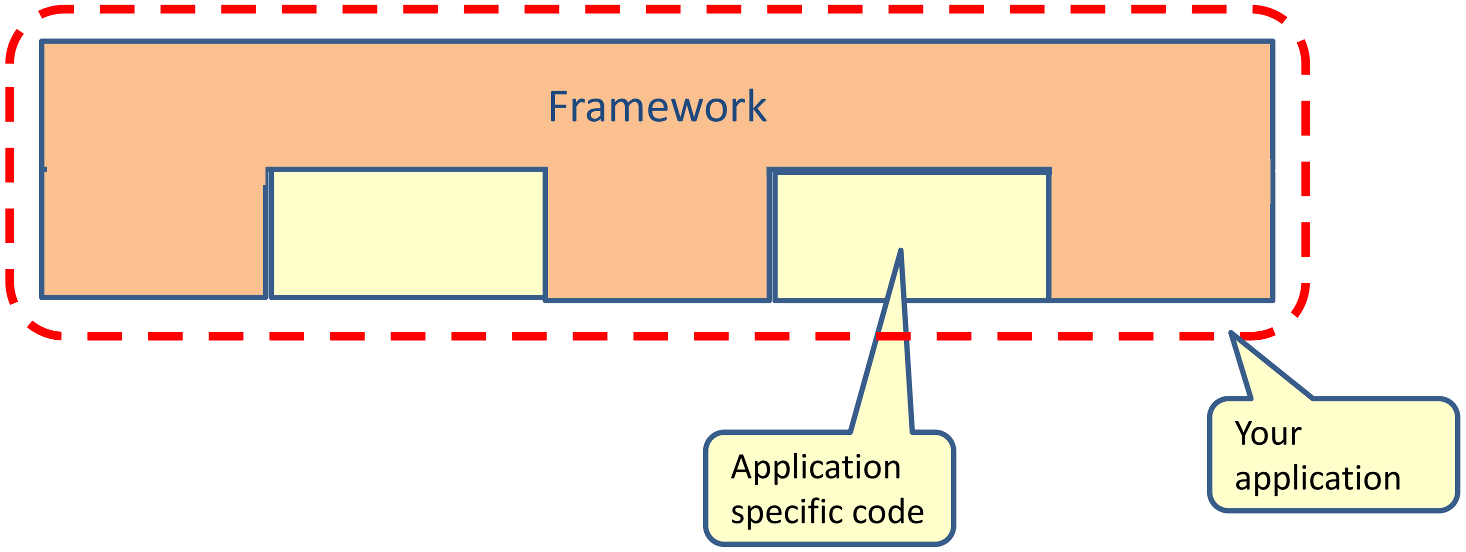
LESSON 11
FRAMEWORK DESIGN

Framework

- A framework is a reusable semi complete application for a specific domain



Framework examples

Web frameworks

- SpringMVC
- Angular
- React
- Vue
- …

ORM frameworks

- Hibernate
- Open JPA
- EclipseLink
- …

Testing frameworks

- JUnit
- TestNG
- Mockito
- RestAssured
- Cucumber
- …

Spring related frameworks

- Spring
- Spring boot
- Spring security
- …

Logging frameworks

- Log4J 2
- LogBack
- SLF4J
- …

Game engine/frameworks

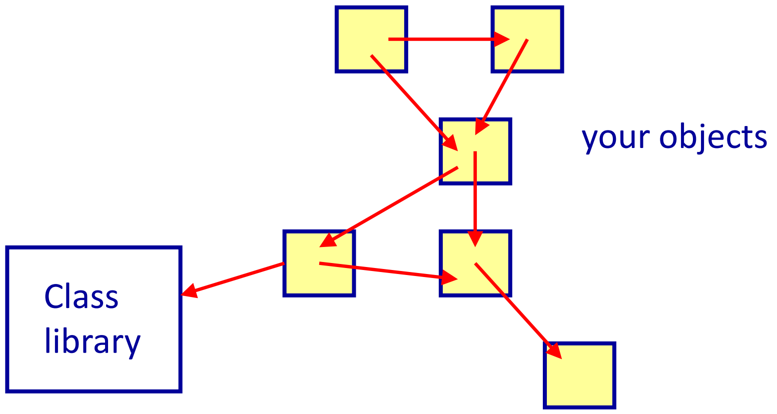
- Unreal Engine
- Unity
- Godot
- …

Why frameworks

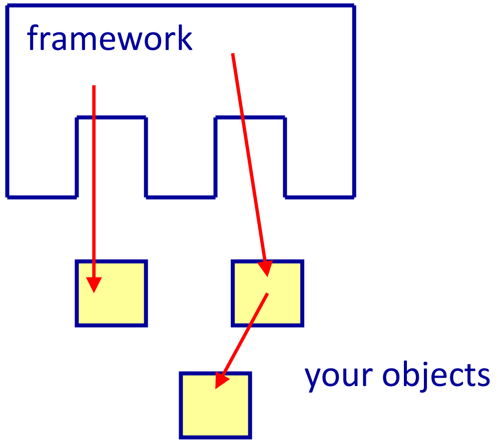
- They embody expertise
  - Developer can focus on the problem domain
- Reuse
- Reliability
  - Reusing a stable framework increases reliability.
- Standardization

Framework vs. Library

- Inversion of Control (IoC)
  - Hollywood principle: Don't call us, we'll call you
  - The framework has control over your code

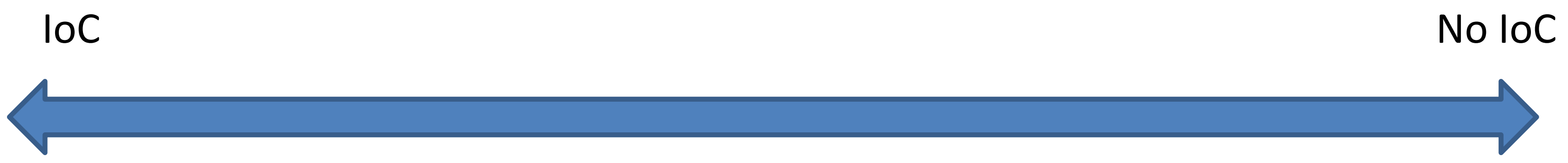


Your code calls the class library



IoC: The framework calls your code

Inversion of Control (IoC)



- JUnit
- SpringMVC
- JavaFX

- Spring

- Java collection framework
- Hibernate
- JDBC

- Framework
- Library
- Platform
- Engine

Example of unit testing

```java
public class Counter {
    private int counterValue=0;

    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return --counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
}
```

```java
import static org.junit.Assert.*;
import org.junit.*

public class CounterTest {
    private Counter counter;

     @Before
     public void setUp() throws Exception {
       counter = new Counter();
     }
     @Test
     public void testIncrement(){
        assertEquals("Counter.increment does not work correctly",1,counter.increment());
        assertEquals("Counter.increment does not work correctly",2,counter.increment());
     }
     @Test
     public void testDecrement(){
        assertEquals("Counter.decrement does not work correctly",-1,counter.decrement());
        assertEquals("Counter.decrement does not work correctly",-2,counter.decrement());
     }
}
```

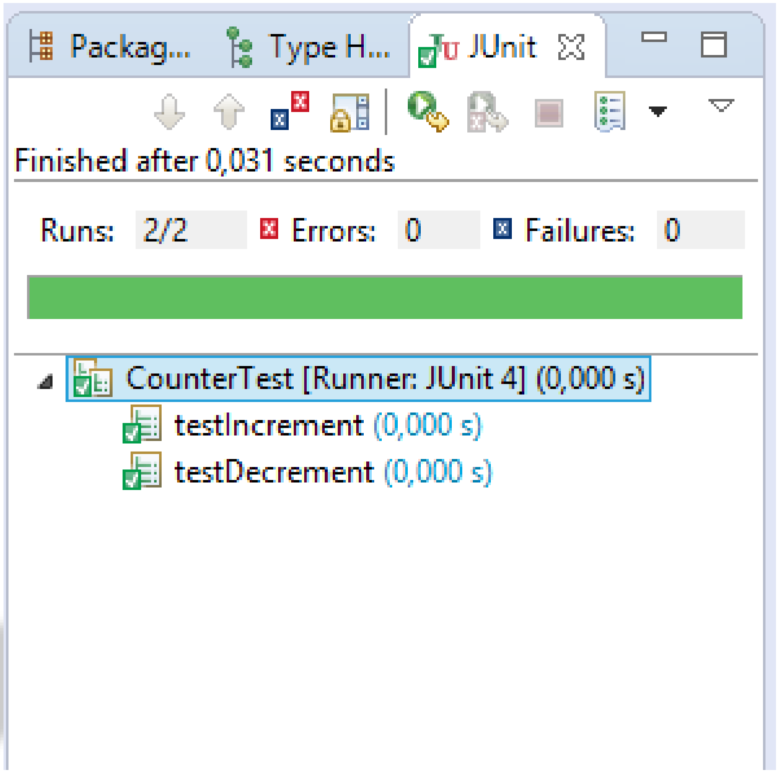
Initialization

Test method

Test method

Running the test

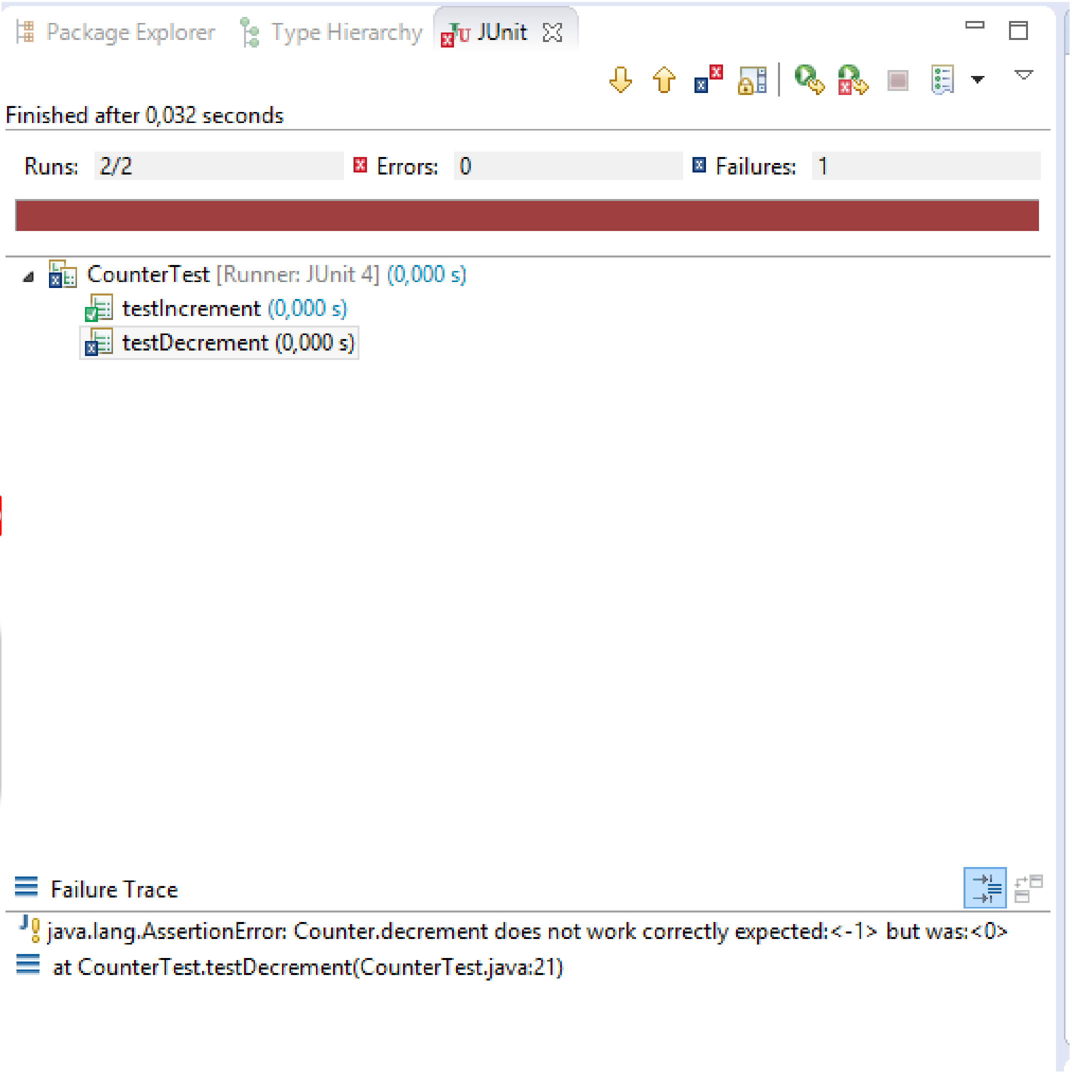
```java
package count;

public class Counter {
    private int counterValue=0;

    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return --counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
}
```



```
Finished after 0,031 seconds

Runs: 2/2    Errors: 0    Failures: 0

CounterTest [Runner: JUnit 4] (0,000 s)
    testIncrement (0,000 s)
    testDecrement (0,000 s)
```

Running the test

```
package count;

public class Counter {
    private int counterValue=0;

    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
}
```



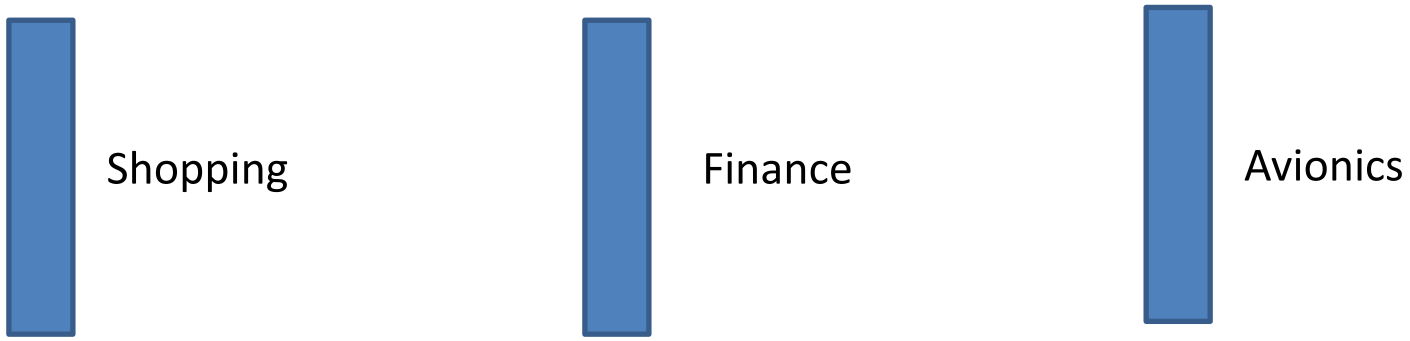
```
Finished after 0,032 seconds
Runs: 2/2    Errors: 0    Failures: 1

CounterTest [Runner: JUnit 4] (0,000 s)
    testIncrement (0,000 s)
    testDecrement (0,000 s)

Failure Trace
java.lang.AssertionError: Counter.decrement does not work correctly expected:<-1> but was:<0>
at CounterTest.testDecrement(CounterTest.java:21)
```

