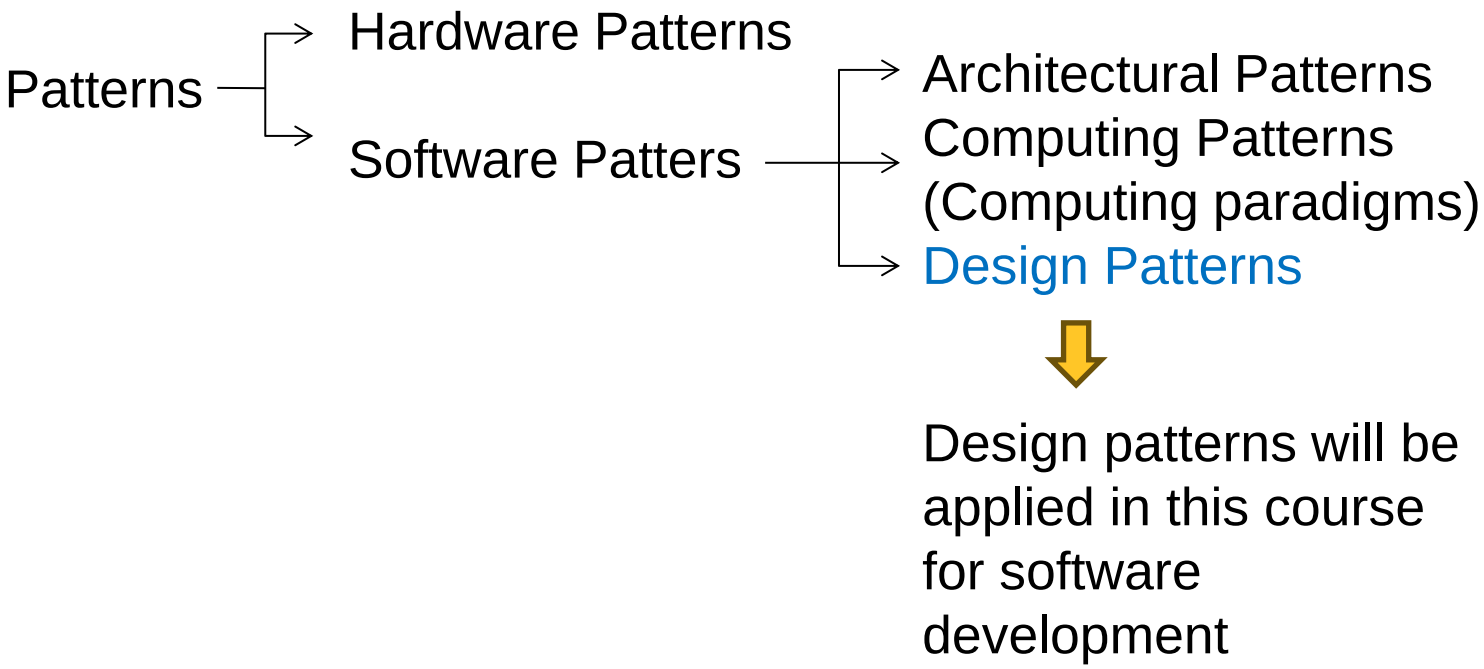
M1 L3
Patterns and Design Patterns

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Overview

- Patterns
- Design Patterns
  - Creational Patterns:
  - Structural Patterns:
  - Behavioral Patterns:
  - Concurrency Patterns:

Software Engineering: Patterns



Software Engineering: Design Patterns

A design pattern is a general reusable solution in software design.

A design pattern is not a finished design. It is

- a template for solving a problem that can be used in many different situations;
- an interface that can have different implementations.

Not all software patterns are design patterns. Design patterns deal specifically with problems at the level of software *design*.

Algorithms of solving a problem is not a part of the design pattern – They belong to the implementation detail / computing pattern.

Classification of Design Patterns

- **Creational Patterns**:
  Abstract factory, Factory method, Lazy initialization, Object pool, Singleton, Utility, …
- **Structural Patterns**:
  Adapter, Decorator, Façade, Proxy, …
- **Behavioral Patterns**:
  Command, Iterator, Mediator, Observer, State, …
- **Concurrency Patterns**:
  Active Object, Monitor, Read-Write Lock, Reactor, …

Creational Patterns

- **Abstract factory:** Provide an interface for creating families of objects without specifying their concrete classes.
- **Factory method:** Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.
- **Lazy initialization:** Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed.
- **Object pool:** Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Singleton:** Ensure a class only has one instance, and provide a global point of access to it.
- **Utility:** A class with a private constructor that contains static methods only.

Structural Patterns

- **Adapter:** Convert the interface of a class into another interface that clients expect.
- **Decorator:** Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.
- **Façade:** Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.
- **Proxy:** Provide a surrogate or placeholder for another object to control the access to it.

