Sabancı Üniversitesi

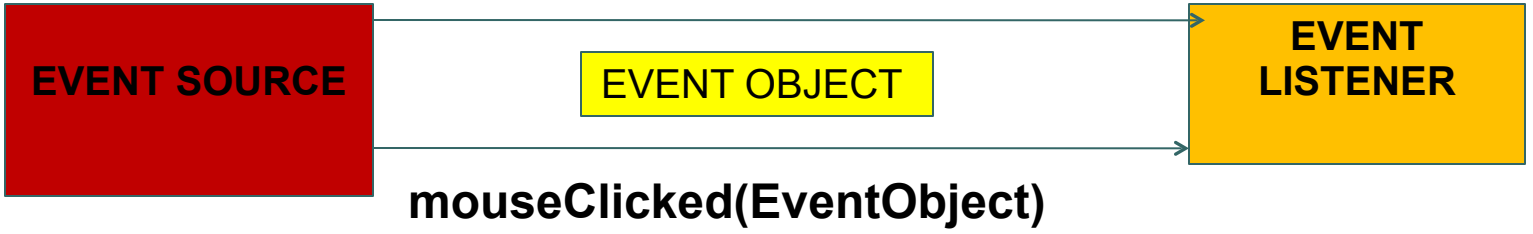
Pre Android UI

09 – Java Events

Sabancı Üniversitesi

Events

- Event mechanism in Java follows the principles of the *Observer Design Pattern,* which has four rules:

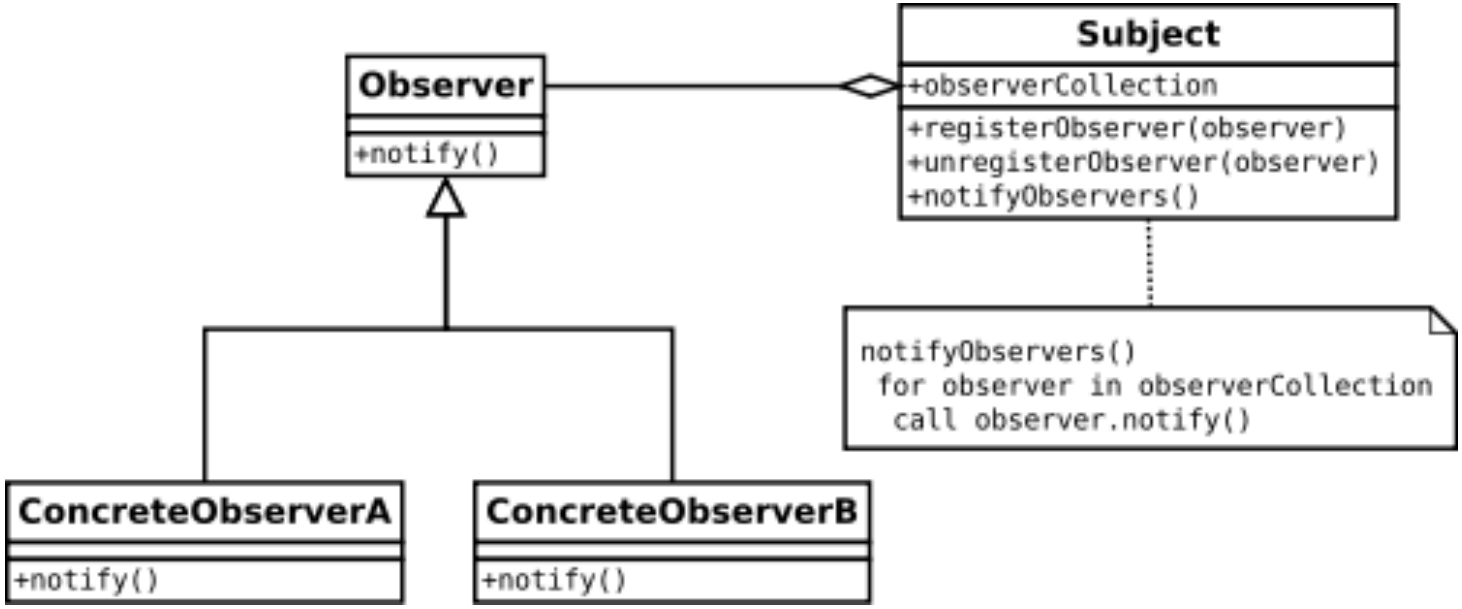


- event source → triggers the event
- event listener → is a registered interested party which receives the event (is notified)
- event listener interface → forms the contract between source and listeners (is implemented by listeners)
- event object → object representing the event (contains information about the event)