Framework classification

- Technical frameworks (horizontal frameworks)

Testing
GUI
Logging

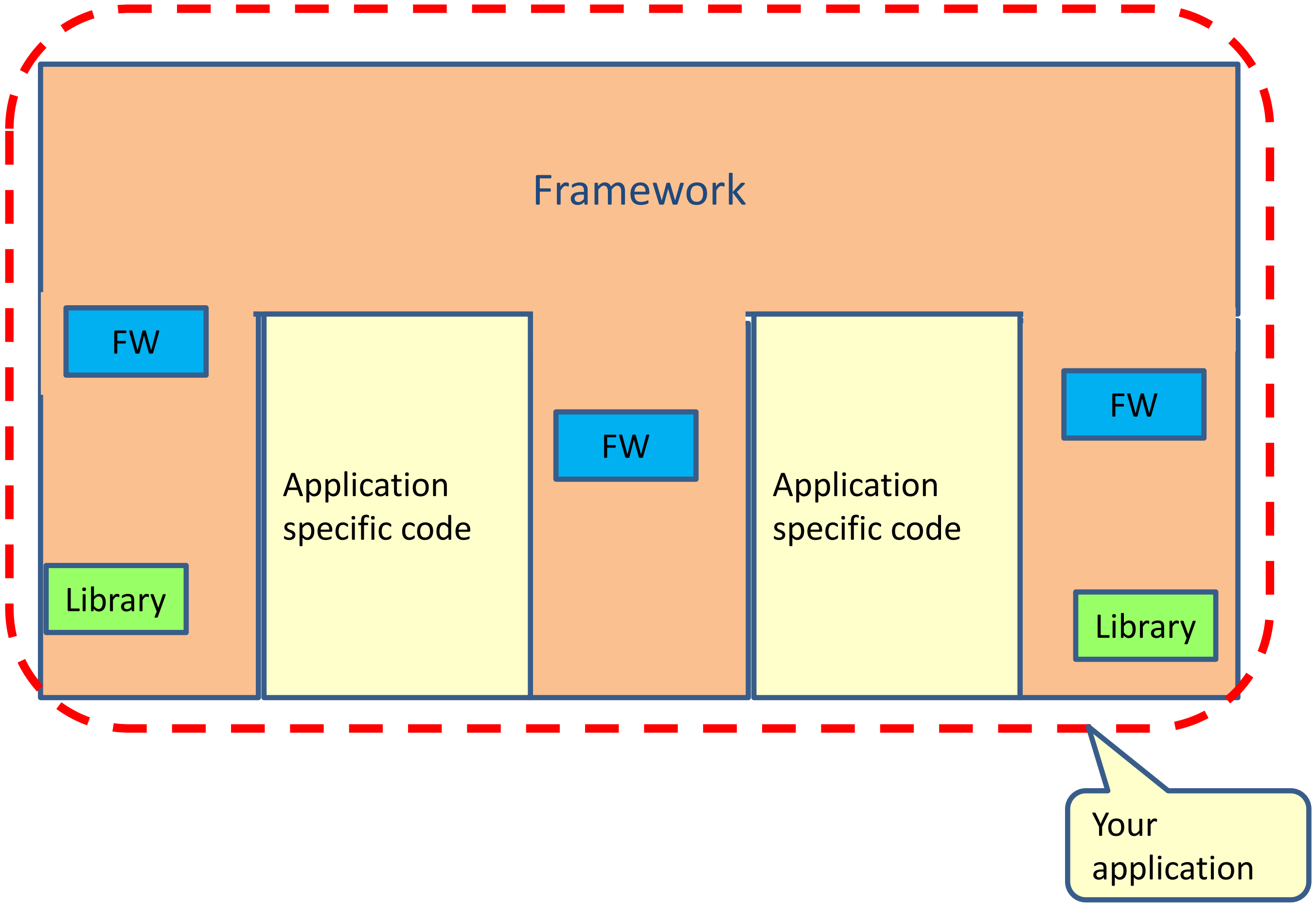
- Business domain frameworks (vertical frameworks)



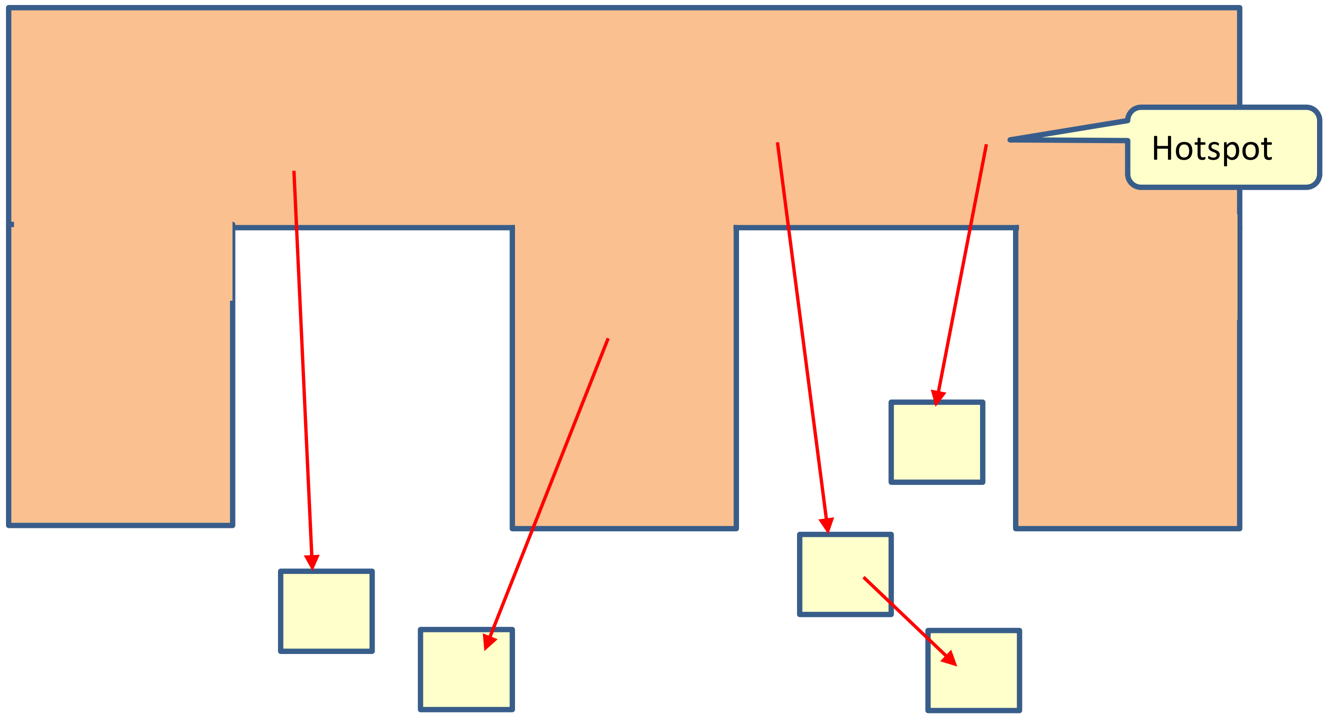
Characteristics of reuse

- To make something generic is 3 to 10 times more expensive than to make something specific
- High risk
- Is everyone aware that this framework exist?
- A framework is a product
  - That need documentation and tests
  - That need maintenance (project, budget)

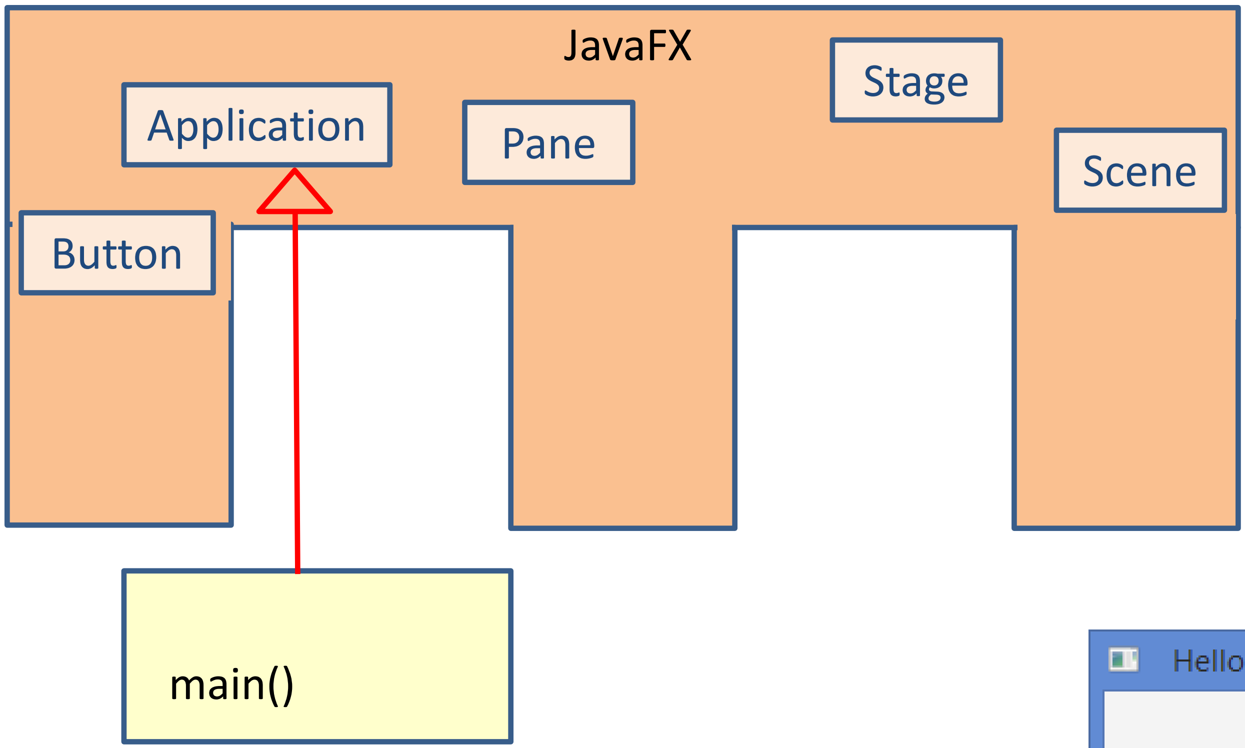
Frameworks + libraries

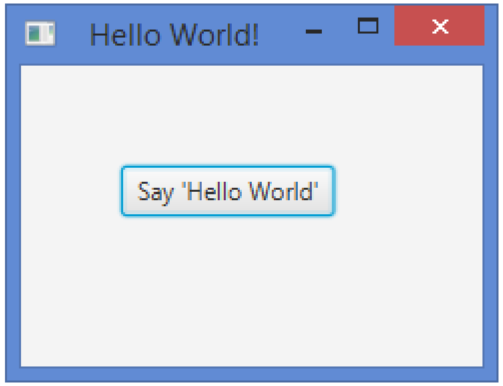


Hotspot (plugin point)



