mechanism to generate and handle the events.

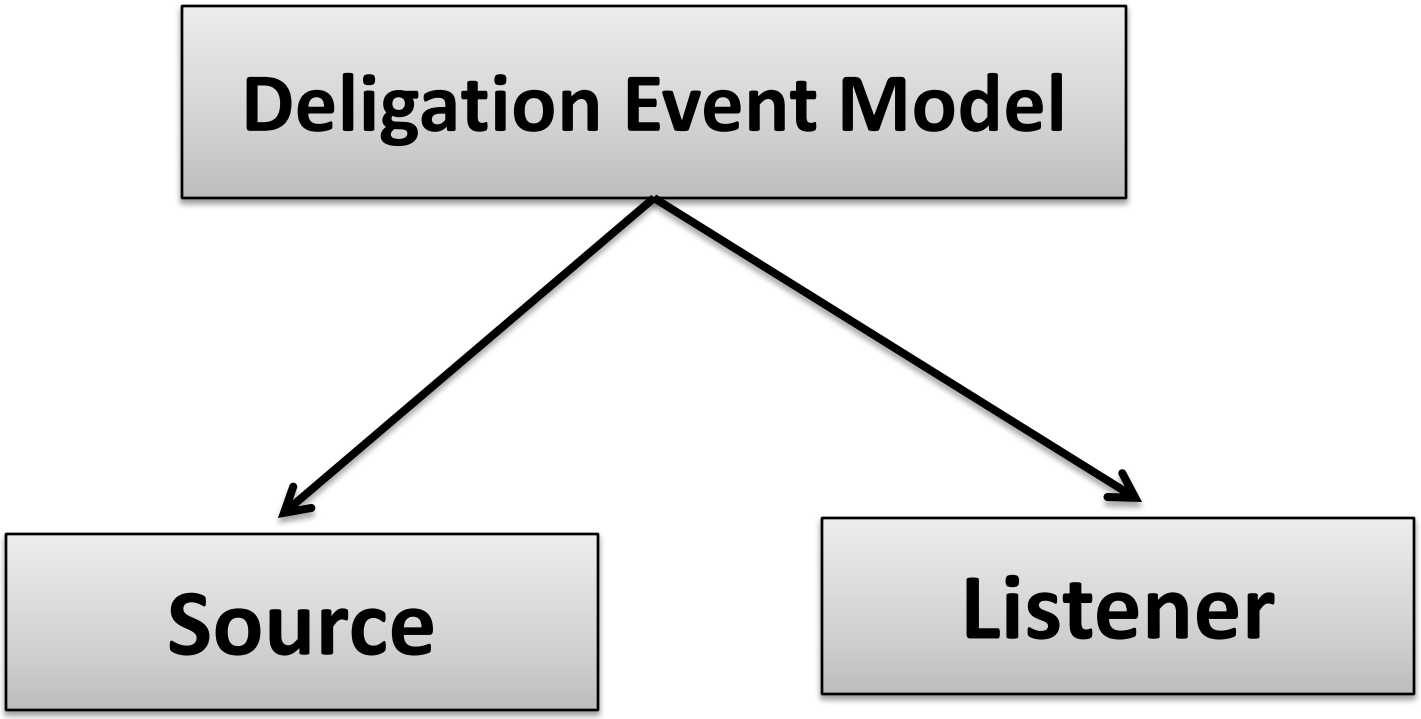
The Delegation Event Model has the following key participants namely:

Source -

- The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler.
- Java provide as with classes for source object.

