LOYOLA
UNIVERSITY
CHICAGO

COMP 413: Intermediate Object-Oriented Programming

Dr. Robert Yacobellis
Advanced Lecturer
Department of Computer Science

10-Minute Quiz 2

- **Unit Testing (SE Radio Episode 167)**
- **Refactoring (SE Radio Episode 46)**

- **1 page of your own notes allowed**

- **Remember: Test 2 at the start of class November 8, not next week (November 1) – next week's class has been cancelled (go to Science Days presentation)!**

Week 9 Topics

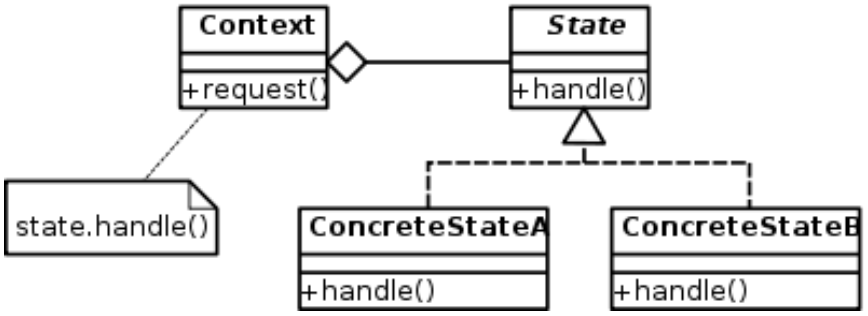
- **More Design Patterns (all covered in Week 8)**
  - **Adapter, Façade, Observer, State, Command – questions?**
- Model-View-Adapter Architectural Design Pattern
- The Stopwatch Example Program
  - UML State Diagrams (used with the State pattern)
  - State Diagram examples, including stopwatch
  - Project 4 overview
  - stopwatch-android-java
- Android Application Development
  - Android framework & activity life cycle
  - Android example apps – hello, simplebatch, clickcounter

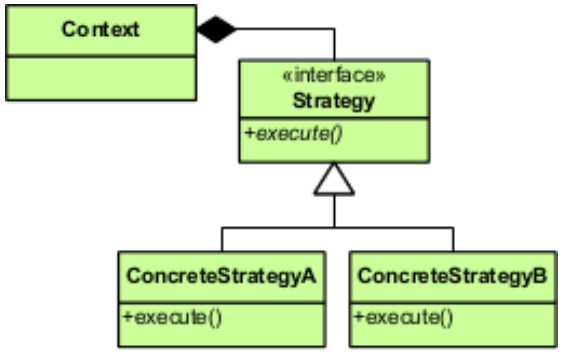
The State Pattern Review (Behavioral)

From Wikipedia, the free encyclopedia

- The **state pattern**, which closely resembles Strategy Pattern, is a behavioral software design pattern, also known as the **objects for states pattern**. This pattern is used in computer programming to encapsulate varying behavior for the same routine based on an object's state object. This can be a cleaner way for an object to change its behavior at runtime without resorting to large monolithic conditional statements.
- The **Context** in the State Pattern holds <u>different</u> **state objects** over time.

<u>State Pattern</u> **<u>Strategy Pattern</u>**





DPiJ Operations Patterns (Ch. 21)

- DPiJ says that operations patterns like State and Strategy "implement an operation in methods across several classes", and goes on to say that (based on UML concepts) …
  - An **operation** is a specification of a service that can be requested from an instance of a class, eg, **void toString()**
  - A **method** is an implementation of an operation
- An operation is thus a level of abstraction up from the concept of a method
- The concept of an *operation* relates to the idea that methods in different classes can have the same interface but implement that interface in different ways
  - Operations patterns address contexts where you need more than one method with the same interface as part of a design

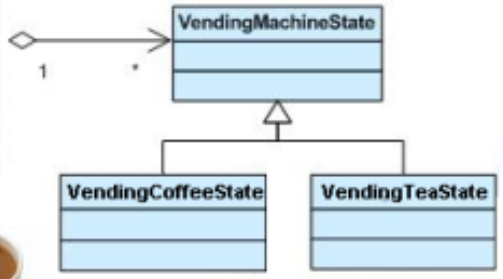
The State Pattern (DPiJ Ch. 22)