Using JavaFX framework





Using JavaFX framework

```
public class HelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button button = new Button();
        button.setText("Say 'Hello World'");
        button.relocate(50, 50);
        button.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        Pane root = new Pane();
        root.getChildren().add(button);

        Scene scene = new Scene(root, 230, 150);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Stage

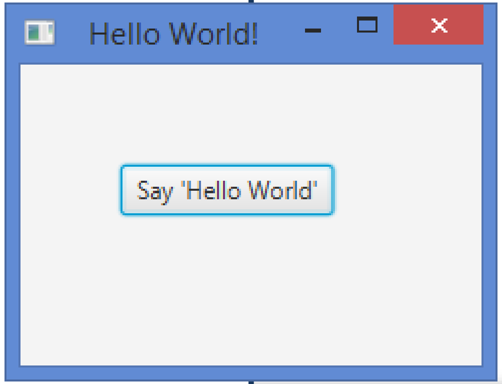
Extend Application

Eventhandler for the button

Pane

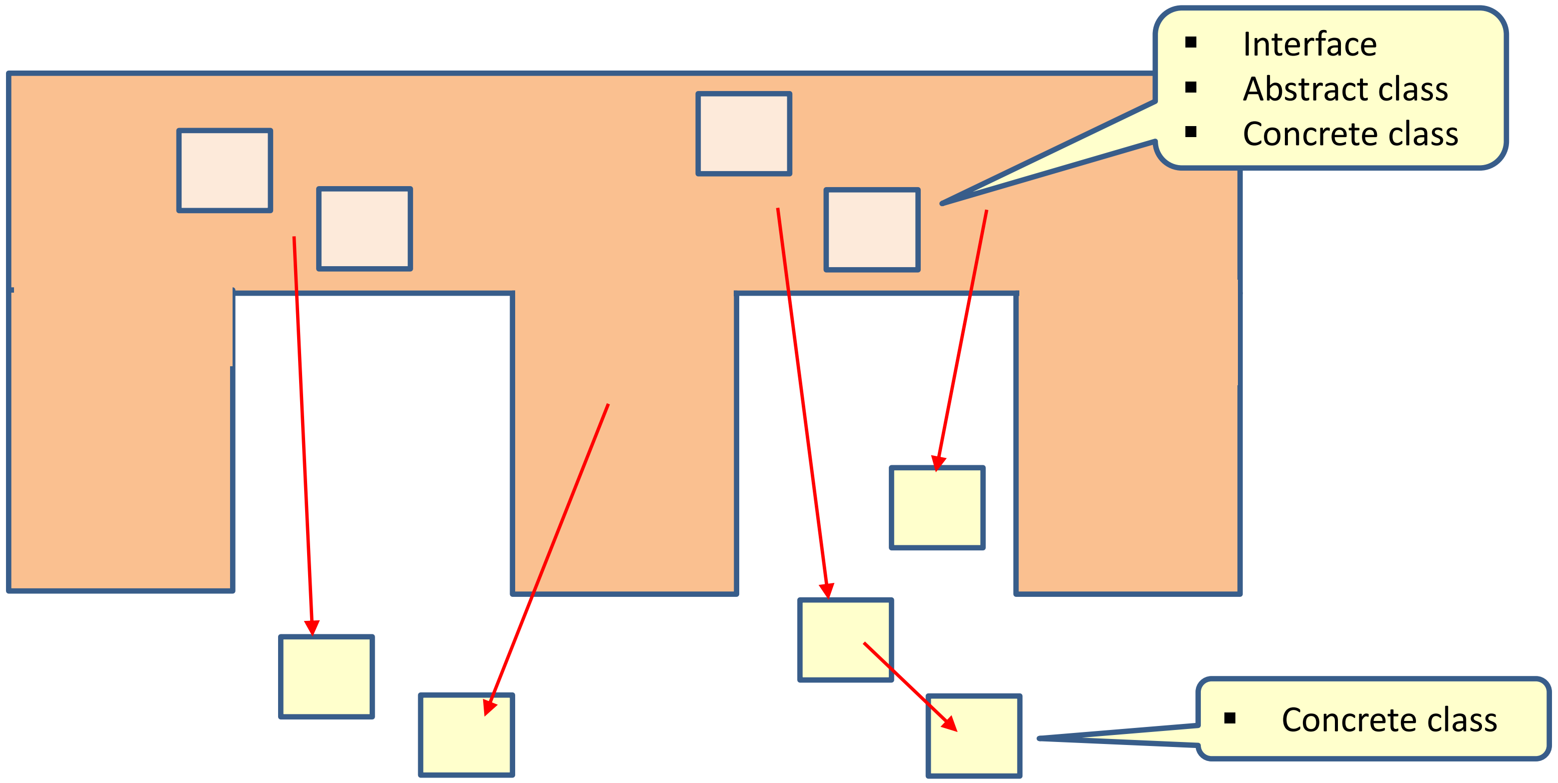
Scene

Launch the application



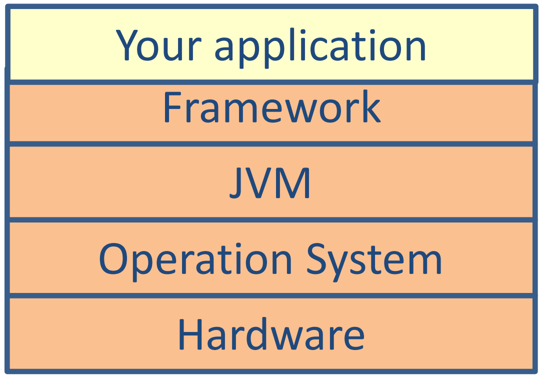
```
Hello World!
```

Framework implementation



