Polymorphic System Architecture

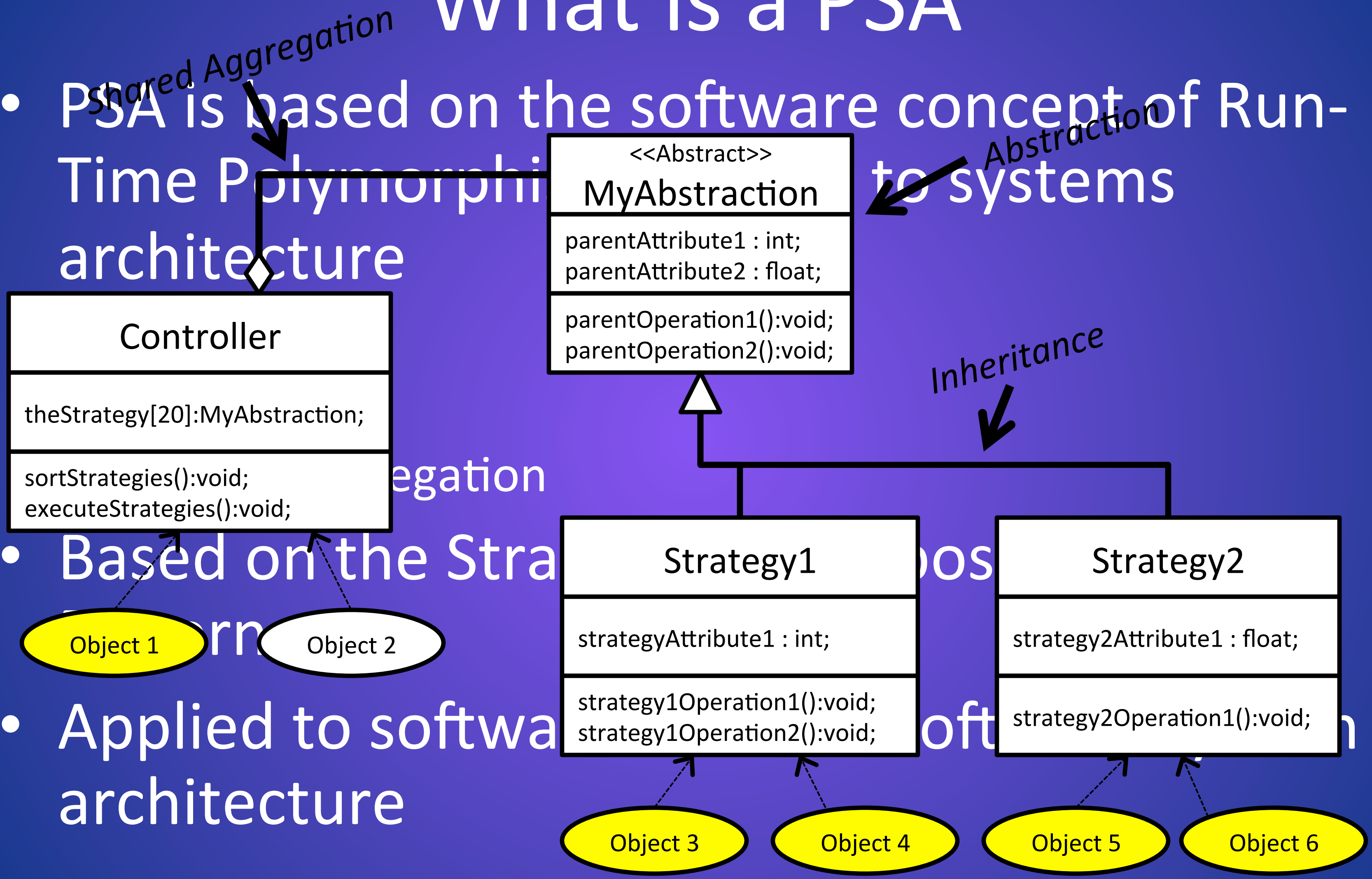
By Jeff Bryson

Software Engineer Staff

Lockheed Martin Simulation, Training, & Support

What is a PSA

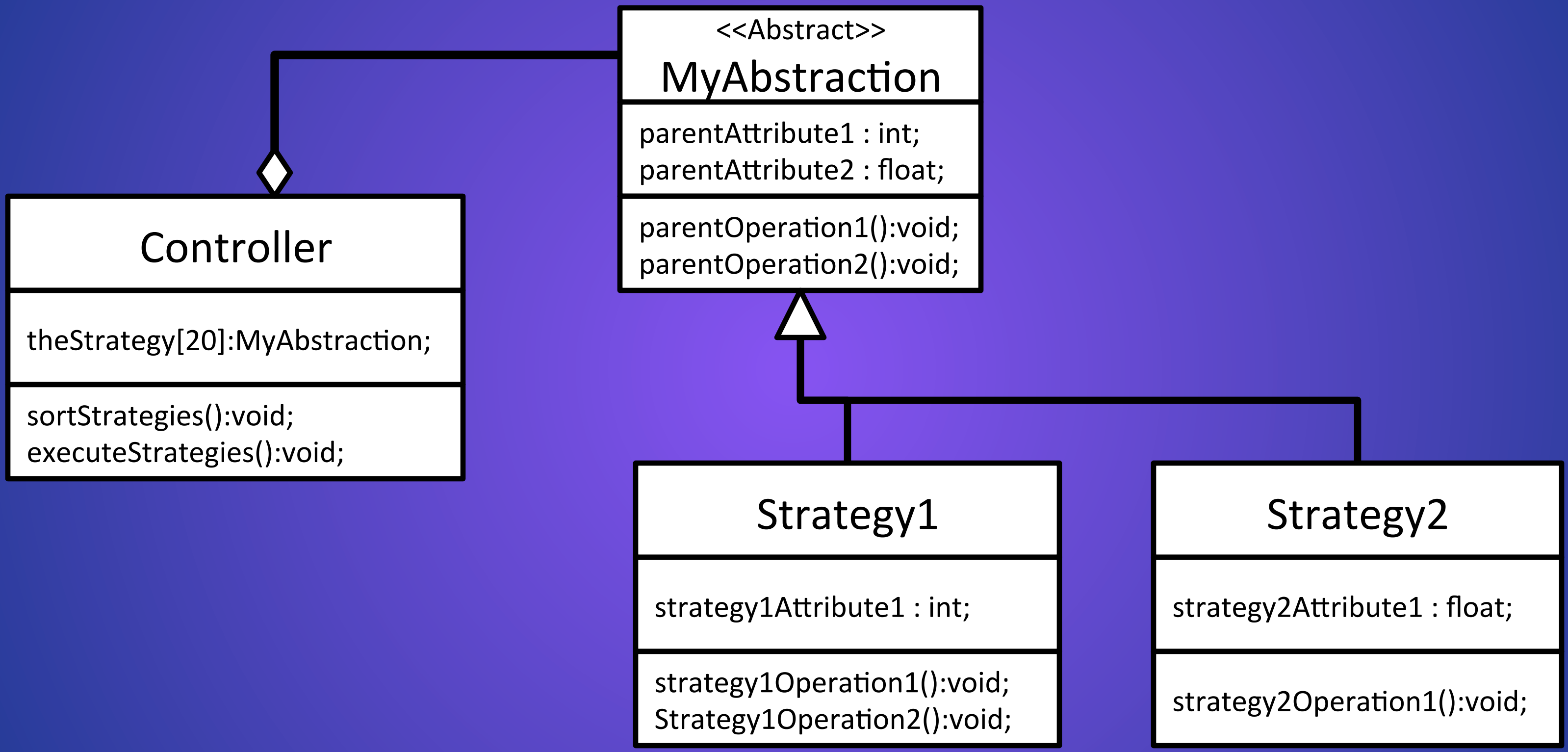
- PSA is based on the software concept of Run-Time Polymorphi to systems architecture
- egation
- Based on the Stra pos rn
- Applied to softwa oft n architecture



What is a PSA

- PSA is based on the software concept of Run-Time Polymorphism applied to systems architecture
  - Abstraction
  - Inheritance
  - Shared Aggregation
- Based on the Strategy & Composite Design Patterns