Listener -

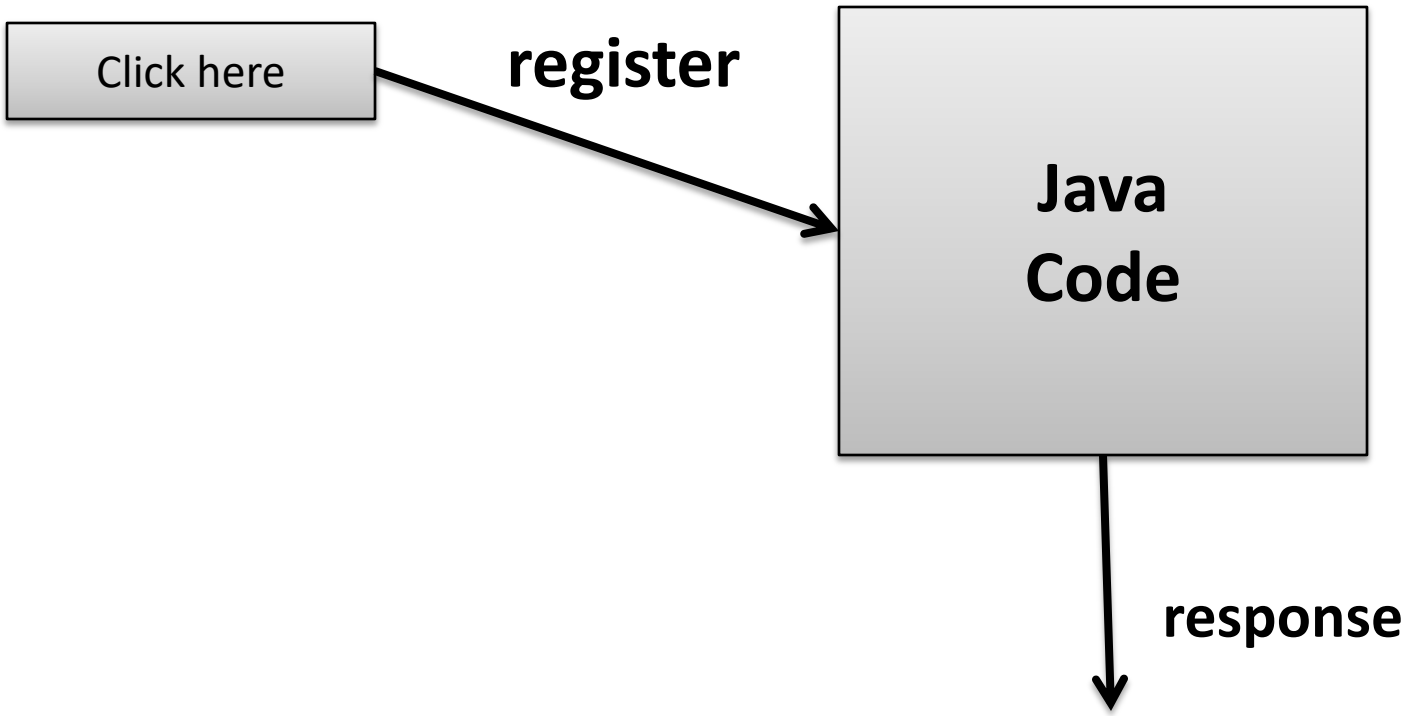
- It is also known as event handler.
- Listener is responsible for generating response to an event.
- From java implementation point of view the listener is also an object.
- Listener waits until it receives an event.
- Once the event is received , the listener process the event an then returns.



Source: Is an object on which event occurs

Listener: Event handler (generates response to event)

Example



- The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event.
- The user interface element is able to delegate the processing of an event to the separate piece of code.
- In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification.
- This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