Disadvantage of frameworks

- Another layer of abstraction
  - You don’t know the internal details of the FW
    - The framework can contain errors
- Steep learning curve

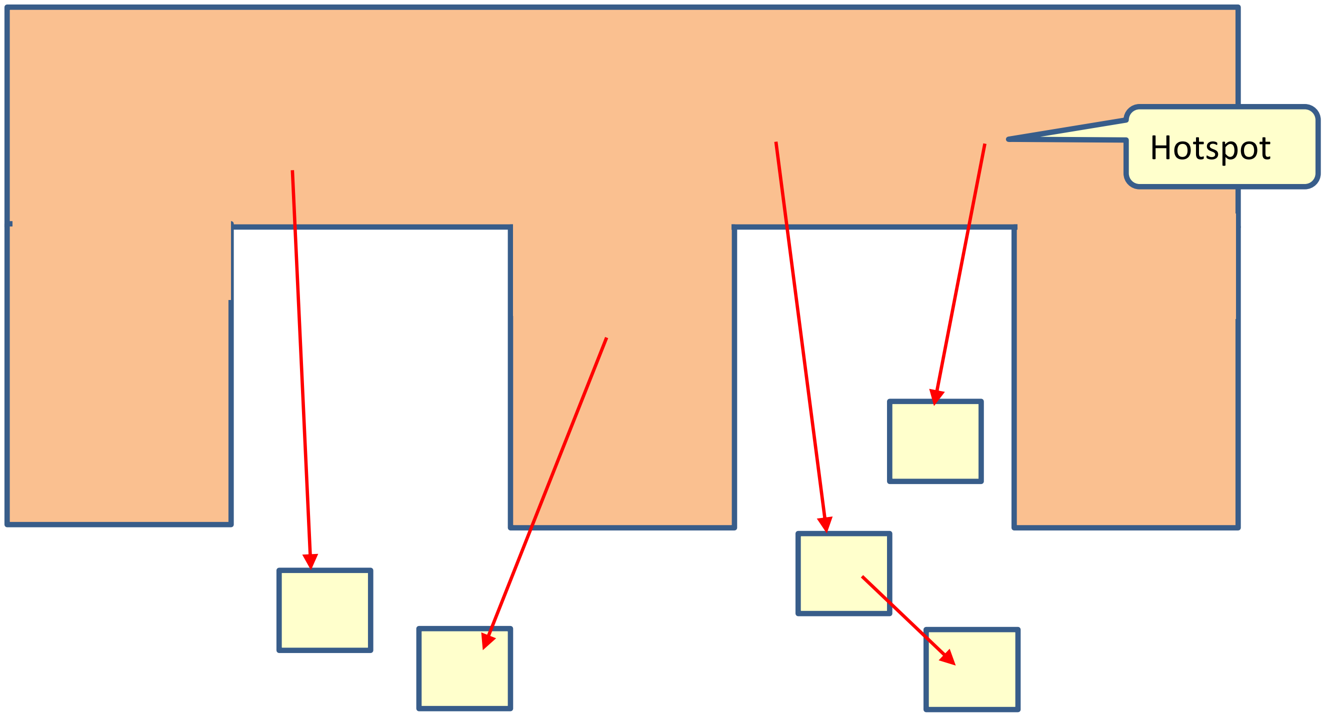


Main point

- Application development is much easier and faster when you reuse a framework rather than writing the application from scratch.

- Life is much easier, simpler and enjoyable if you make use of the framework of Nature, the Unified Field of all the laws of nature. Established in being, perform action.

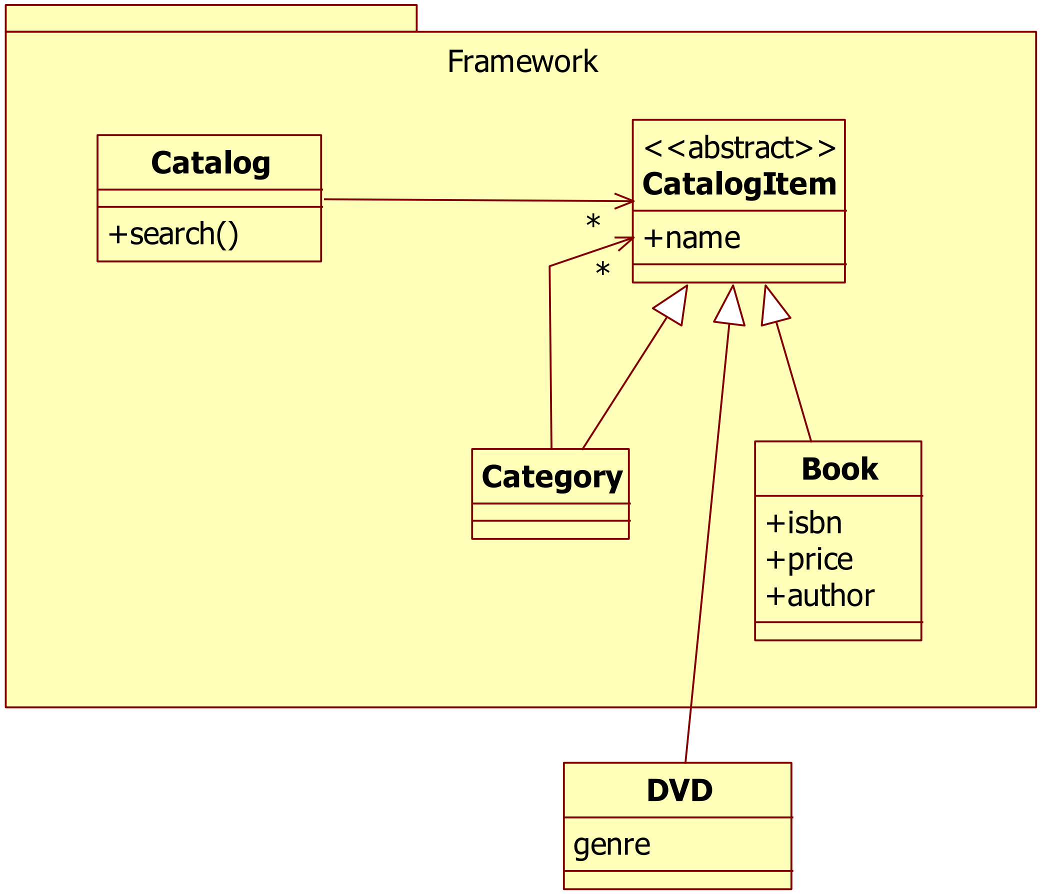
Hotspot (plugin point)



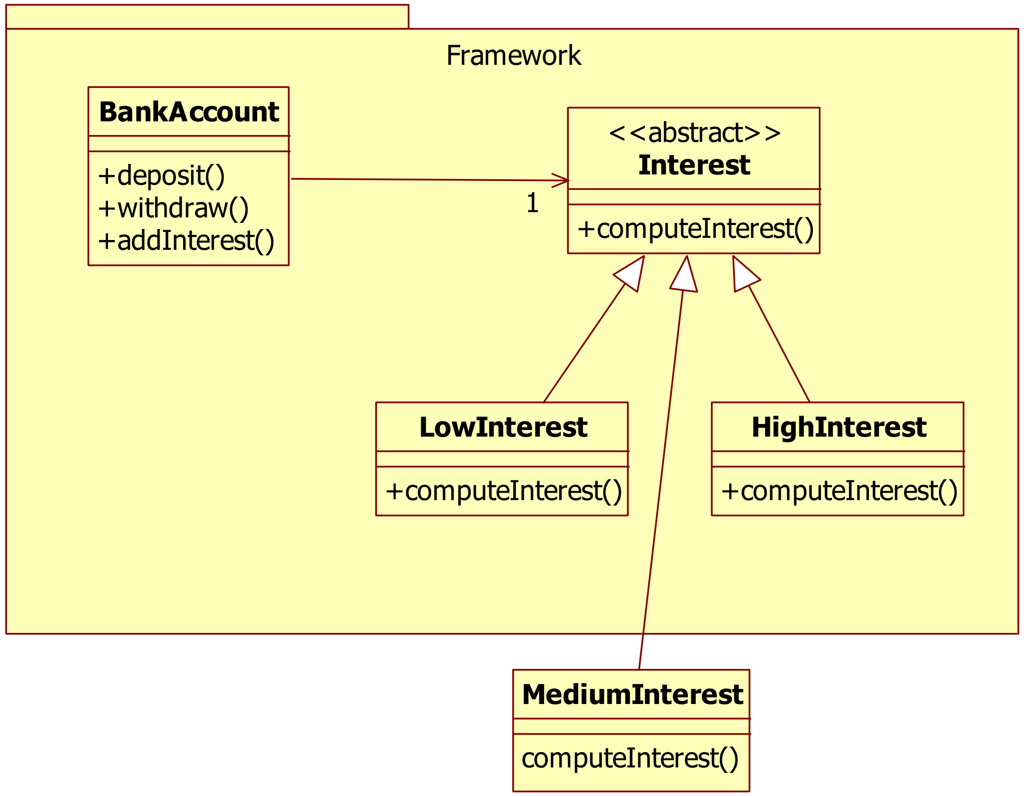
How to make hotspots?

- Plugin new algorithms
  - Strategy pattern, Chain of responsibility pattern
- Plugin new state behavior