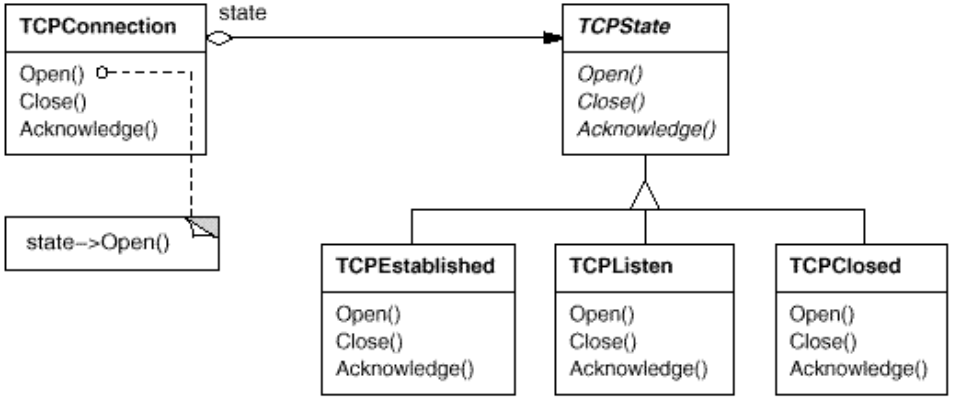
- If you intend to distribute an operation so that each class represents a different state, use the State Pattern
  - If an object has a state that you want to model, you may find that the state is tracked via a variable which appears in complex, cascading *if* statements; the State Pattern offers a simpler approach
  - In this pattern, states are modeled as objects from different classes
  - Communication & dependencies can be managed in different ways

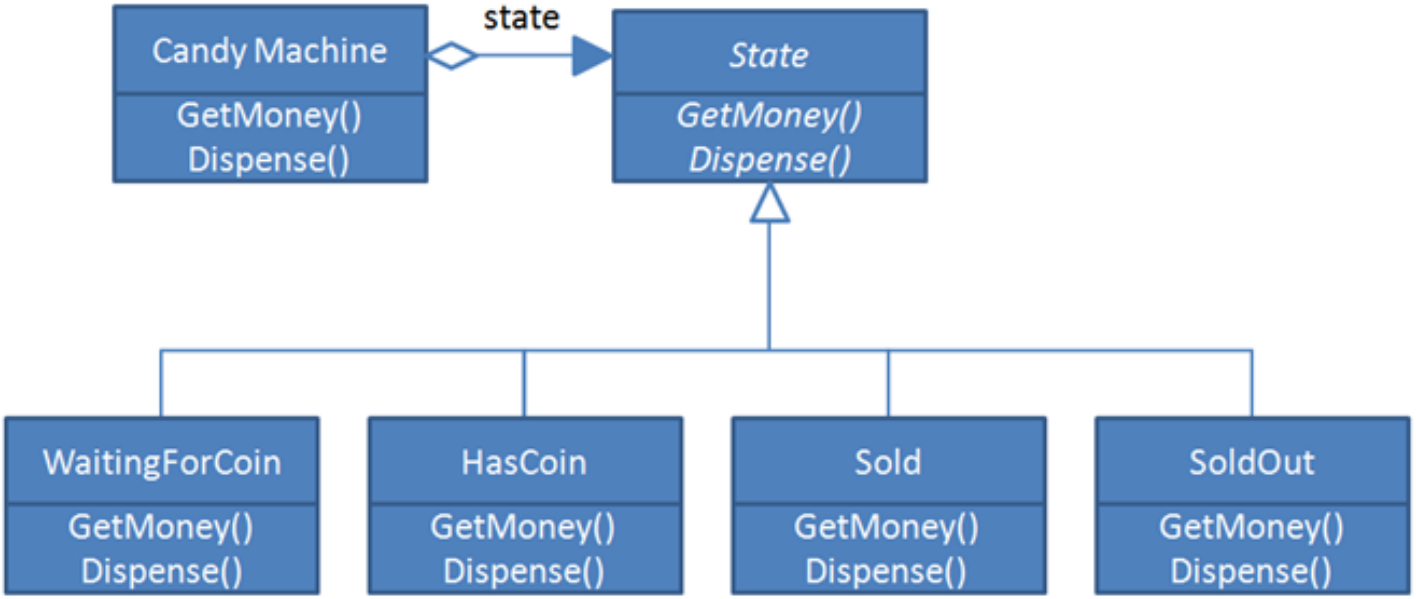
http://www.tutorialspoint.com/design_pattern/state_pattern.htm