What is a PSA



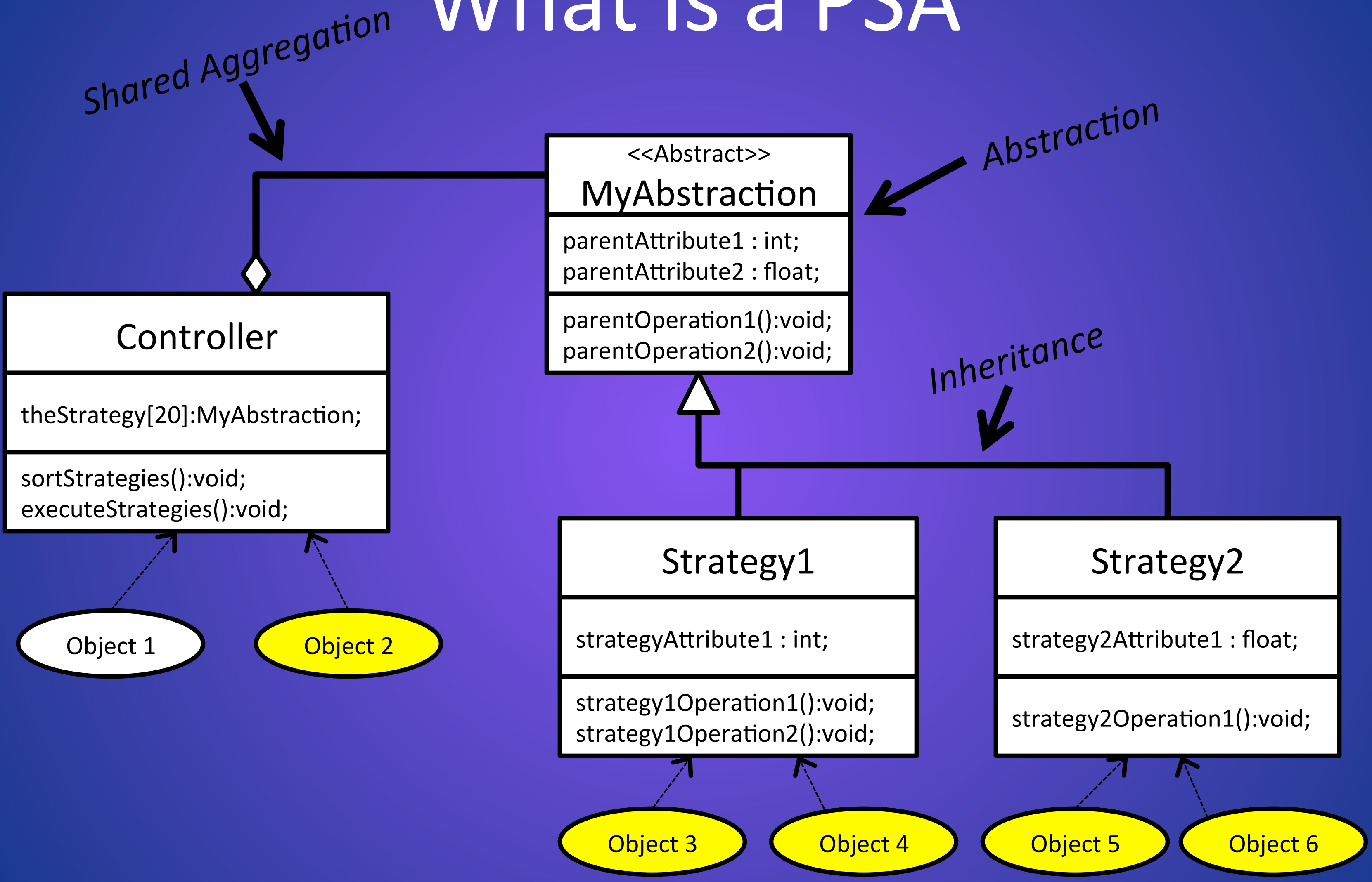
<<Abstract>>
MyAbstraction
parentAttribute1 : int;
parentAttribute2 : float;
parentOperation1():void;
parentOperation2():void;

Controller
theStrategy[20]:MyAbstraction;
sortStrategies():void;
executeStrategies():void;

Strategy1
strategy1Attribute1 : int;
strategy1Operation1():void;
Strategy1Operation2():void;

Strategy2
strategy2Attribute1 : float;
strategy2Operation1():void;