  - State pattern
- Plugin new listeners
  - Observer pattern
- Translate between your code and FW code
  - Adapter pattern
- Plugin new actions
  - Command pattern
- Plugin new traversal algorithm
  - Iterator pattern
- Create new objects
  - Factory
- Add classes to a tree structure
  - Composite pattern

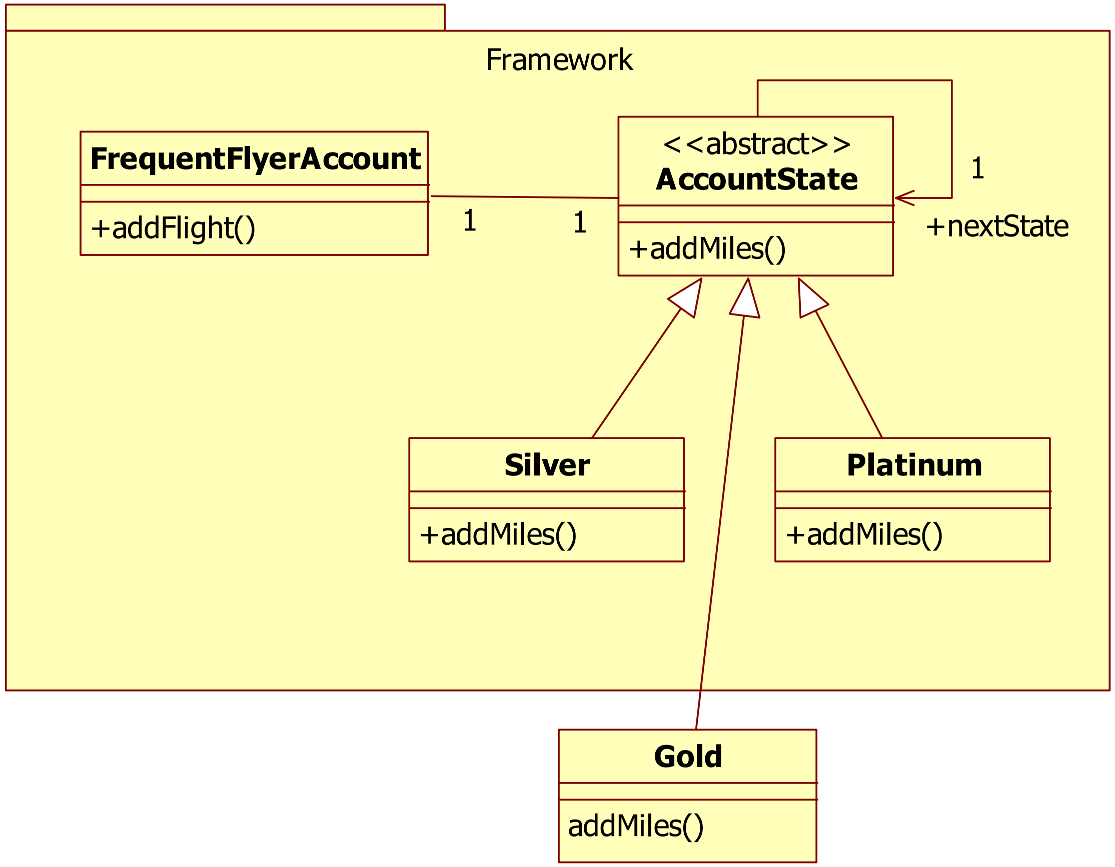
Plugin points: Composite pattern



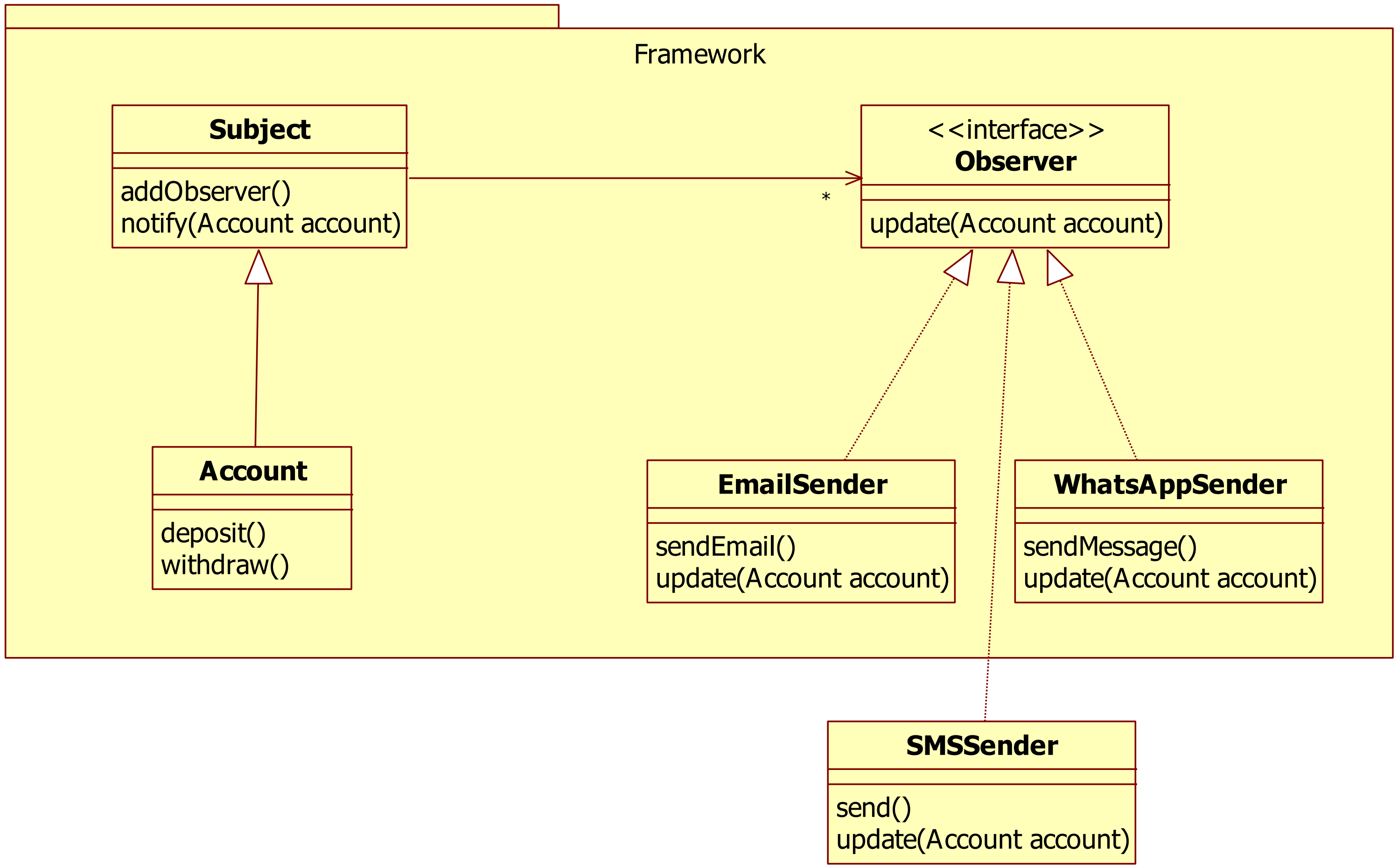
Plugin points: Strategy pattern



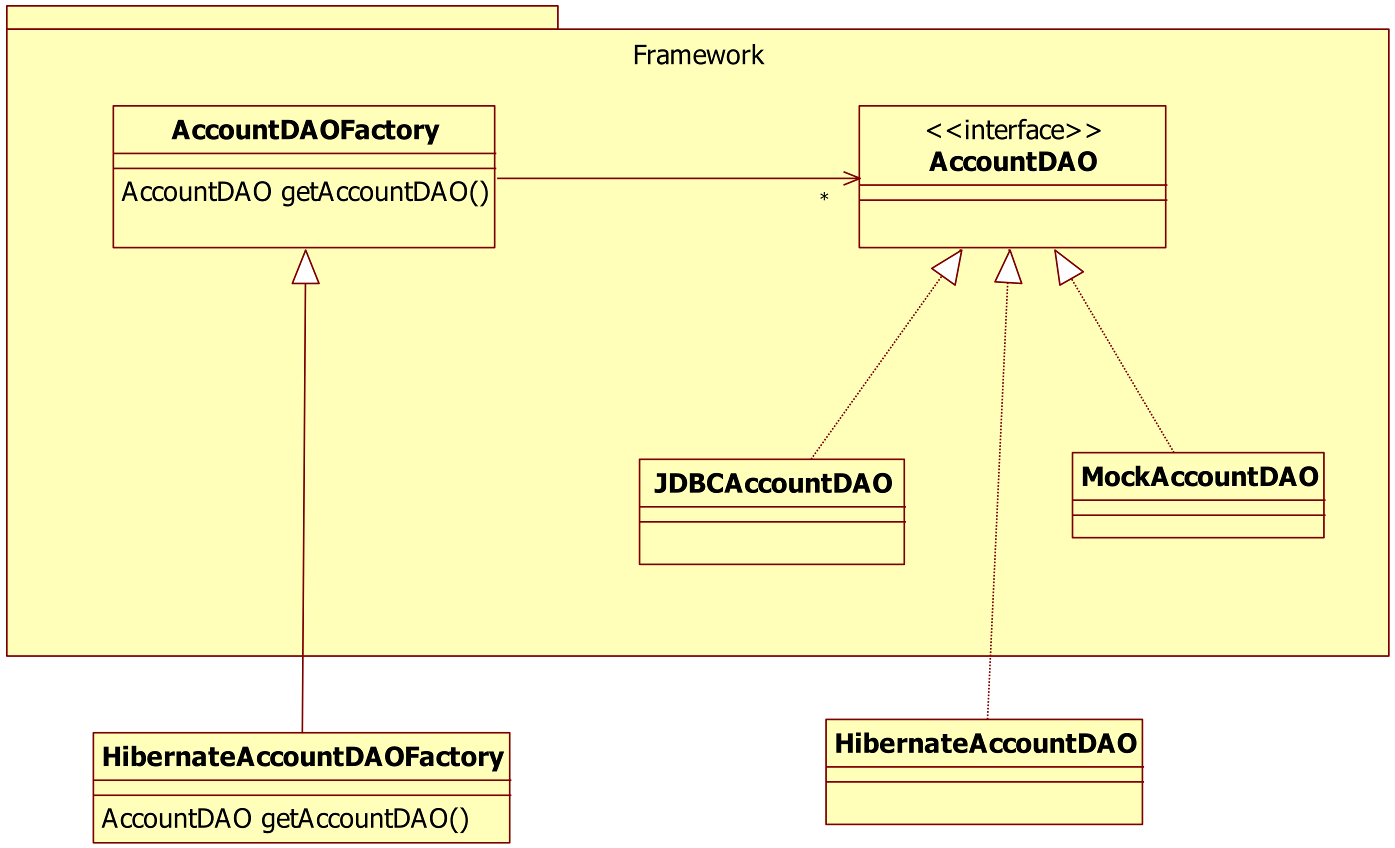
Plugin points: State pattern



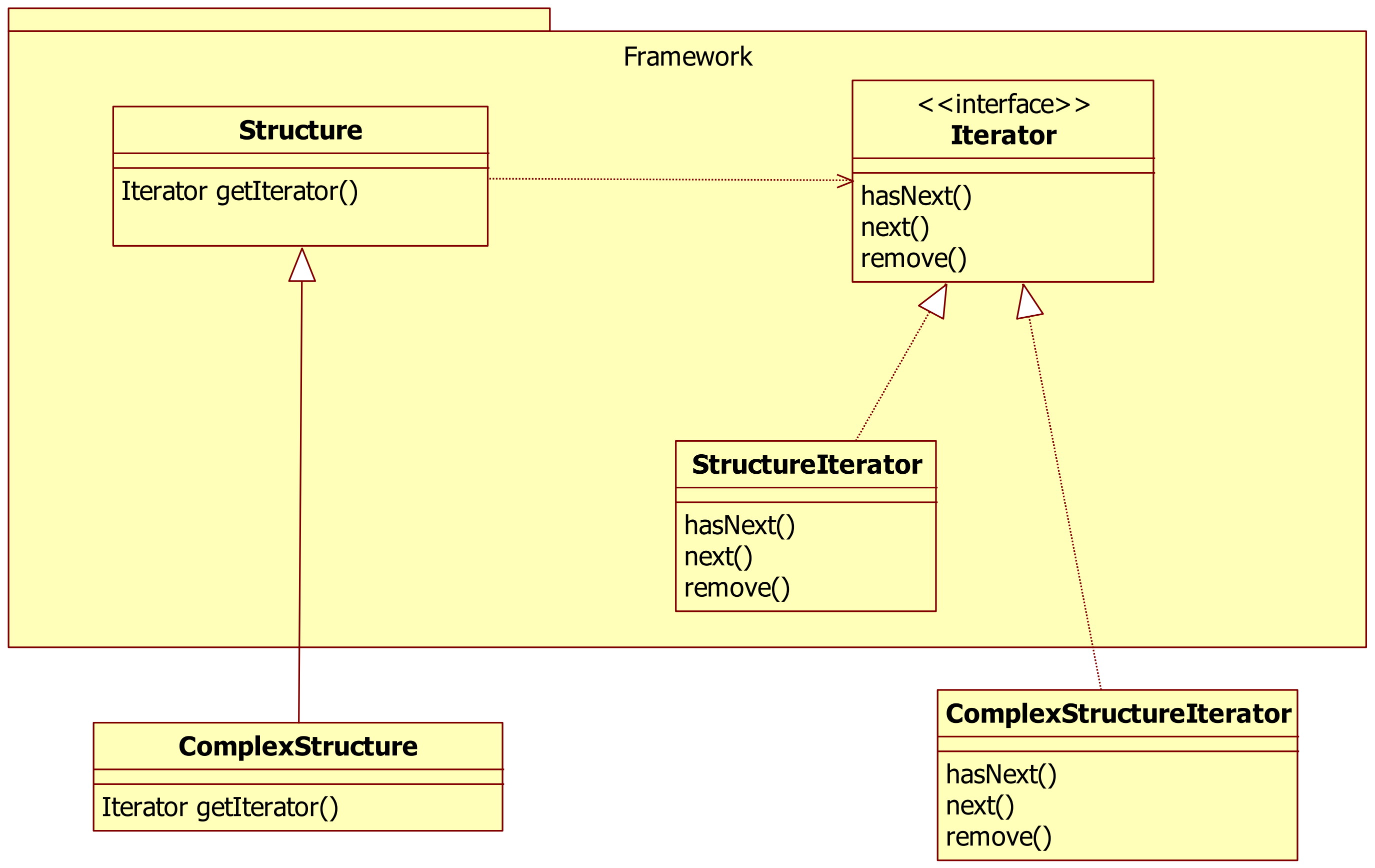
Plugin points: Observer pattern



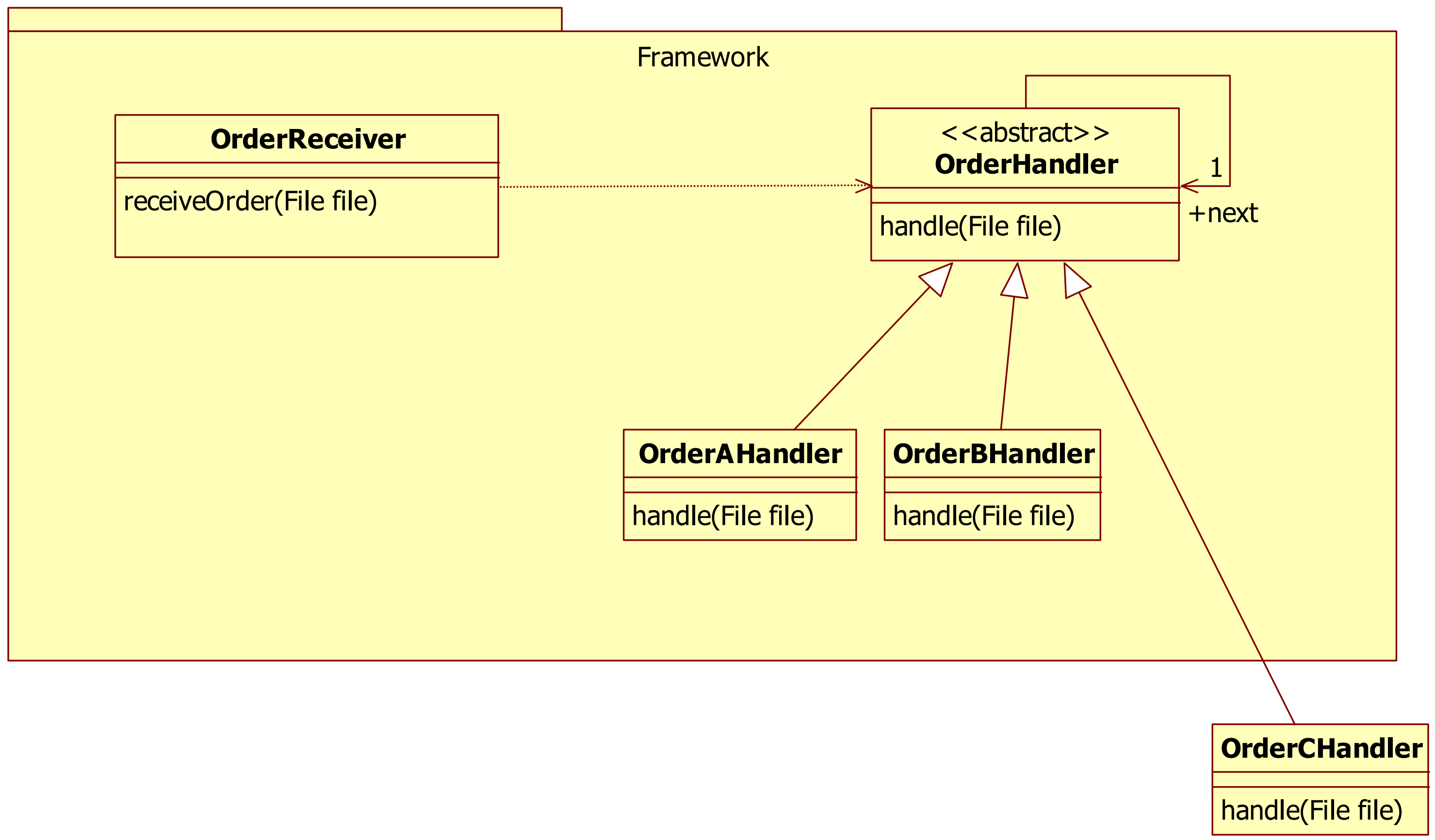
Plugin points: Factory pattern



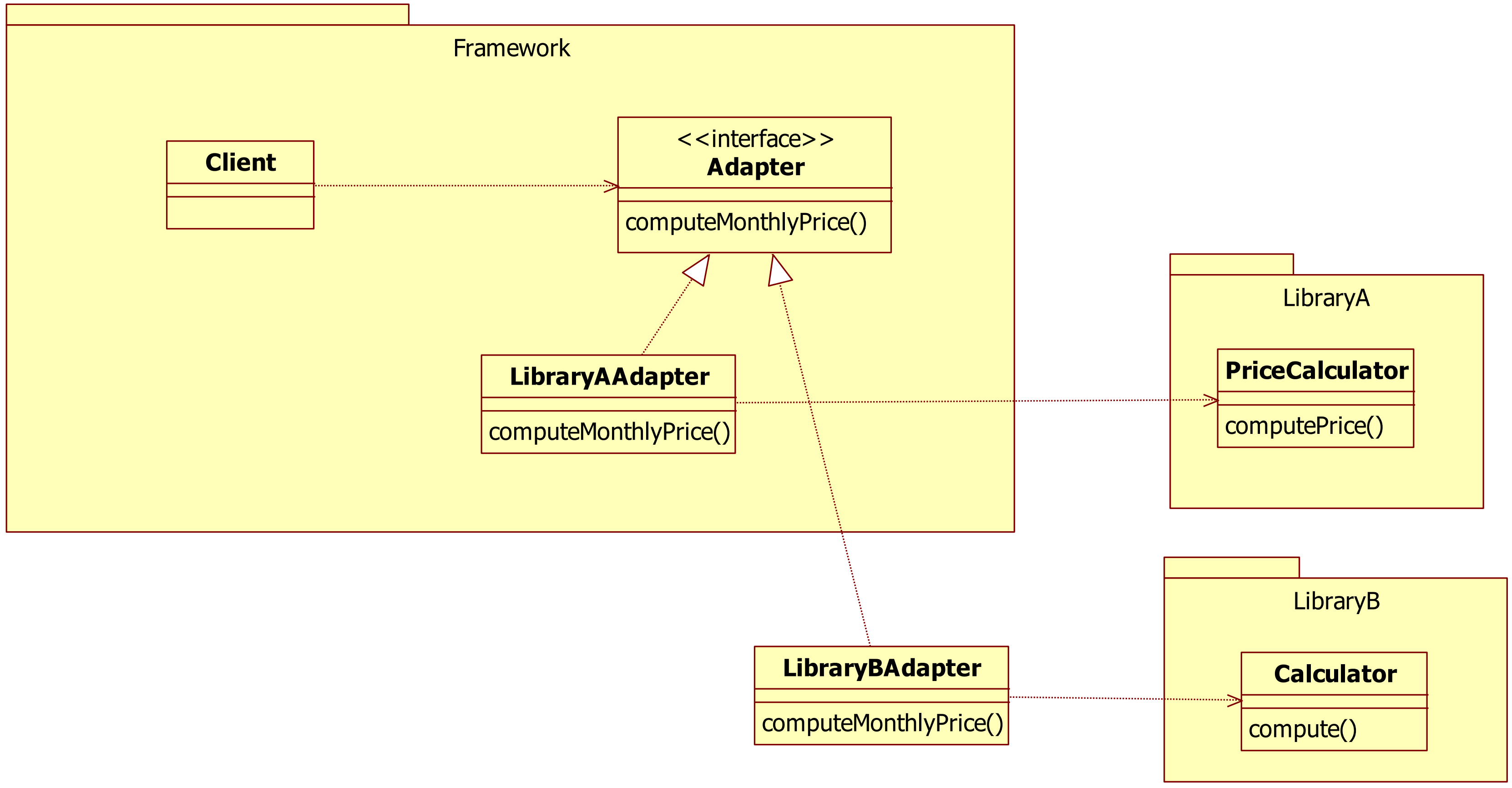