→ **Bob Tarr State slides 2-4, 13-20**

State Design Pattern Examples





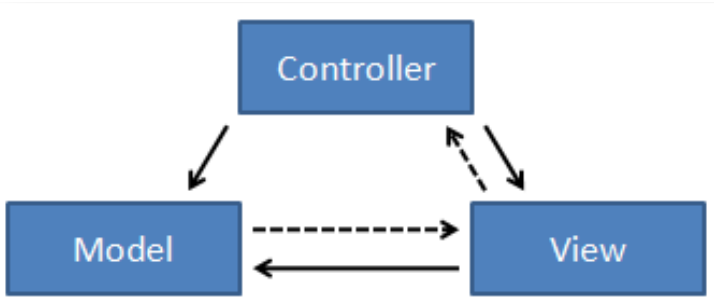


Week 9 Topics

- More Design Patterns
  - Adapter, Façade, Observer, State, Command
- **Model-View-Adapter Architectural Design Pattern**
- The Stopwatch Example Program
  - UML State Diagrams (used with the State pattern)
  - State Diagram examples, including stopwatch
  - Project 4 overview
  - stopwatch-android-java
- Android Application Development
  - Android framework & activity life cycle
  - Android example apps – hello, simplebatch, clickcounter

Model-View-Controller vs. Model-View-Adapter

- **Model–View–Controller (MVC)** is an architectural pattern used in software engineering. Successful use of the pattern isolates business logic from user interface considerations, resulting in an application where it is easier to modify either the visual appearance of the application or the underlying business rules without affecting the other. In MVC, the **model** represents the information (the data) of the application; the **view** corresponds to elements of the user interface such as text, checkbox items, and so forth; and the **controller** manages the communication of data and the business rules used to manipulate the data to and from the model. **Source: Wikipedia.**

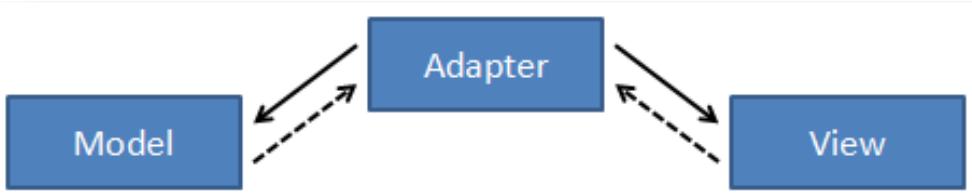


Solid lines are direct relationships, dashed lines are indirect ones. In MVC the Model and View can communicate.

http://www.tutor :ern/mvc_pattern.htm

Model-View-Controller vs. Model-View-Adapter

- **Model–View–Adapter (MVA)** is a variant of MVC where an **adapter** object strictly mediates between the **model** and the **view**. The **adapter** holds a reference to both the **model** and the **view** and directly calls methods on both. It also attaches itself as a ***listener*** to both the **model** and the **view** in order to receive events. It receives property change events from the **model** and action events (checkbox ticked, text entered, etc.) from the **view**, and then routes appropriate changes to the other side. The **adapter** is entirely responsible for keeping the **model** and the **view** in sync; the **model** and **view** are both relatively dumb structures, each knowing nothing about the other.



https://www.palantir.com/2009/04/model-view-adapter/

Model-View-Adapter Advantages

- All “moving parts” are centralized in one place, the Adapter. No worrying about where to add a listener; no hunting around to find isolated listeners.
- Separation of concerns between the View and the Adapter. The View is responsible for layout and visual presentation while the Adapter is responsible for synchronization and the dynamic aspects of the user interface.
- Better decoupling between Models and Views. Specifically, the View doesn’t need to know anything about the Model.