What is a PSA



Shared Aggregation

Abstraction

Inheritance

<<Abstract>>
MyAbstraction

parentAttribute1 : int;
parentAttribute2 : float;

parentOperation1():void;
parentOperation2():void;

Controller

theStrategy[20]:MyAbstraction;

sortStrategies():void;
executeStrategies():void;

Strategy1

strategyAttribute1 : int;

strategy1Operation1():void;
strategy1Operation2():void;

Strategy2

strategy2Attribute1 : float;

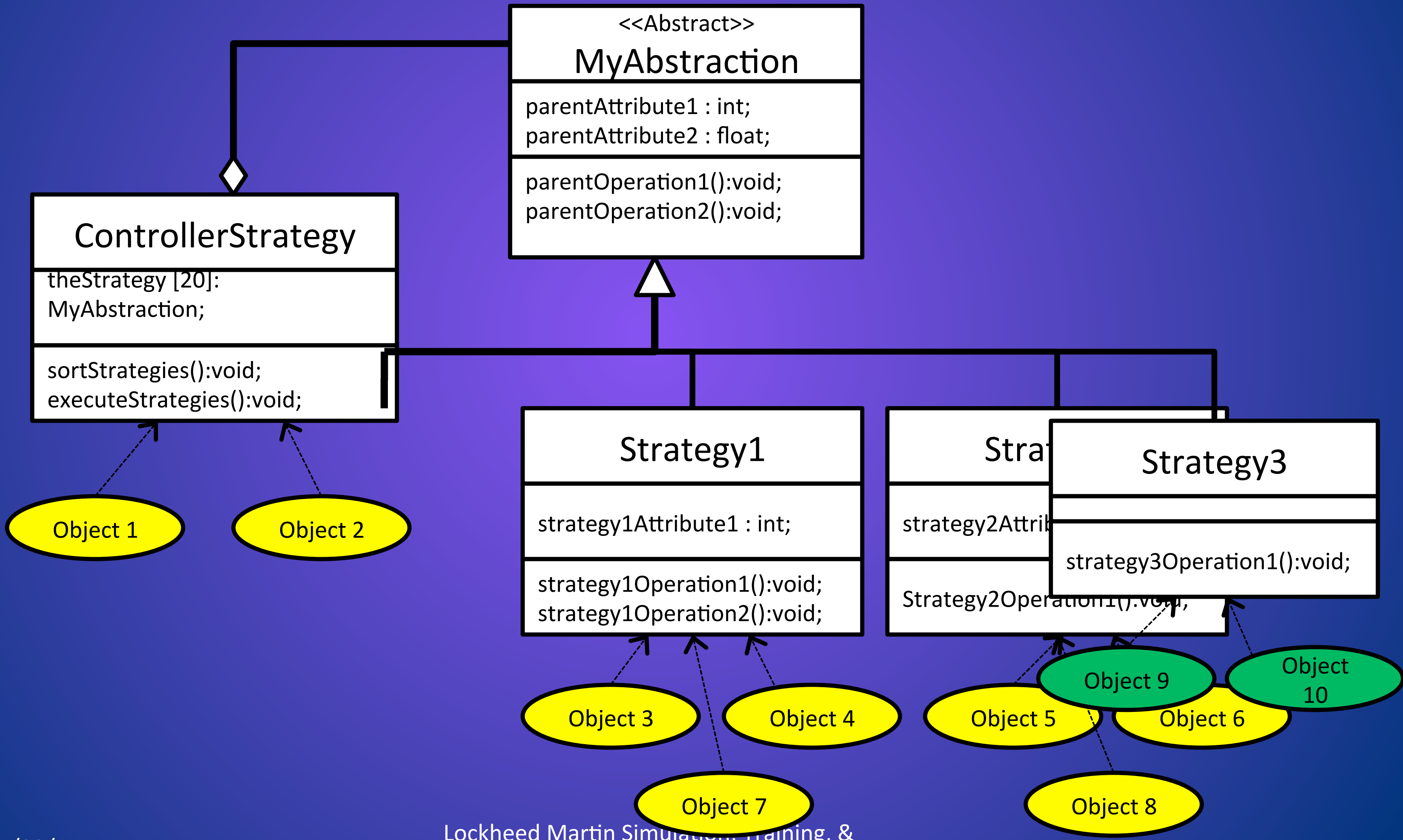
strategy2Operation1():void;