Plugin points: Iterator pattern



Plugin points: COR pattern

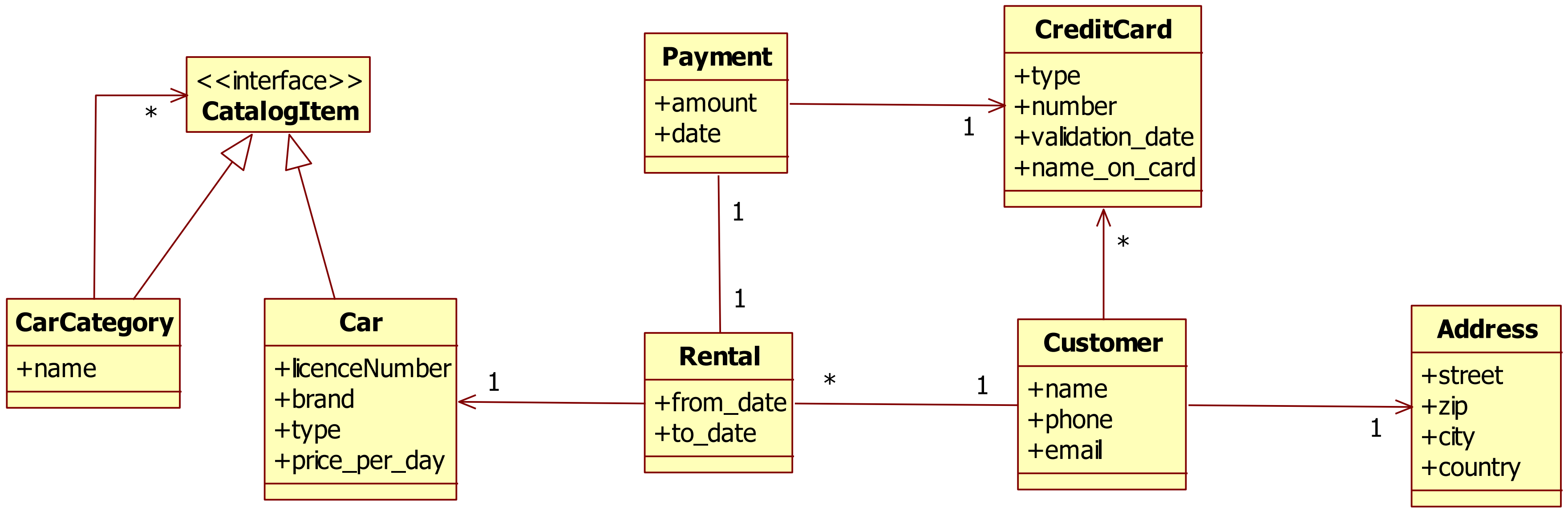


Plugin points: Adapter pattern

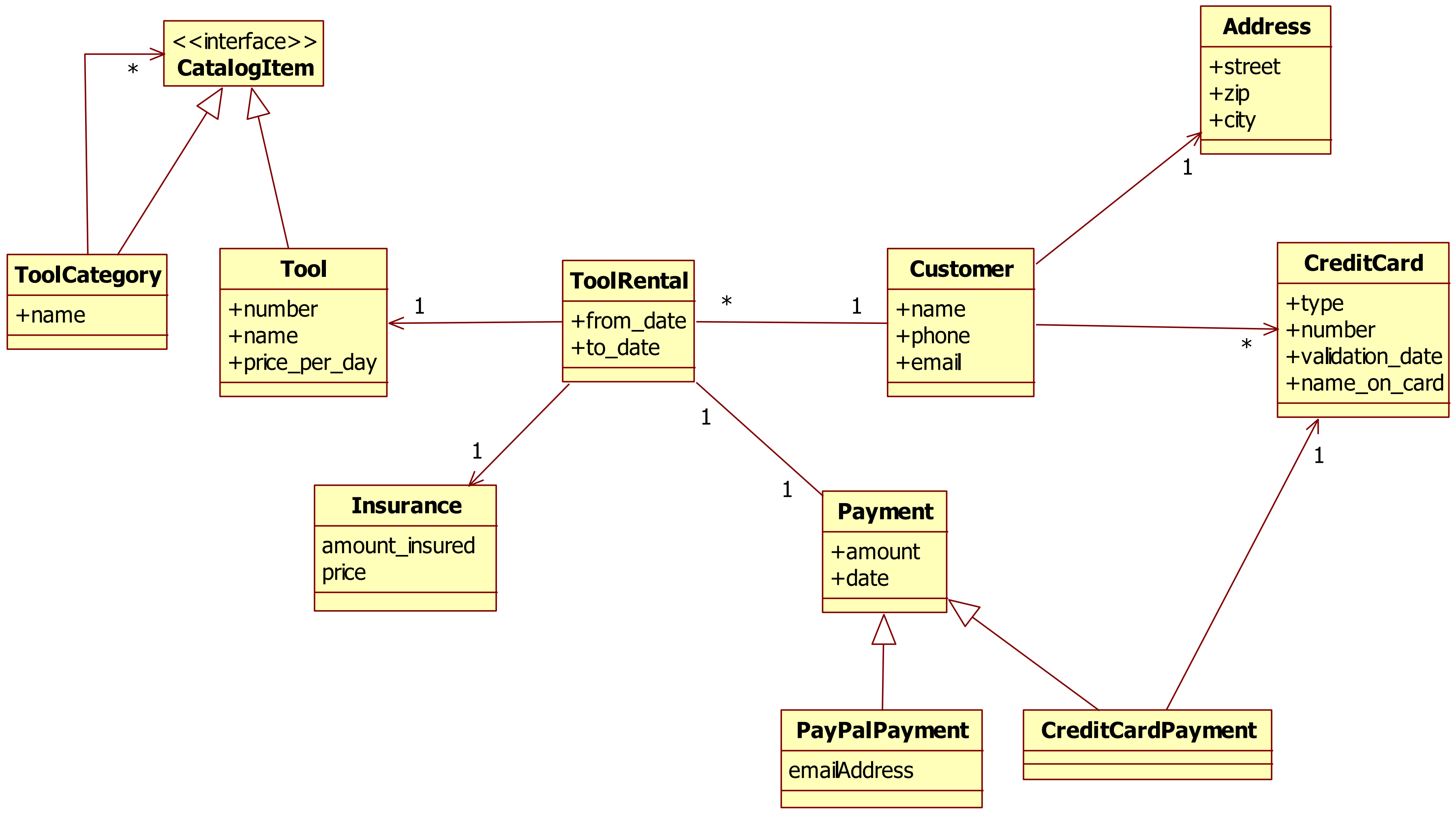


RENTAL FRAMEWORK

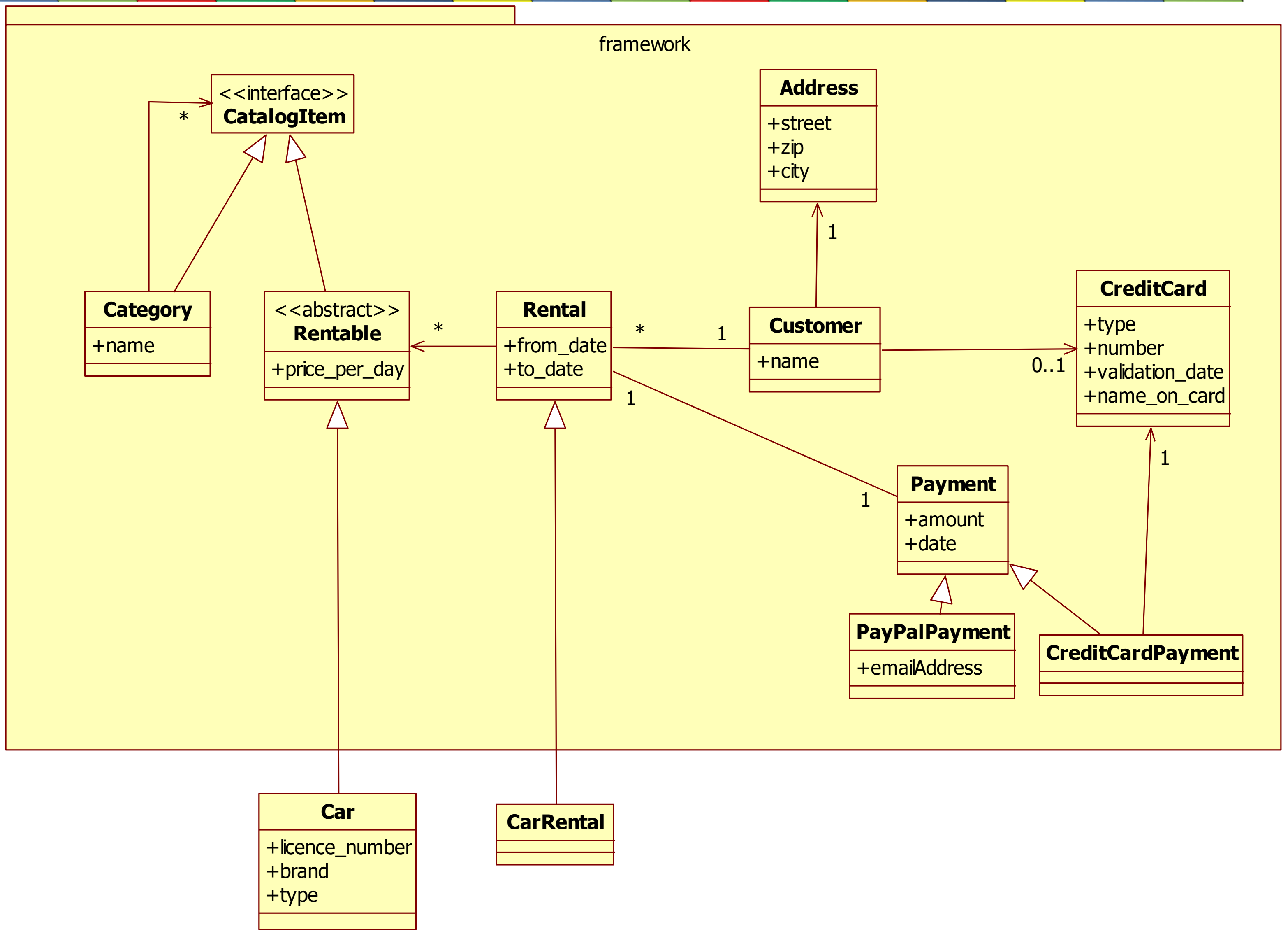
Car rental application



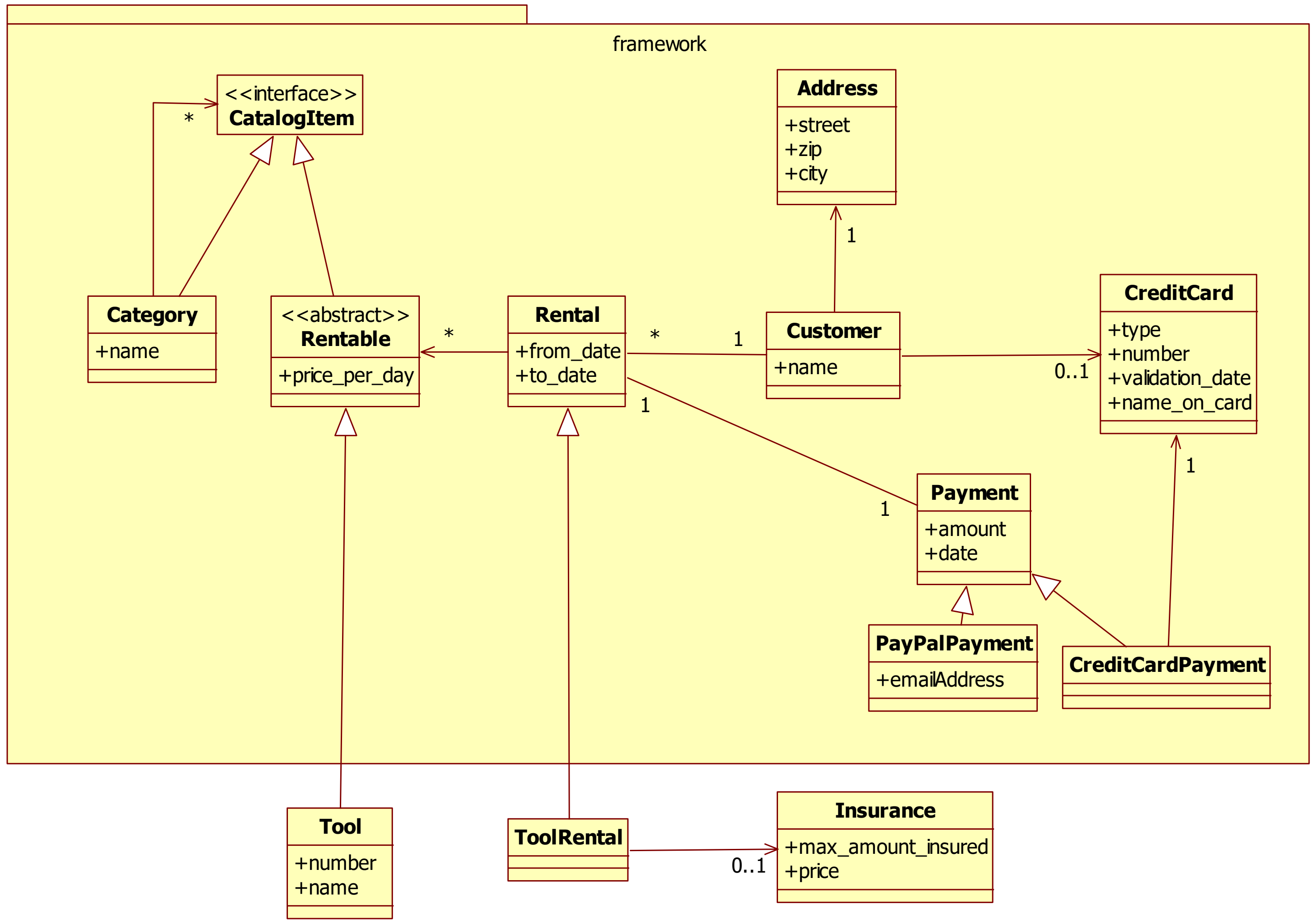
Tool rental application



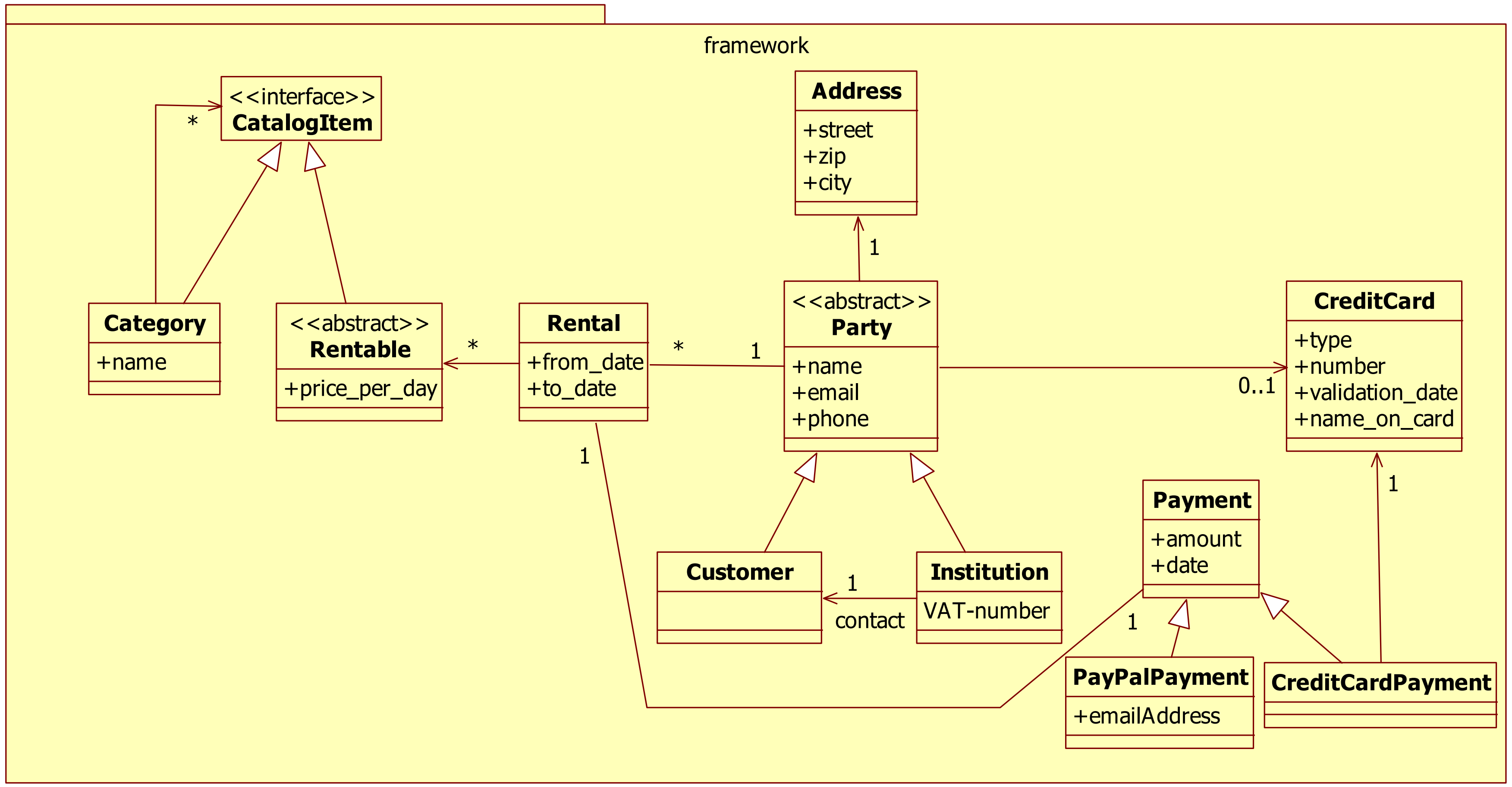
Framework + Car rental application



Framework + Tool rental application

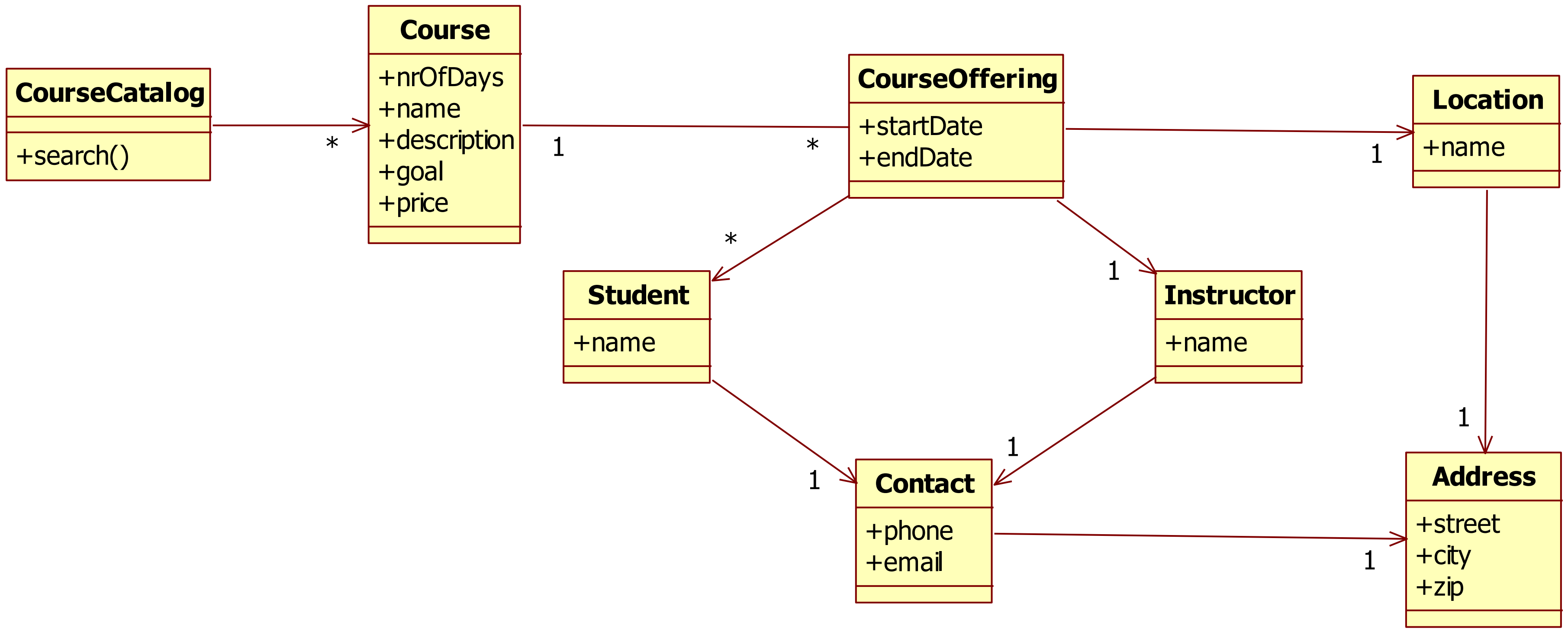


Party pattern

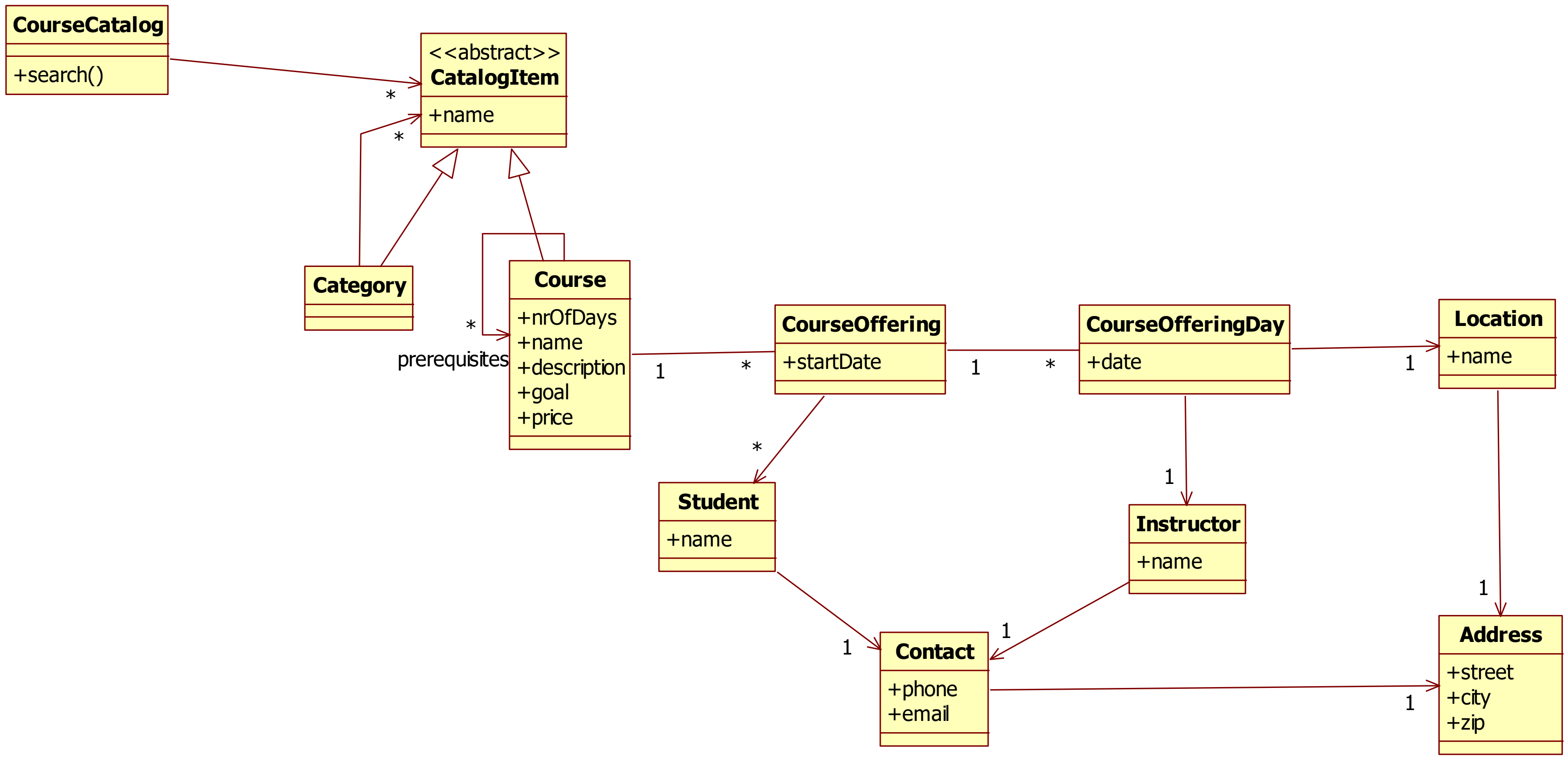


COURSE REGISTRATION FRAMEWORK

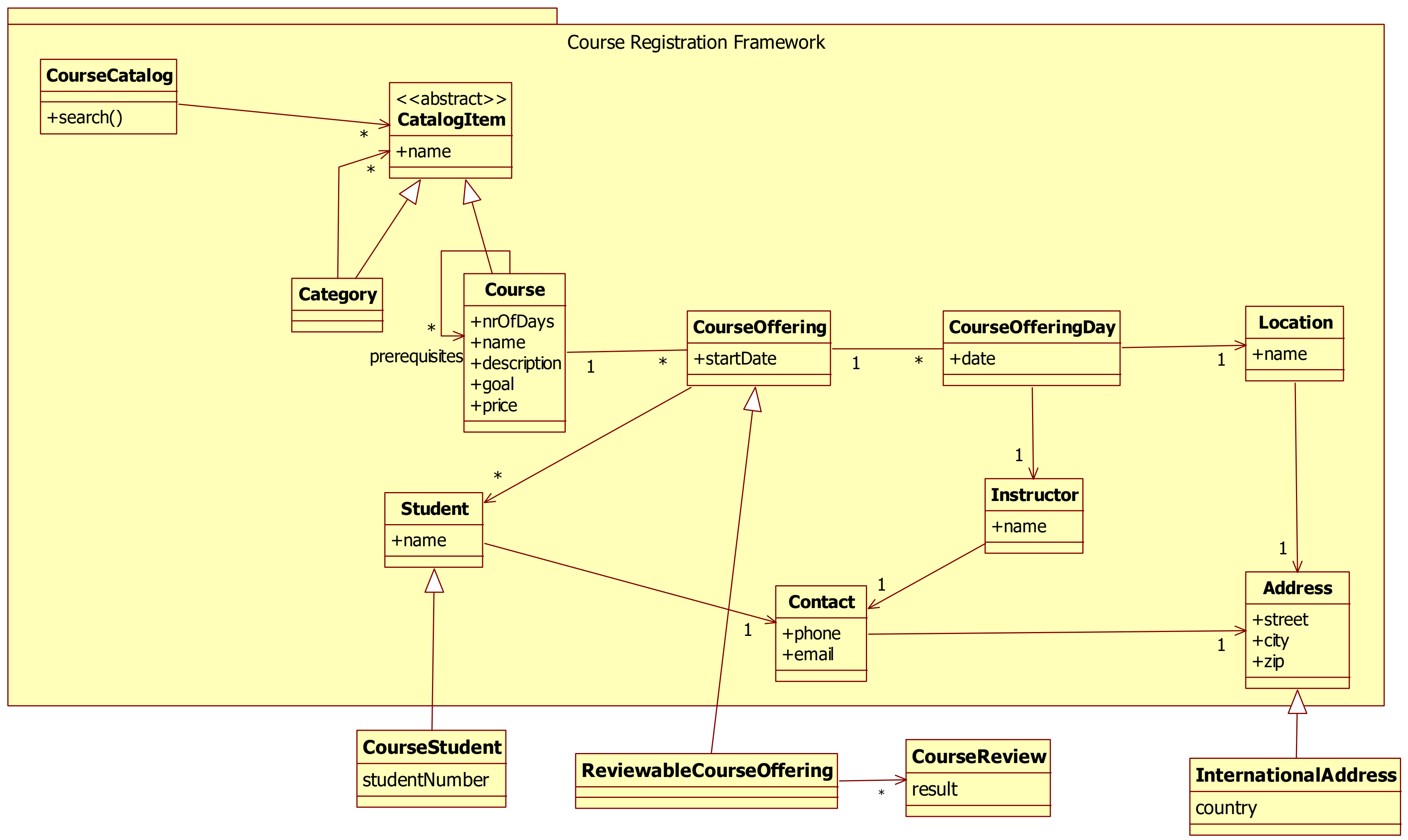
Simple course registration system



Advanced course registration system

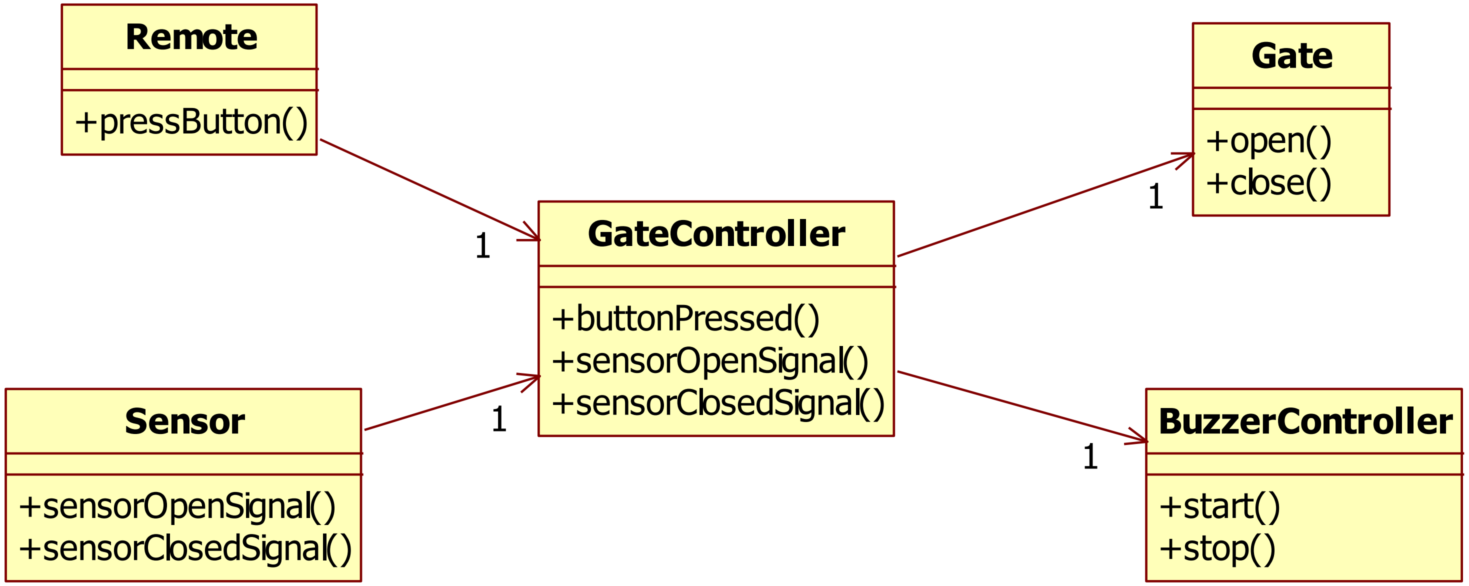


Course registration framework