Sabancı Üniversitesi

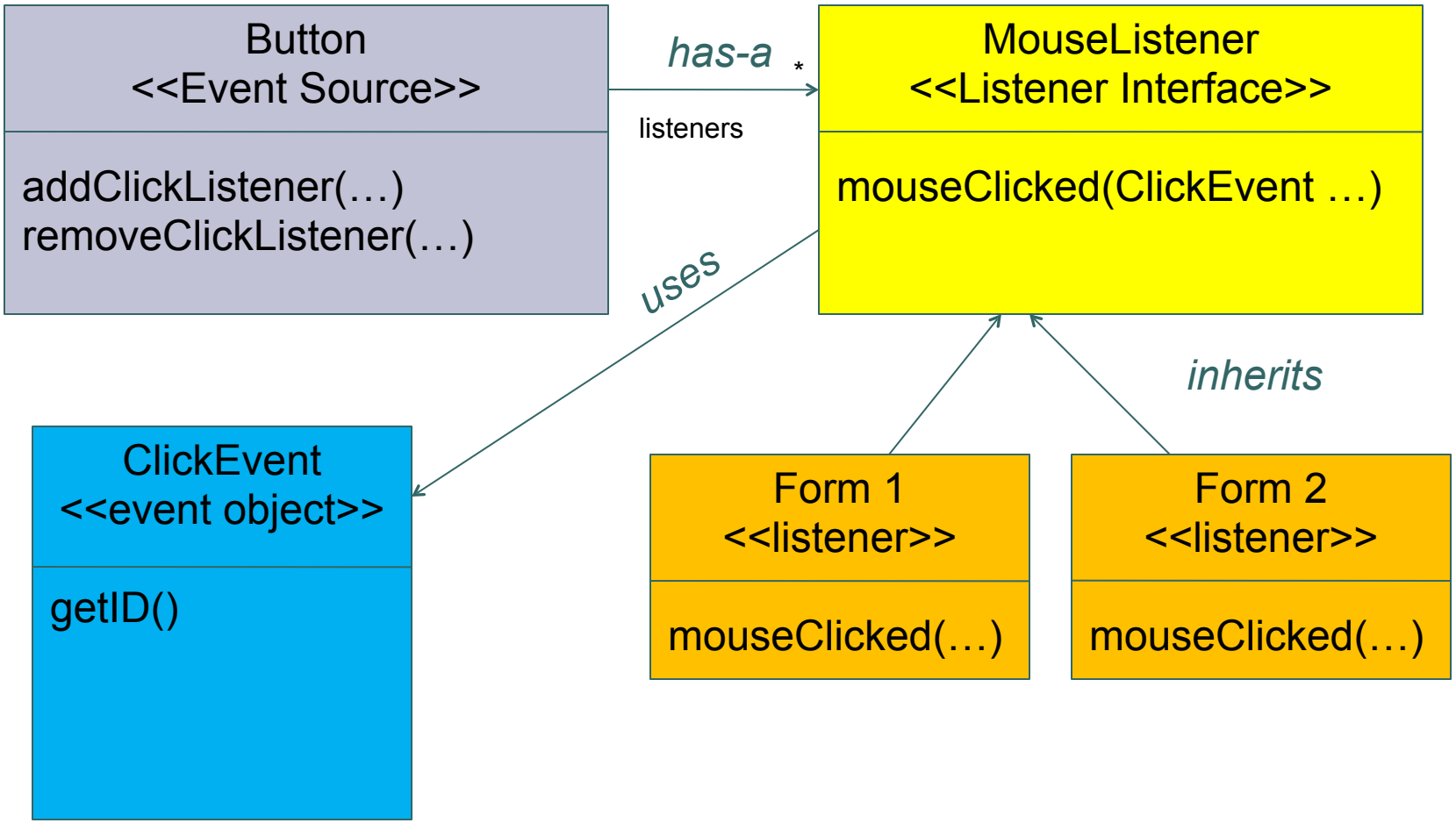
The Observer Pattern

- Button: **Subject**, who informs that event has happened, i.e. click
- UI, form: **Observers**, they are informed by the subject when the event occurs, i.e. button clicked, display another page