- This architectural pattern is used in the stopwatch program, reviewed below

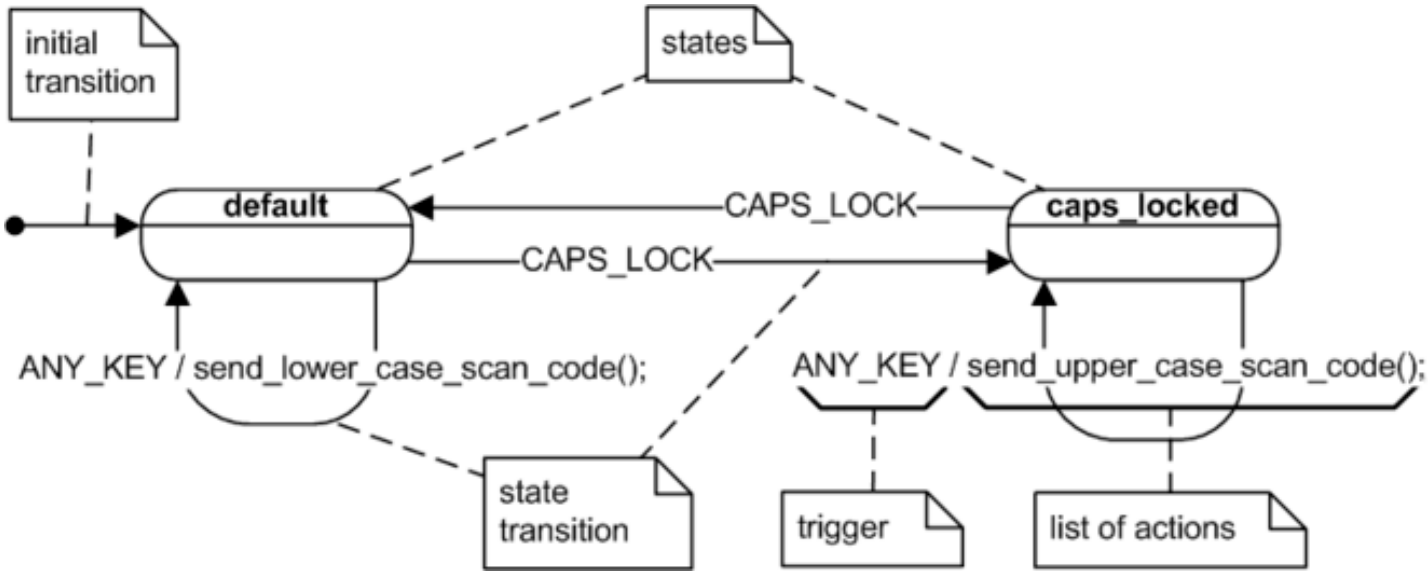
Week 9 Topics

- More Design Patterns
  - Adapter, Façade, Observer, State, Command
- Model-View-Adapter Architectural Design Pattern
- **The Stopwatch Example Program**
  - **UML State Diagrams (used with the State pattern)**
  - **State Diagram examples, including stopwatch**
  - **Project 4 overview**
  - **stopwatch-android-java**
- Android Application Development
  - Android framework & activity life cycle
  - Android example apps – hello, simplebatch, clickcounter

State Diagrams and UML State Machines

- A **state diagram** is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics. **Source: Wikipedia**.
- A **UML state machine** is an object-based variant of a Harel statechart, adapted and extended by UML. The goal of UML state machines is to overcome the main limitations of traditional finite-state machines while retaining their main benefits. The term "UML state machine" can refer to two kinds of state machines: behavioral state machines and protocol state machines. Behavioral state machines can be used to model the behavior of individual entities (e.g., class instances). Protocol state machines are used to express usage protocols and can be used to specify the legal usage scenarios of classifiers, interfaces, and ports. **Source: Wikipedia**.

Basic UML State Diagrams



UML state diagrams are directed graphs in which nodes denote states and connectors denote state transitions. For example, Figure 1 shows a UML state diagram corresponding to a computer keyboard state machine. In UML, states are represented as rounded rectangles labeled with state names. The transitions, represented as arrows, are labeled with the triggering events followed optionally by the list of executed actions. The **initial transition** originates from the solid circle and specifies the default state when the system first begins. Every state diagram should have such a transition, which should not be labeled, since it is not triggered by an event. The initial transition can have associated actions.