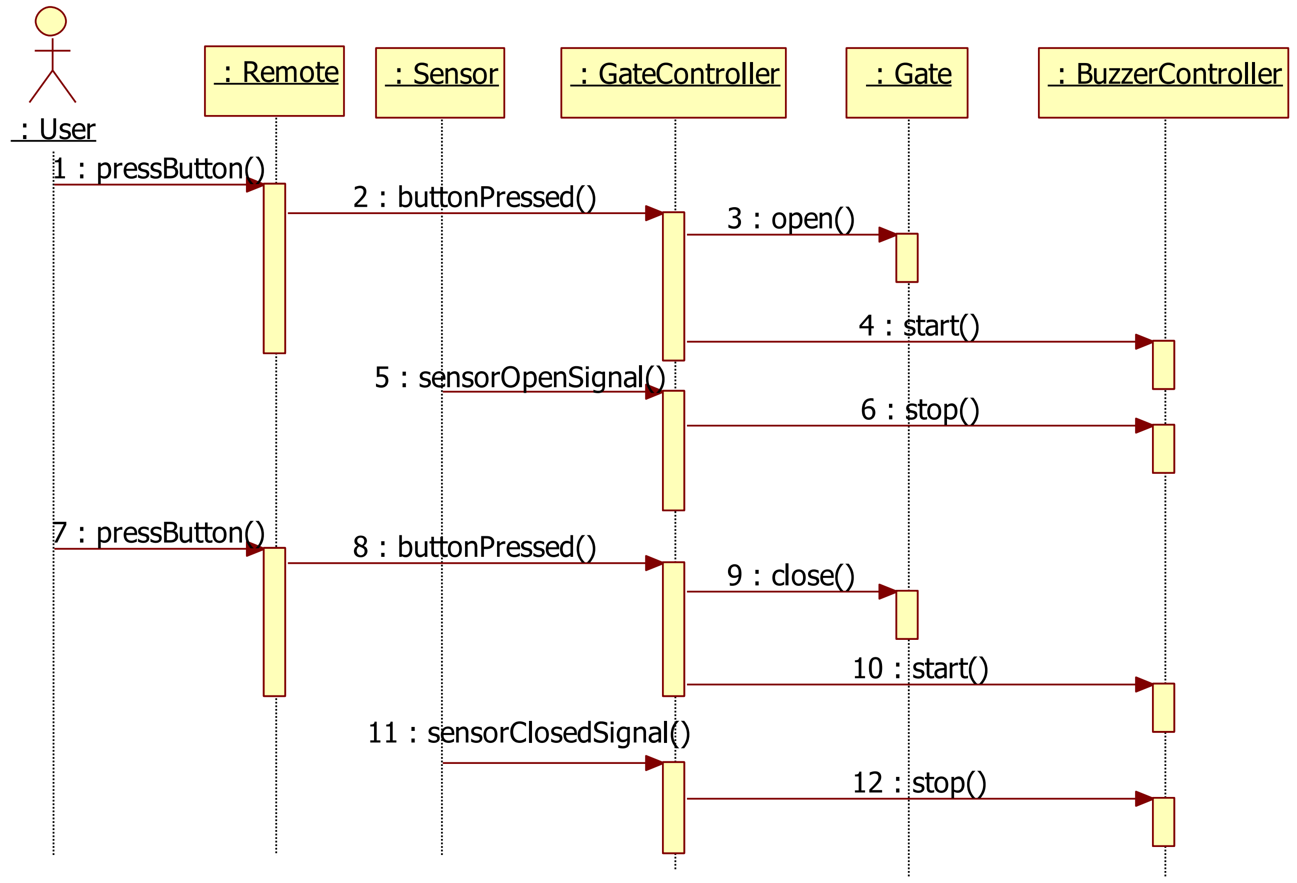


GATE CONTROLLER FRAMEWORK

Gate controller application



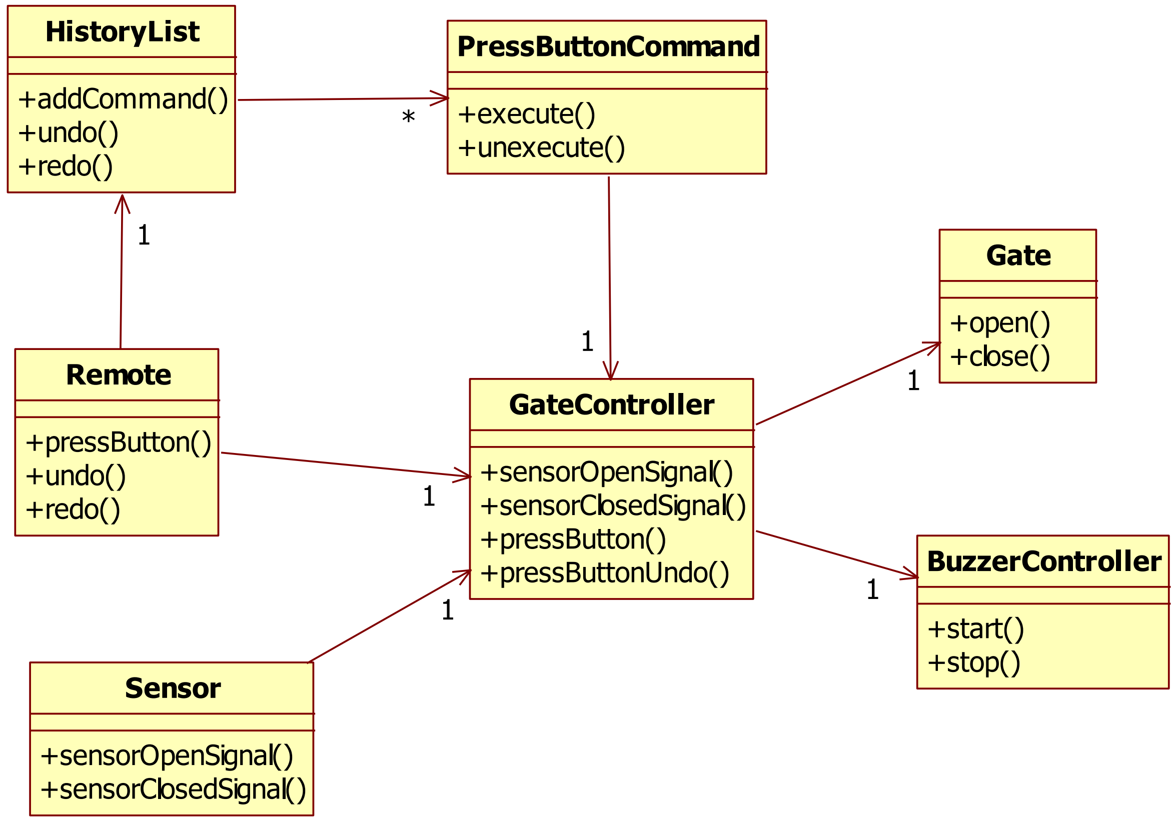
Gate controller application



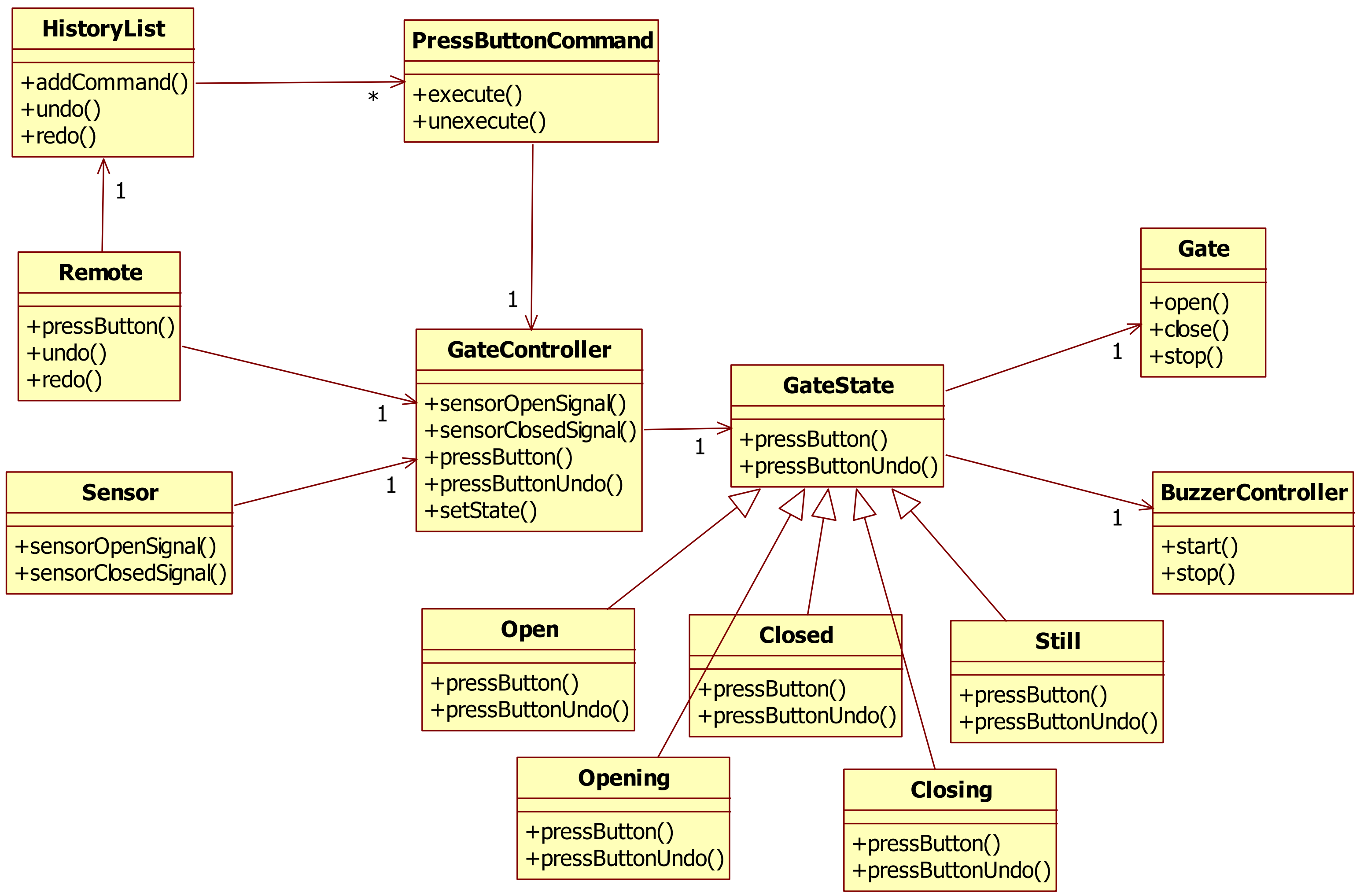
GateController framework

- Add undo/redo button
- Support different gate states (still, 75% open)
- Support multiple signaling devices (buzzers, lights, etc.)
- Support different gates

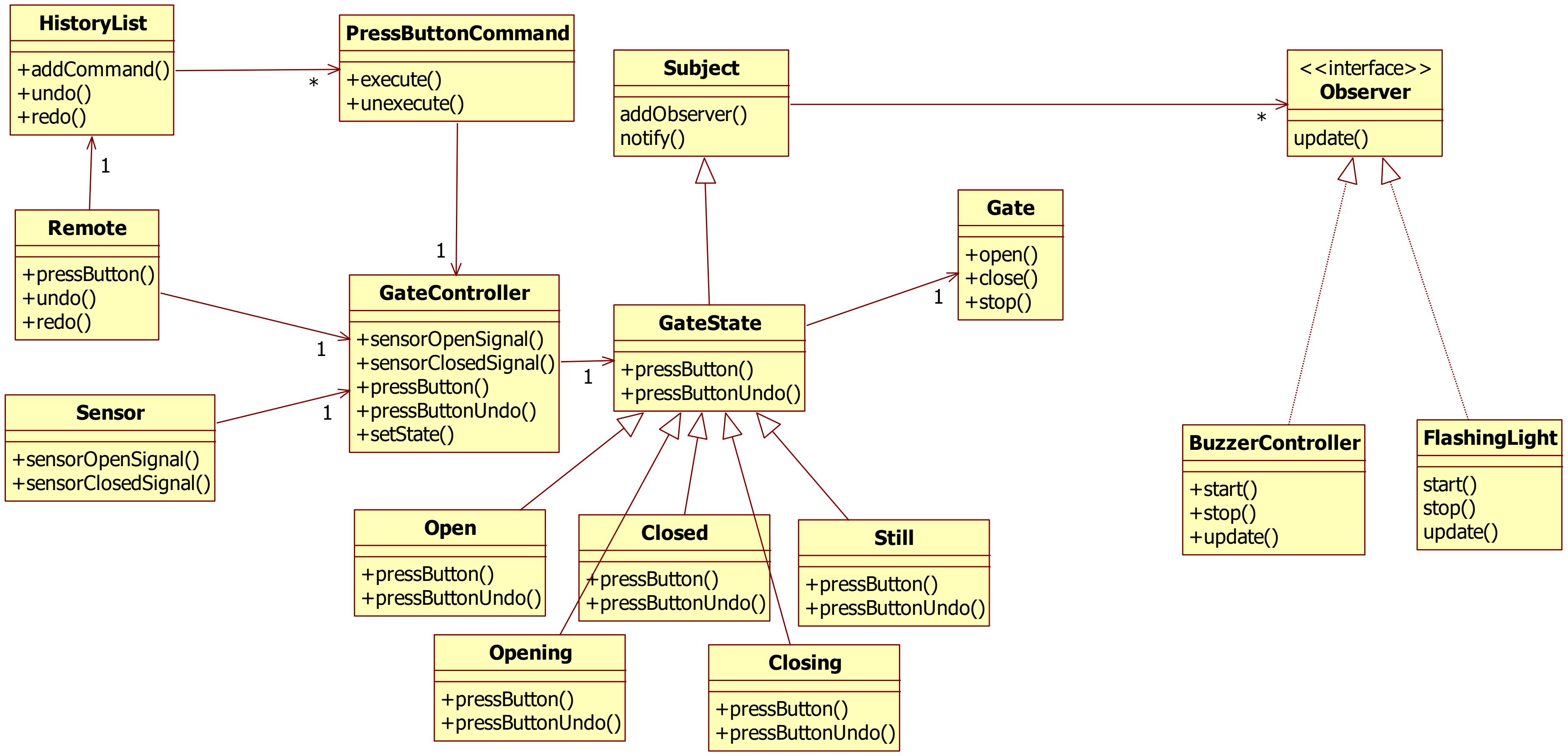
Support undo/redo button



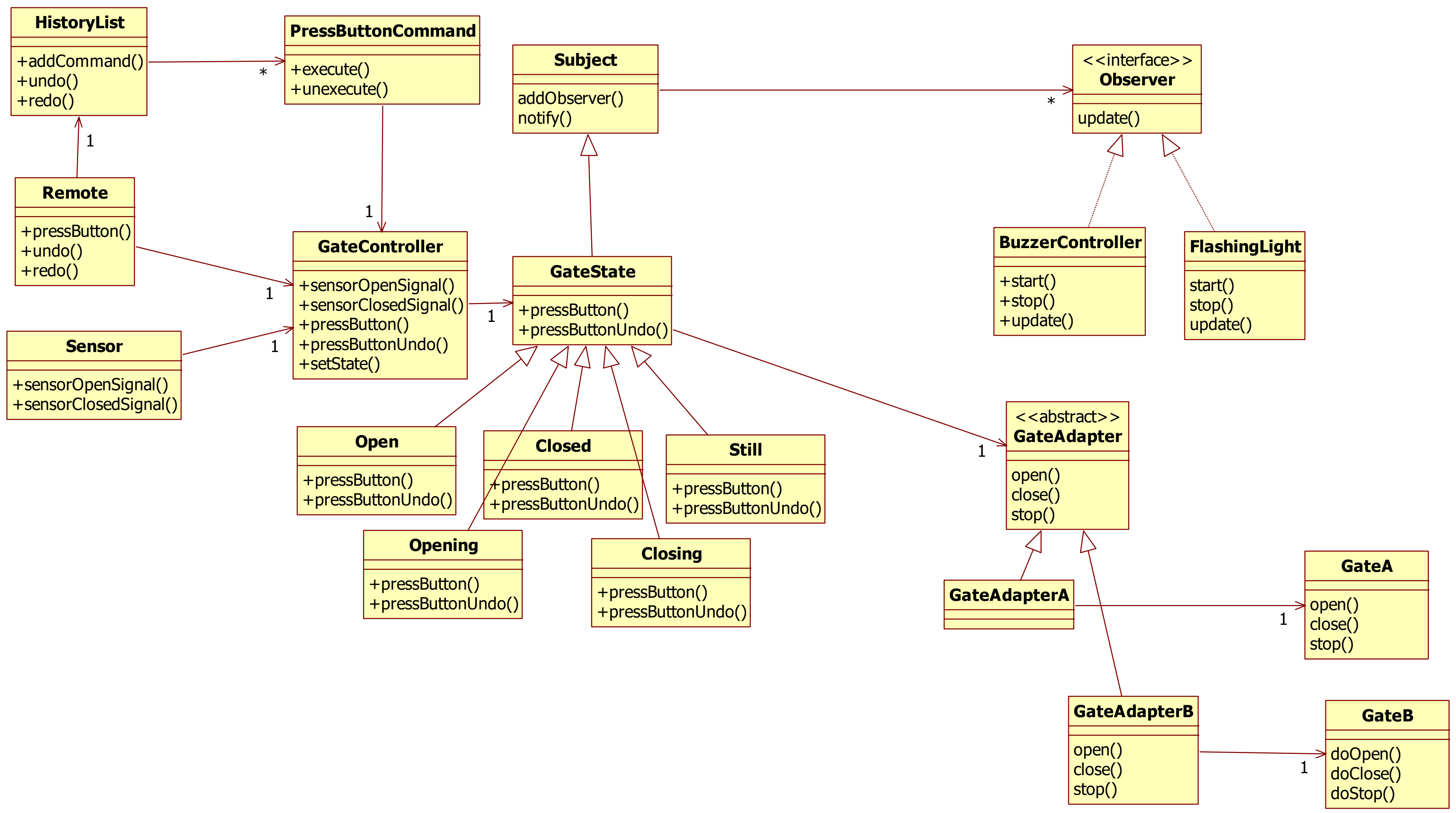
Support different gate states (still, 75% open, half open)



Support multiple signaling devices (buzzers, lights, etc.)



Support different gates



Main point

- A Framework captures domain specific expertise in abstract and concrete classes.

- The Unified Field which is the home of all the laws of nature, captures the intelligence of the whole universe.

Connecting the parts of knowledge with the wholeness of knowledge

1. Frameworks embody expertise: this frees developers who are not necessarily experts in a certain area from the complexity of the underlying details.
2. Frameworks are based on patterns. These patterns create the plugin points for the framework.

---

3. **Transcendental consciousness** is the home of all the Laws of Nature which govern the entire universe.
4. **Wholeness moving within itself:** In unity consciousness one spontaneously perceives the eternally silent, fully awake field of Pure Consciousness in the midst of all diversity.