Sabancı Üniversitesi

An Example with UML



Sabancı Üniversitesi

The Listener Interface

- Event listener interface defines the contract between source and the listeners
  - must extend `java.util.EventListener`

```
public interface MouseListener extends EventListener{...}
```

  - contains groups of related event handling methods

```
public interface MouseListener extends EventListener{
        public mouseClicked(…);
}
```

Sabancı Üniversitesi

The Listener Interface

- Methods should have a single argument which is subclass of `java.util.EventObject`

```
public interface MouseListener extends EventListener{
        public mouseClicked(MouseClickEvent evt);
}
```

- Methods may throw checked exceptions

Sabancı Üniversitesi

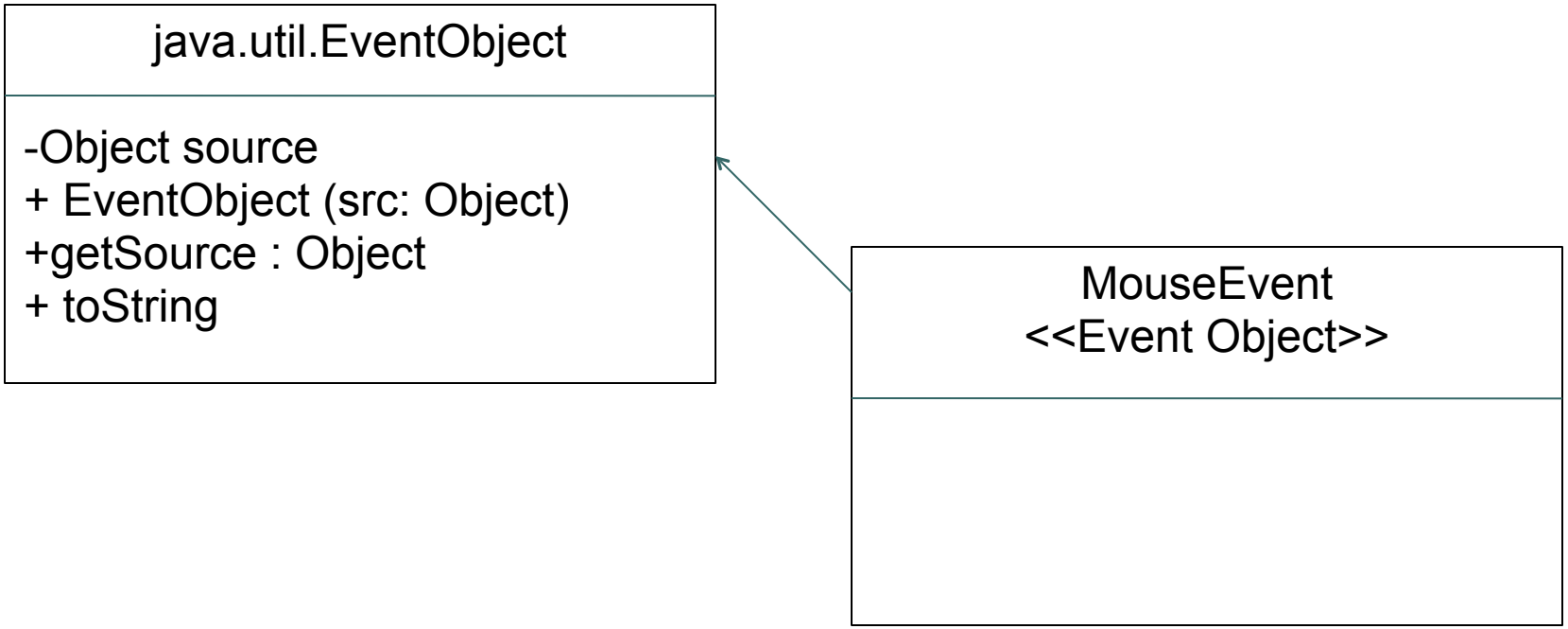
Listener Adapter

- Consider supplying an event adapter with your listener interface - a convenient base class with empty implementation

```
public class MouseListenerAdapter implements MouseListener{
        public mouseClicked(ClickEvent evt){
                //nothing
        }
}
```