Other Example UML State Diagrams

- examples
- stopwatch model (hardware perspective)
- our stopwatch model

- You'll use the stopwatch Android implementation as one starting point for **Project 4** – see next two slides

Project 4 – Implement a Simple Timer in Android – Due Tuesday, 12/6

Functional Requirements (55%)

- The timer has the following controls:
  - (0.5) One two-digit display of the form 88 (two printed characters).
  - (0.5) One multi-function button.
- The timer behaves as follows (part 1 of 2):
  - (0.5) The timer always displays the remaining time in seconds.
  - (0.5) Initially, the timer is stopped and the (remaining) time is zero.
  - (0.5) If the button is pressed when the timer is stopped, the time is incremented by one up to a preset maximum of 99. (The button acts as an increment button.)
  - (0.5) If the time is greater than zero and three seconds elapse from the most recent time the button was pressed, then the timer beeps once and starts running (ie, counting down).

Project 4 – Implement a Simple Timer in Android – Due Tuesday, 12/6

Functional Requirements (55%)

- The timer behaves as follows (part 2 of 2):
  - (0.5) While running, the timer subtracts one from the time for every second that elapses.
  - (0.5) If the timer is running and the button is pressed, the timer stops and the time is reset to zero. (The button acts as a cancel button.)
  - (0.5) If the timer is running and the time reaches zero by itself (without the button being pressed), then the timer stops and the alarm starts beeping continually and indefinitely.
  - (0.5) If the alarm is sounding and the button is pressed, the alarm stops sounding; the timer is now stopped and the (remaining) time is zero. (The button acts as a stop or reset button.)
  - (0.5) The timer handles rotation by continuing in its current state.
- You'll develop a state diagram for this in a future class …

Working on Project 4

- You'll work on Project 4 in larger Groups (from the UML exercise):

Name	Group
Killham, Eric John	1
Tapia, Rene	1
Cicale, Julia	1
Rodriguez Orjuela, Jose Luis	1
Mir, Sarfaraz Ali Khan	1
Nowreen, Syeda Tashnuva	1
Goel, Neha	2
Soliz Rodriguez, Percy Gabriel	2
Misra, Anadi	2
Pacheco, Andrea	2
Mehta, Shipra Ashutosh	2
Sindhu, Pinky	2
Al Khofi, Sundas Abdullatif A	3
Liu, Siyuan	3
Meghrajani, Aman Maheshkumar	3
Oakey, Anthony William	3
Solomon-Phillips, Tyree James	3

- You'll do a futu ercise; to prepare for it, **install ArgoUM /, on your PC by 11/15**

stopwatch-android-java Objectives

- **Modeling**
  - Modeling state-dependent behavior with state machine diagrams
  - Distinguishing between view states and (behavioral) model states
- **Semantics**
  - Event-driven/asynchronous program execution
  - User-triggered input events
  - Internal events from background timers
  - Concurrency issues: single-thread rule of accessing/updating the view in the GUI thread