Behavioral Patterns

- **Command:** Encapsulate a request as an object, thereby letting you parameterize clients with different requests.
- **Iterator:** Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- **Mediator** between objects: Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently, e.g., we could use cse445instructor@asu.edu, instead of a concrete instructor email address, e.g., john.doe25@asu.edu
- **State:** Allow an object to alter its behavior when its internal state changes.
- **Observer:** Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Concurrency Patterns

- **Active Object**: It decouples method execution from method invocation. The goal is to introduce concurrency, by using asynchronous method invocation and a scheduler for handling requests.

- **Monitor:** It is an approach to synchronize two or more tasks that use a shared resource, usually an object.

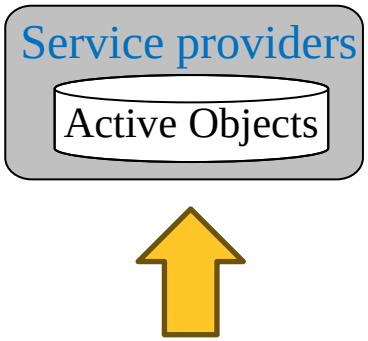
- **Read-Write Lock:** It allows concurrent read access to an object, but requires exclusive access for read-write and write-write operations.

- **Reactor:** It is a concurrent programming pattern for handling service requests delivered concurrently to a service handler by one or more inputs. The service handler then de-multiplexes the incoming requests and dispatches them synchronously to the associated request handlers.

Applications of These Design Patterns

The patterns will be further discussed in the context of their applications, in this course:

- **Concurrency Patterns**:
  Active Object, Monitor, Read-Write Lock, Reactor, …

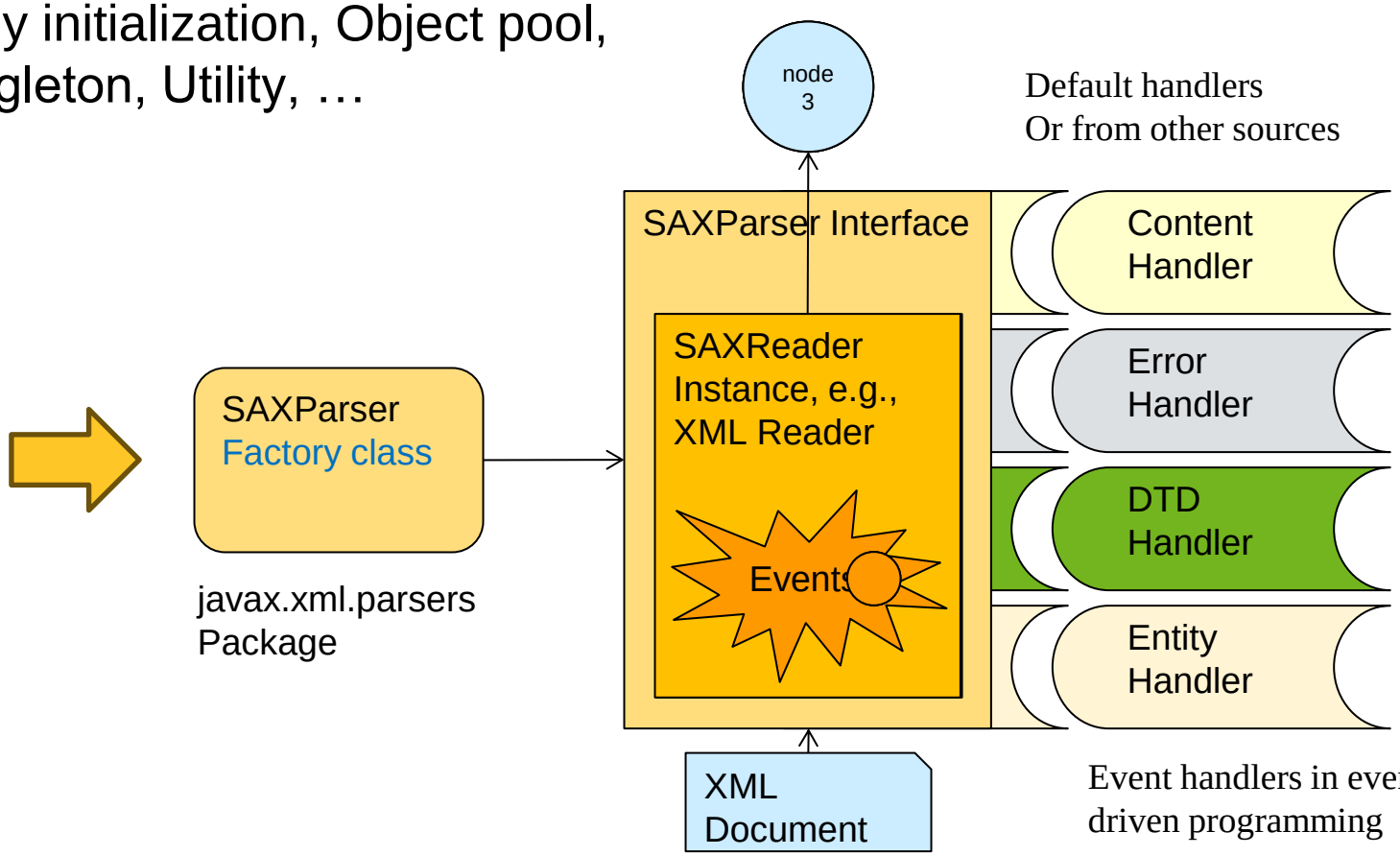



```
// thread producer
public void setBuffer( int val ) {
    lock (obj) {
        while (!writable)
            Monitor.Wait(obj);
        bufferCell = val;
        writeable = false;
        Monitor.Pulse(obj);
    }
}
```