Purpose

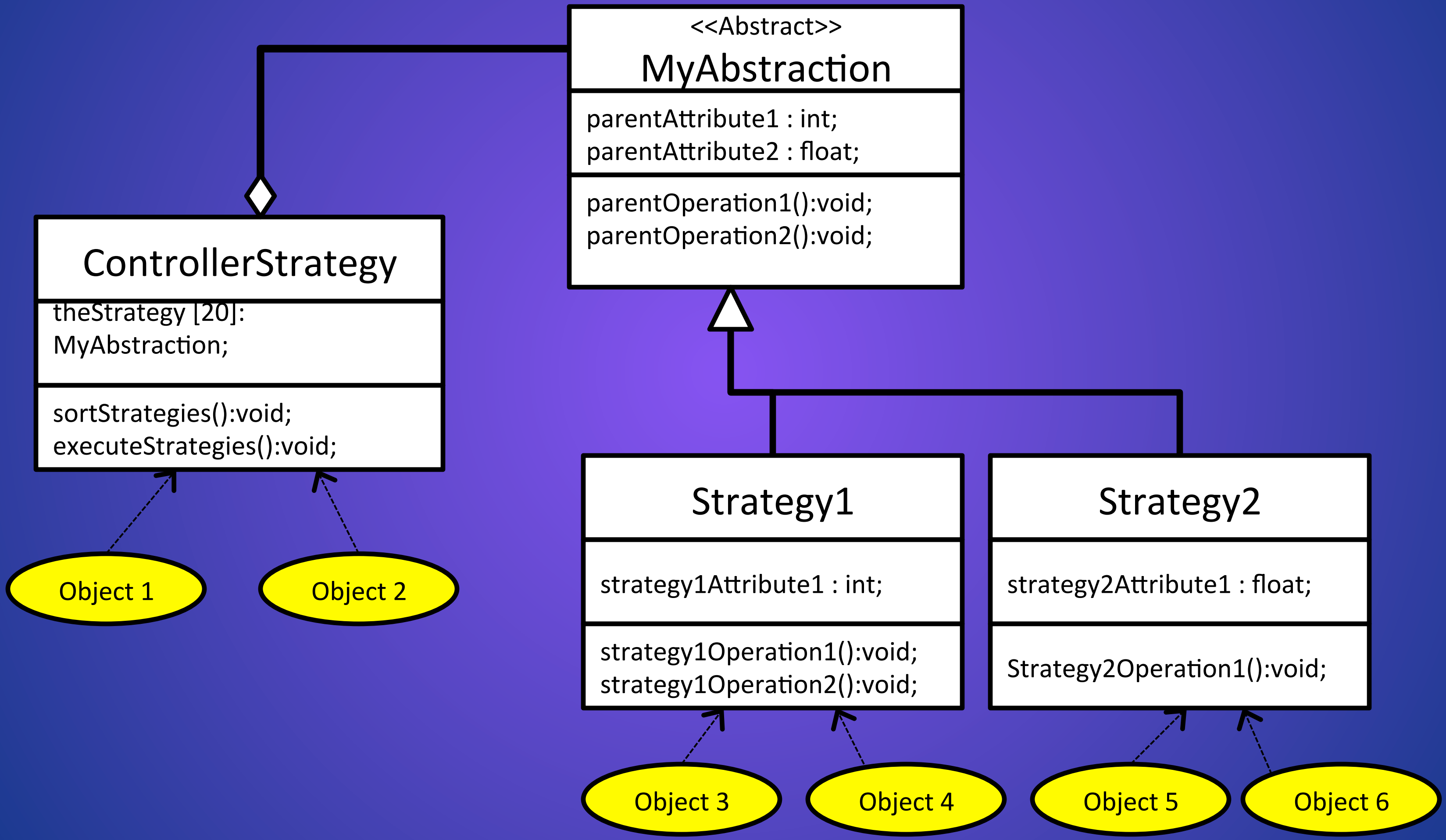
- Satisfy functionality with the system Architecture
  - Dynamic Reconfiguration of System Functionality
  - Plug-N-Play
  - Extendability
  - System Redundancy
  - System Reuse
  - Systems of Systems