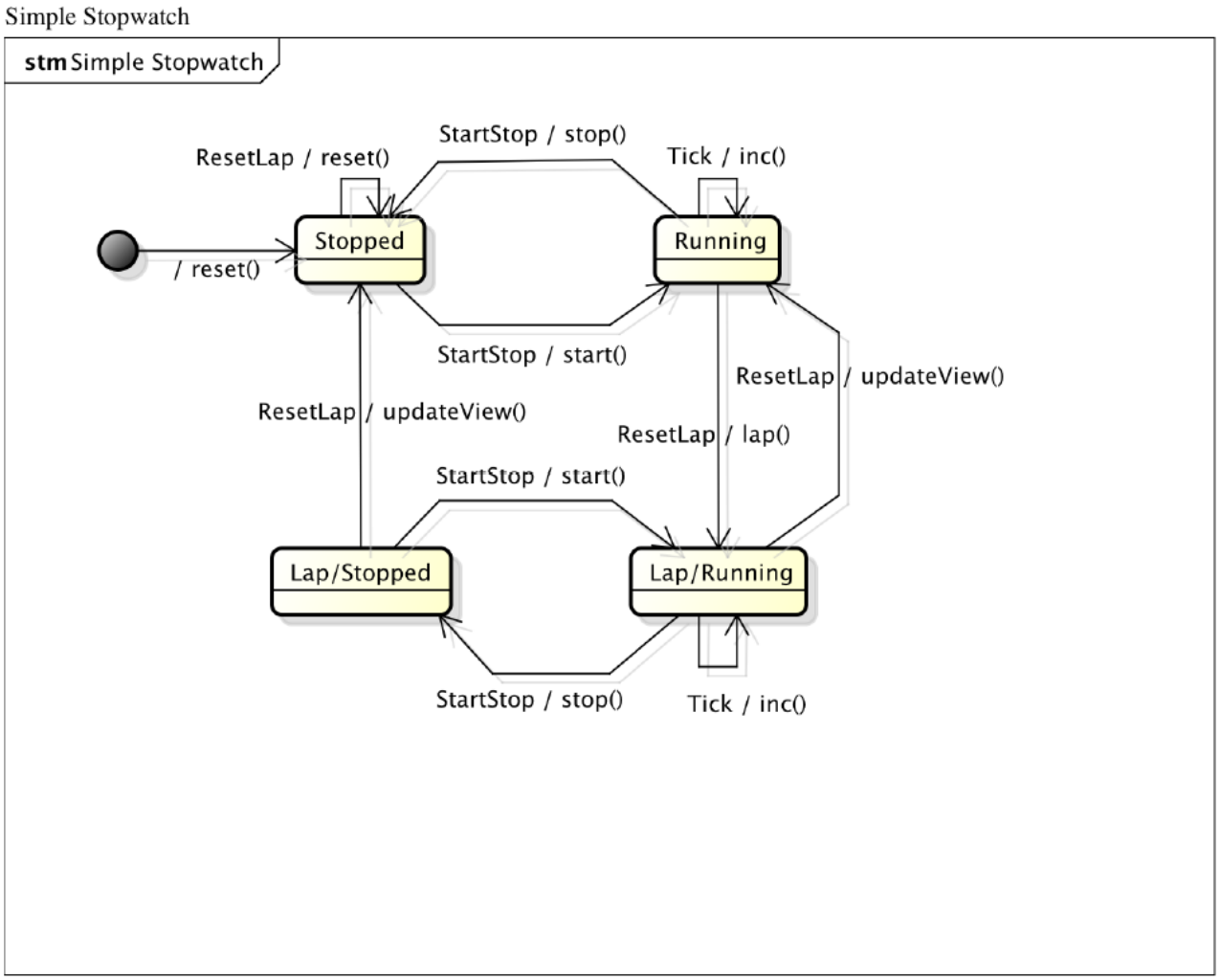
stopwatch-android-java Architecture, Design, and Testing – 1

- **Key architectural issues and patterns**
  - Simple dependency injection (DI)
  - Model-view-adapter (MVA) architectural pattern, differences between MVA and model-view-controller (MVC)
  - Mapping MVA to Android
  - Distinguishing among dumb, reactive, and autonomous model components
- **Key design patterns**
  - Implementing event-driven behavior using the Observer pattern
  - Implementing state-dependent behavior using the State pattern
  - Command pattern for representing tasks as objects
  - Façade pattern to hide complexity in the model from the adapter

stopwatch-android-java Architecture, Design, and Testing – 2

- **Relevant class-level design principles**
  - Dependency Inversion Principle (DIP)
  - Single Responsibility Principle (SRP)
  - Interface Segregation Principle (ISP)
- **Package-level architecture and relevant principles**
  - Dependency graph
  - Stable Dependencies Principle (SDP)
  - Acyclic Dependencies Principle (ADP)
- **Testing**
  - Different types of testing - component-level unit testing, system testing, instrumentation testing
  - Mock-based testing, testcase Superclass pattern, test coverage