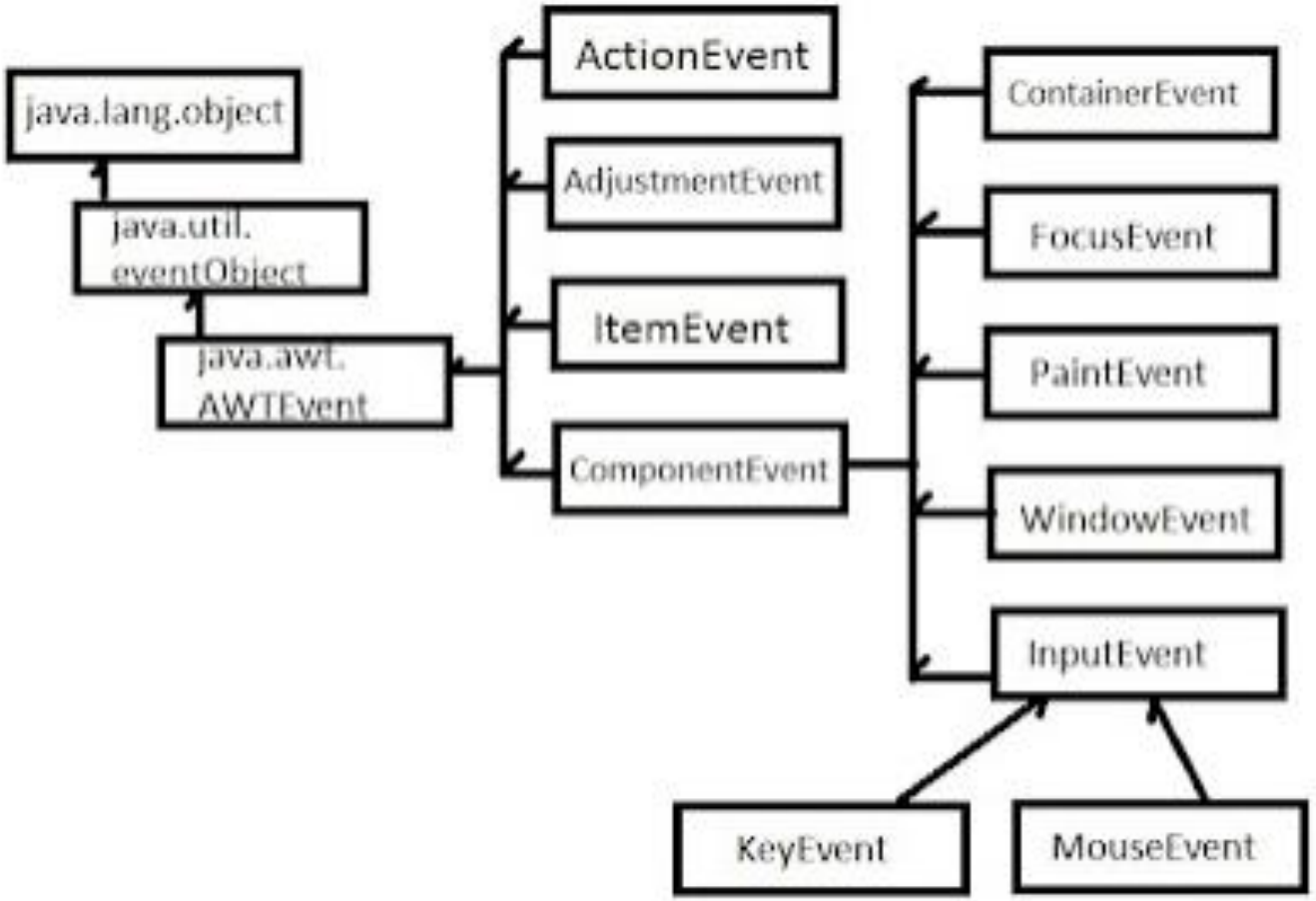
Points to remember about listener

- In order to design a listener class we have to develop some listener interfaces.
- These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.
- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

Callback Methods

- These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer.
- Here the callback methods represents an event method.
- In response to an event java jre will fire callback method.
- All such callback methods are provided in listener interfaces.
- If a component(button, frame..) wants some listener will listen to it's events the the source must register itself to the listener.

AWT InputEvent Class



- The InputEvent class is root event class for all component-level input events.
- Input events are delivered to listeners before they are processed normally by the source where they originated.
- This allows listeners and component subclasses to "consume" the event so that the source will not process them in their default manner.

Class declaration

- Following is the declaration for **java.awt.event.InputEvent** class:

*public abstract class **InputEvent** extends **ComponentEvent***

Event class:

- KeyEvent Class
- MouseEvent Class,
- TextEvent class,
- WindowEvent class

KeyEvent

- On entering the character the Key event is generated. There are three types of key events which are represented by the integer constants. These key events are following
- KEY_PRESSED
- KEY_RELASED
- KEY_TYPED

Fields in KeyEvent Class

- **static int KEY_PRESSED** --The "key pressed" event.
- **static int KEY_RELEASED** --The "key released" event.
- **static int KEY_TYPED** --The "key typed" event.

Class declaration

- Following is the declaration for **java.awt.event.KeyEvent** class:

*public class **KeyEvent** extends **InputEvent***

Constructors in KeyEvent Class

1. **KeyEvent**(Component source, int id, long when, int modifiers, int keyCode**)**

- Deprecated. as of JDK1.1

2. **KeyEvent(**Component source, int id, long when, int modifiers, int keyCode, char keyChar**)**

- Constructs a KeyEvent object.