Applications of These Design Patterns

The patterns will be further discussed in the context of their applications, in this course:

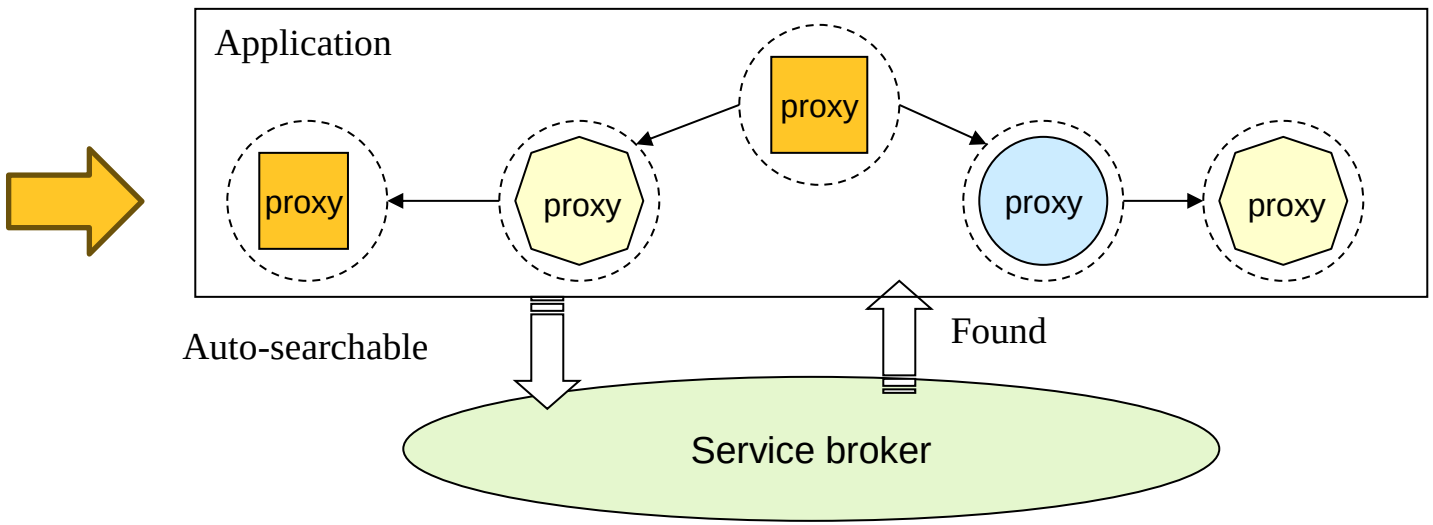
- **Creational Patterns**:
  Abstract factory, Factory method, Lazy initialization, Object pool, Singleton, Utility, …



Applications of These Design Patterns

The patterns will be further discussed in the context of their applications, in this course:

- **Structural Patterns**:
  Adapter, Decorator, Façade, Proxy, …



Applications of These Design Patterns

The patterns will be further discussed in the context of their applications, in this course:

- **Behavioral Patterns**:
  Command, Iterator, Mediator, Observer, State, …

```
connection.Open ();
StringBuilder builder = new StringBuilder ();
builder.Append ("select count (*) from users " + "where username = \' ");
builder.Append (username);
builder.Append ("\' and cast (rtrim (password) as " + "varbinary) = cast (\' ");
builder.Append (password);
builder.Append ("\' as varbinary)");
SqlCommand command = new SqlCommand (builder.ToString (), connection);
int count = (int) command.ExecuteScalar ();
return (count > 0);
```


Ira A. Fulton Schools of
Engineering
Arizona State University