The Event Object

- It is the object for wrapping and carrying the information regarding the event fired
- must extend `java.util.EventObject`



Sabancı Üniversitesi

The Event Object

- The event object typically carries additional information about the event
- It must be immutable

```
public class ButtonClickEvent extends EventObject {
          private int id;
          private Date date;
          public ButtonClickEvent(Object obj, int id, Date date) {
                    super(obj);
                    this.id = id;
                    this.date = date;
          }
          public int getId() {
                    return id;
          }
          public void setId(int id) {
                    this.id = id;
          }
          public Date getDate() {
                    return date;
          }
          public void setDate(Date date) {
                    this.date = date;
          }
}
```

Sabancı Üniversitesi

The Event Source

- The event source triggers the event and notifies the listeners
- It does not need to implement or extend anything particular

```
public class Button {
        private ArrayList<ButtonClickListener> clickListeners = new
ArrayList<ButtonClickListener>();
        public int id;
        public Button(int id) {
                this.id = id;
        }
        public void click(){
                for (ButtonClickListener listener : clickListeners) {
                        listener.button_Clicked(new ButtonClickEvent(this,
                                        this.id, (new Date())));
                }
        }
}
```

Sabancı Üniversitesi

The Event Source

- The event source requires methods for registering and de-registering events

```
public void addClickListener(ButtonClickListener listener){
        clickListeners.add(listener);
        }
public void removeClickListener(ButtonClickListener listener){
        clickListeners.remove(listener);
        }
```

Sabancı Üniversitesi

The Event Source

- When triggering the event keep multi-threading in mind
  - During the event dispatching other threads may register/ unregister, therefore this will not work
  - A solution is using Vector (thread safe) to keep your listeners

```
Private Vector<MouseListener> listeners = new Vector<MouseListener>();
public void click(){
                for (ButtonClickListener listener : clickListeners) {
                listener.button_Clicked(new ButtonClickEvent(this,
                                this.id, (new Date())));
                }
        }
```

Sabancı Üniversitesi

The Event Source

- Can use synchronized methods
- When triggering the event, an EventObject is instantiated and the listener's related method is invoked

```
private List<MouseListener> listeners = new ArrayList<MouseListener>();
public synchronized void addClickListener(ButtonClickListener listener){
clickListeners.add(listener);
          }
public synchronized void removeClickListener(ButtonClickListener listener){
          clickListeners.remove(listener);
          }
public void click(){
          ButtonClickEvent evt = new ButtonClickEvent(this,…);
          MouseListener[] copy;
          synchronized(this){
                    copy = listeners.toArray(new MouseListener[0]);
          }//Make a copy while noe one can add or remove listeners
                    for (ButtonClickListener listener: copy) {
                              listener.button_Clicked(new ButtonClickEvent(this,
                                        this.id, (new Date())));
                    }
          }
```

Sabancı Üniversitesi

The Event Source

- You can use a `CopyOnWriteArrayList` (thread safe variant of ArrayList) to store listeners

```
private List<MouseListener> listeners = new CopyOnWriteArrayList<MouseListener>();
```

Sabancı Üniversitesi

Event Listeners

- Any class can be an event listener, it should just implement the listener interface or a subclass of it

```
public class Form implements ButtonClickListener {
          private Button myButton = new Button(1);

          public Form() {
                    myButton.addClickListener(this);
          }
          @Override
          public void button_Clicked(ButtonClickEvent evt) {
                    System.out.println("Button clicked with id:" + evt.getId() + " at "
+ evt.getDate().toString());

          }

}
```

- The listener needs to register itself with the source