3. **KeyEvent**(Component source, int id, long when, int modifiers, int keyCode, char keyChar, int keyLocation**)**

Following are the types of methods provided by KeyEvent class

int getKeyCode()

- It is used for getting the integer code associated with a key.
- It is used for KEY_PRESSED and KEY_RELEASED events.
- The keycodes are defined as constants in KeyEvent class

char getKeyChar()

- This method is used to get the Unicode character of the key pressed.
- It works with the KEY_TYPED events

int getKeyLocation()

- Returns the location of the key that originated this key event.

static String getKeyModifiersText(int modifiers)

- Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift".

static String getKeyText(int keyCode)

- Returns a String describing the keyCode, such as "HOME", "F1" or "A"

MouseEvent Class

- This event indicates a mouse action occurred in a component. This low-level event is generated by a component object for Mouse Events and Mouse motion events.
- a mouse button is pressed
- a mouse button is released
- a mouse button is clicked (pressed and released)
- a mouse cursor enters the unobscured part of component's geometry
- a mouse cursor exits the unobscured part of component's geometry
- a mouse is moved
- the mouse is dragged

Class declaration

- Following is the declaration for **java.awt.event.MouseEvent** class:

public class MouseEvent extends InputEvent

Field

- Following are the fields for **java.awt.event.MouseEvent** class:
- **static int BUTTON1** --Indicates mouse button #1; used by getButton()
- **static int BUTTON2** --Indicates mouse button #2; used by getButton()
- **static int BUTTON3** --Indicates mouse button #3; used by getButton()
- **static int MOUSE_CLICKED** --The "mouse clicked" event
- **static int MOUSE_DRAGGED** --The "mouse dragged" event
- **static int MOUSE_ENTERED** --The "mouse entered" event
- **static int MOUSE_EXITED** --The "mouse exited" event
- **static int MOUSE_FIRST** --The first number in the range of ids used for mouse events
- **static int MOUSE_LAST** -- The last number in the range of ids used for mouse events
- **static int MOUSE_MOVED** --The "mouse moved" event
- **static int MOUSE_PRESSED** -- The "mouse pressed" event
- **static int MOUSE_RELEASED** --The "mouse released" event
- **static int MOUSE_WHEEL** --The "mouse wheel" event
- **static int NOBUTTON** --Indicates no mouse buttons; used by getButton()
- **static int VK_WINDOWS** --Constant for the Microsoft Windows "Windows" key.

MouseEvent Constructors

1. **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger)**

Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, and click count.

2. **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger, int button)**

Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, and click count.

3. **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int xAbs, int yAbs, int clickCount, boolean popupTrigger, int button)**

Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, absolute coordinates, and click count.

Class methods

1. int getButton()

Returns which, if any, of the mouse buttons has changed state.

2. **int getClickCount()**

Returns the number of mouse clicks associated with this event.

3. **Point getLocationOnScreen()**

Returns the absolute x, y position of the event.

4. **static String getMouseModifiersText(int modifiers)**

Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".

5. **Point getPoint()**

Returns the x,y position of the event relative to the source component.

6. **int getX()**

Returns the horizontal x position of the event relative to the source component.

Contd..

7. **int getXOnScreen()**

Returns the absolute horizontal x position of the event.

8. **int getY()**

Returns the vertical y position of the event relative to the source component.

9. **int getYOnScreen()**

Returns the absolute vertical y position of the event.

10. **boolean isPopupTrigger() Returns whether or not this mouse event is the popup menu trigger event for the platform.**

11. **String paramString()**

Returns a parameter string identifying this event.

12. **void translatePoint(int x, int y)**

Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.

Mouse Listener/Mouse Motion Listener

MouseListener (interface)

1. mouseClicked()
2. mouseEntered()
3. mouseExited()
4. mouseReleased()
5. mousePressed()

MouseMotionListener(interface)

6. mouseMoved()
7. mouseDragged()

Key Listener

- Key events in java are generated when a key is pressed, the key is typed and a key is released. The key Listener interface is handling key events.
- Each of the key event handling methods takes a keyEvent object is its arguments.
- A key event object contains information about the key event. The various methods available in the KeyListener interface are as follows:

void keyPressed(keyEvent e)

- The keyPressed ()method is used to call when a key is pressed.

void keyTyped(keyEvent e)

- The keyTyped ()method is used to call when a key is typed.

void KeyReleased(keyEvent e)

The keyReleased() method is used to call when a key is released.