stopwatch-android-java
State Machine Mapped to Java Code

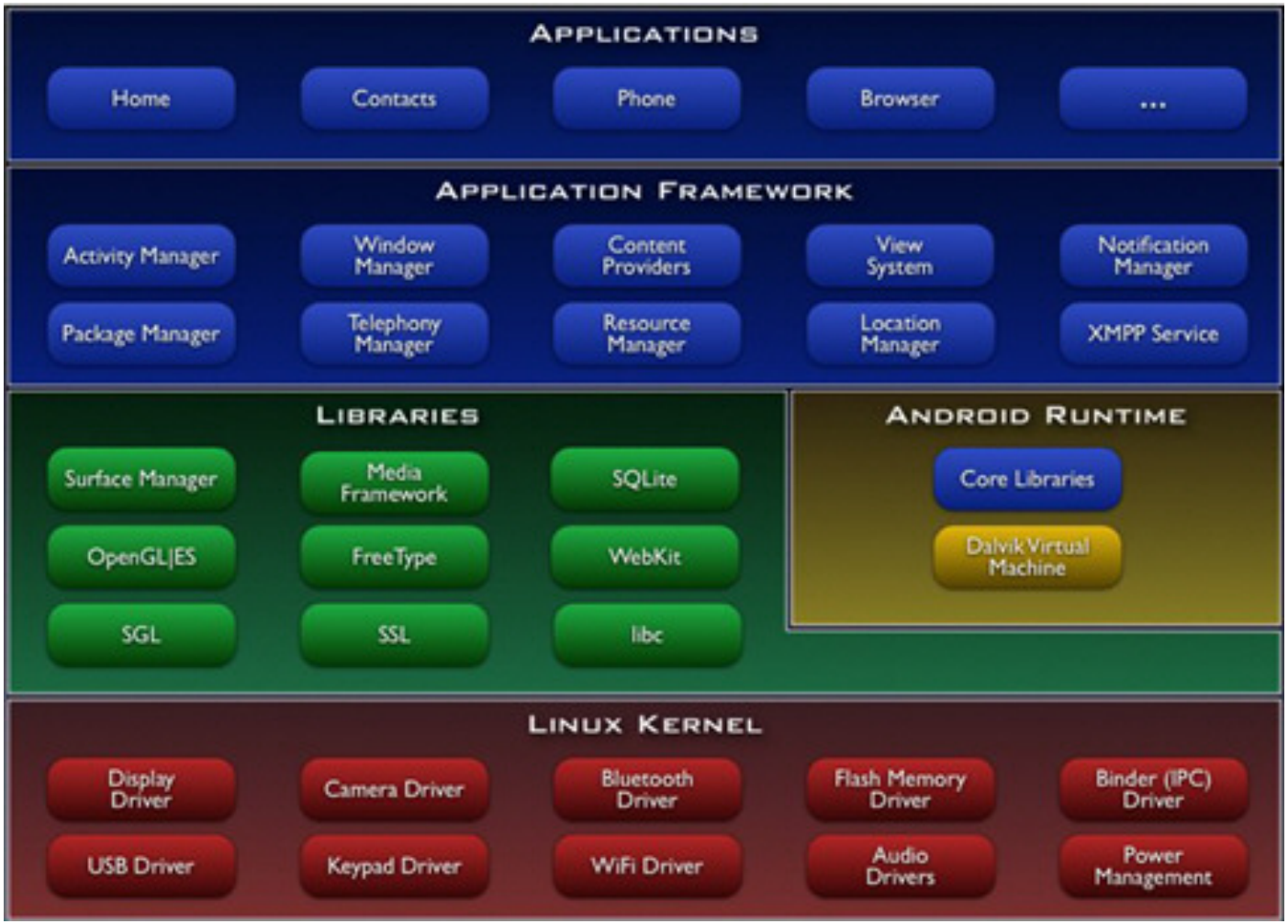


LOYOLA UNIVERSITY CHICAGO

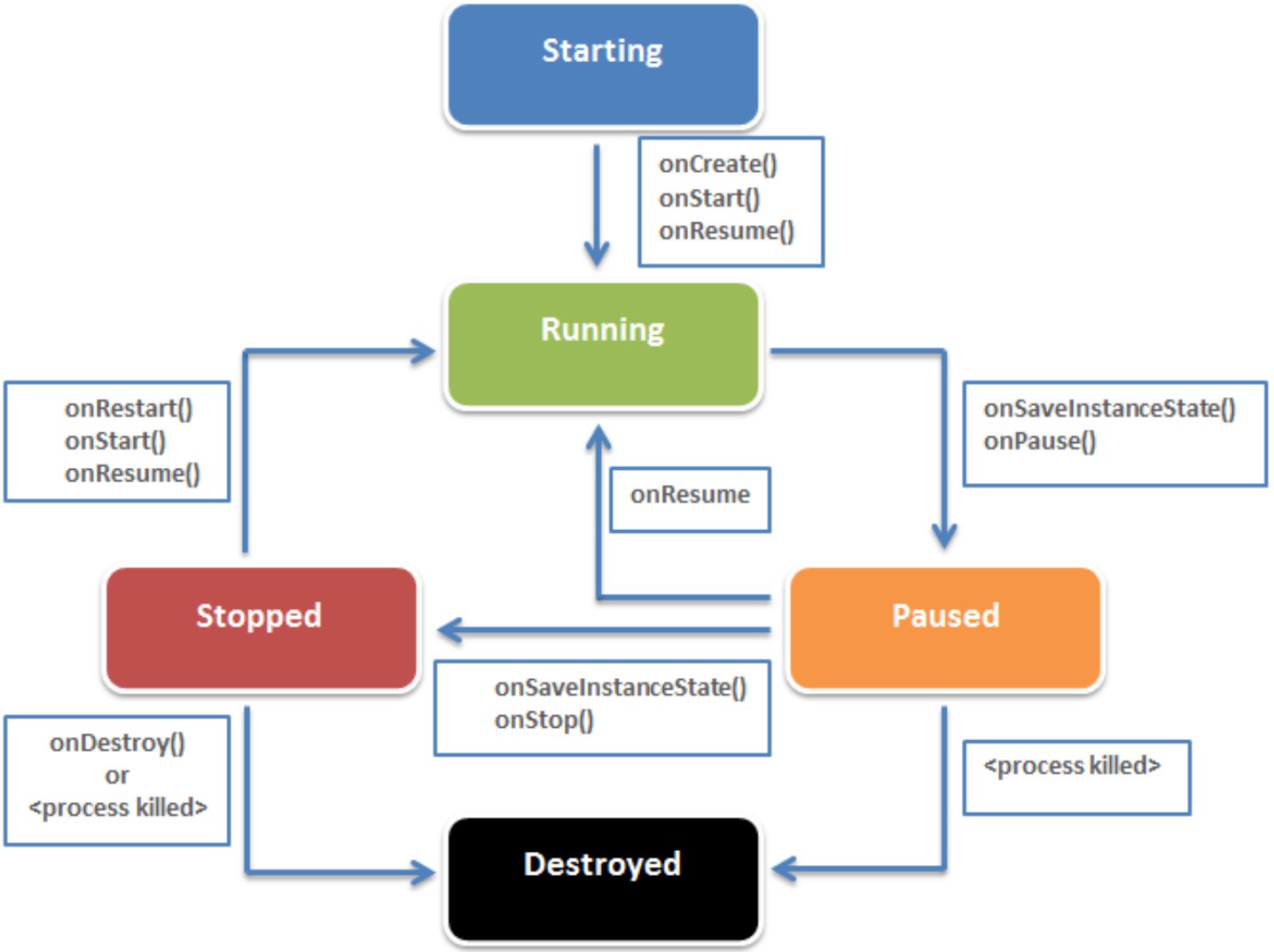
Week 9 Topics

- More Design Patterns
  - Adapter, Façade, Observer, State, Command
- Model-View-Adapter Architectural Design Pattern
- The Stopwatch Example Program
  - UML State Diagrams (used with the State pattern)
  - State Diagram examples, including stopwatch
  - Project 4 overview
  - stopwatch-android-java
- **Android Application Development**
  - **Review of Android framework & activity life cycle**
  - **Android example apps – hello, simplebatch, clickcounter**

Android Framework/Architecture



Android Activity Life Cycle



Example Android Programs

- **hello-android-java – notification**
- **simplebatch-android-java – scrollable text output**
  - **The Android Activity Lifecycle**
- **simpledraw-android-java**
- **clickcounter-android-java**

Week 9 Topics

- More Design Patterns
  - Adapter, Façade, Observer, State, Command
- Model-View-Adapter Architectural Design Pattern
- The Stopwatch Example Program
  - UML State Diagrams (used with the State pattern)
  - State Diagram examples, including stopwatch
  - Project 4 overview
  - stopwatch-android-java
- Android Application Development
  - Android framework & activity life cycle
  - Android example apps – hello, simplebatch, clickcounter