WindowEvent

- The object of this class represents the change in state of a window.
- This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transfered into or out of the Window.

Class declaration

- Following is the declaration for **java.awt.event.WindowEvent** class:
- public class WindowEvent extends ComponentEvent

Field

Following are the fields for **java.awt.event.WindowEvent** class:

- **static int WINDOW_ACTIVATED** --The window-activated event type.
- **static int WINDOW_CLOSED** -- The window closed event.
- **static int WINDOW_CLOSING** -- The "window is closing" event.
- **static int WINDOW_DEACTIVATED** -- The window-deactivated event type.
- **static int WINDOW_DEICONIFIED** -- The window deiconified event type.
- **static int WINDOW_FIRST** -- The first number in the range of ids used for window events.
- **static int WINDOW_GAINED_FOCUS** -- The window-gained-focus event type.
- **static int WINDOW_ICONIFIED** -- The window iconified event.
- **static int WINDOW_LAST** -- The last number in the range of ids used for window events.
- **static int WINDOW_LOST_FOCUS** -- The window-lost-focus event type.
- **static int WINDOW_OPENED** -- The window opened event.
- **static int WINDOW_STATE_CHANGED** -- The window-state-changed event type.

Constructors

1. **WindowEvent(Window source, int id)**
- Constructs a WindowEvent object.
2. **WindowEvent(Window source, int id, int oldState, int newState)**
- Constructs a WindowEvent object with the specified previous and new window states.
3. **WindowEvent(Window source, int id, Window opposite)**
- Constructs a WindowEvent object with the specified opposite Window.
4. **WindowEvent(Window source, int id, Window opposite, int oldState, int newState)**
- Constructs a WindowEvent object.

methods

1. **int getNewState()**
- For WINDOW_STATE_CHANGED events returns the new state of the window.
2. **int getOldState()**
- For WINDOW_STATE_CHANGED events returns the previous state of the window.
3. **Window getOppositeWindow()**
- Returns the other Window involved in this focus or activation change.
4. **Window getWindow()**
- Returns the originator of the event.
5. **String paramString()**
- Returns a parameter string identifying this event.

Java WindowListener Interface

- The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent.
- The WindowListener interface is found in java.awt.event package..

Methods of WindowListener interface

The signature of 7 methods found in WindowListener interface are given below:

- **public abstract void** windowActivated(WindowEvent e);
- **public abstract void** windowClosed(WindowEvent e);
- **public abstract void** windowClosing(WindowEvent e);
- **public abstract void** windowDeactivated(WindowEvent e);
- **public abstract void** windowDeiconified(WindowEvent e);
- **public abstract void** windowIconified(WindowEvent e);
- **public abstract void** windowOpened(WindowEvent e);

TextEvent class

Homework

- Explain Text Event handling in java with suitable examples.

JAVA Adapter Class

- Java adapter classes *provide the default implementation of listener interfaces*.
- If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

Pros of using Adapter classes:

- It assists the unrelated classes to work combinedly.
- It provides ways to use classes in different ways.
- It increases the transparency of classes.
- It provides a way to include related patterns in the class.
- It provides a pluggable kit for developing an application.
- It increases the reusability of the class.

The adapter classes are found in **java.awt.event, java.awt.dnd** and **javax.swing.event** packages.

- The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java Inner Class

- **Java inner class** or nested class is a class that is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.
- Additionally, it can access all the members of the outer class, including private data members and methods.

Syntax

```
class Java_Outer_class
{
 //code
 class Java_Inner_class
     {
       //code
      }
}
```

Advantages

- Nested classes represent a particular type of relationship that is **it can access all the members (data members and methods) of the outer class,** including private.
- Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- **Code Optimization**: It requires less code to write.

Need of Java Inner class

- Sometimes users need to program a class in such a way so that no other class can access it. Therefore, it would be better if you include it within other classes.
- If all the class objects are a part of the outer object then it is easier to nest that class inside the outer class. That way all the outer class can access all the objects of the inner class.

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing an interface or extending class. The java compiler decides its name.
Local Inner Class	A class was created within the method.
Static Nested Class	A static class was created within the class.
Nested Interface	An interface created within class or interface.

Homework.

Explain following with suitable examples.

- Nested Inner Class
- Method Local Inner Classes
- Static Nested Classes
- Anonymous Inner Classes