- ~~Simplify~~ Reduce the complexity of the solution
  - Requires governances for the abstraction
  - Requires a polymorphic mediator
  - Requires architecture to be more than decomposition

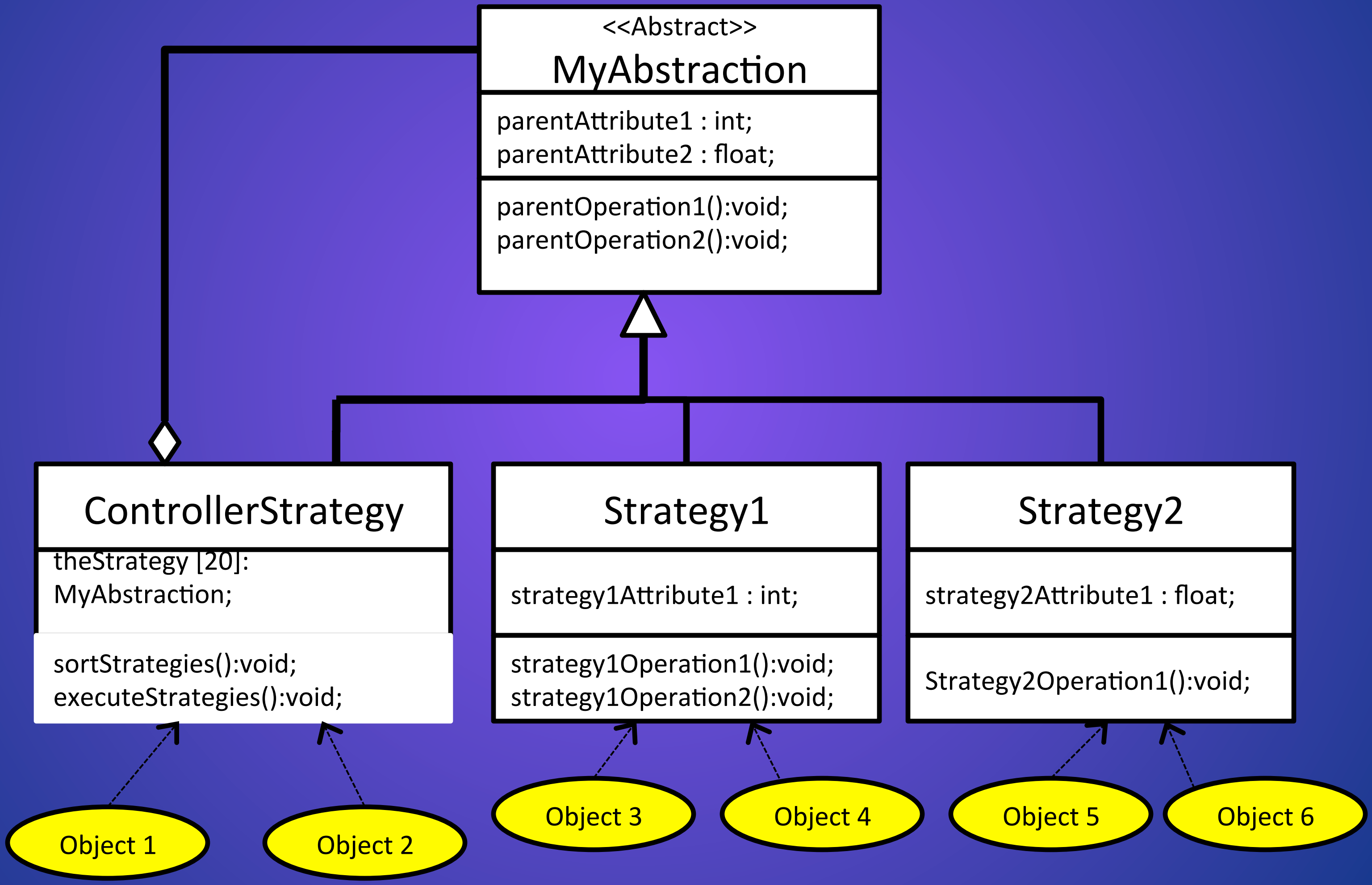
How does this provide extension



How does this provide extension



How does this provide extension



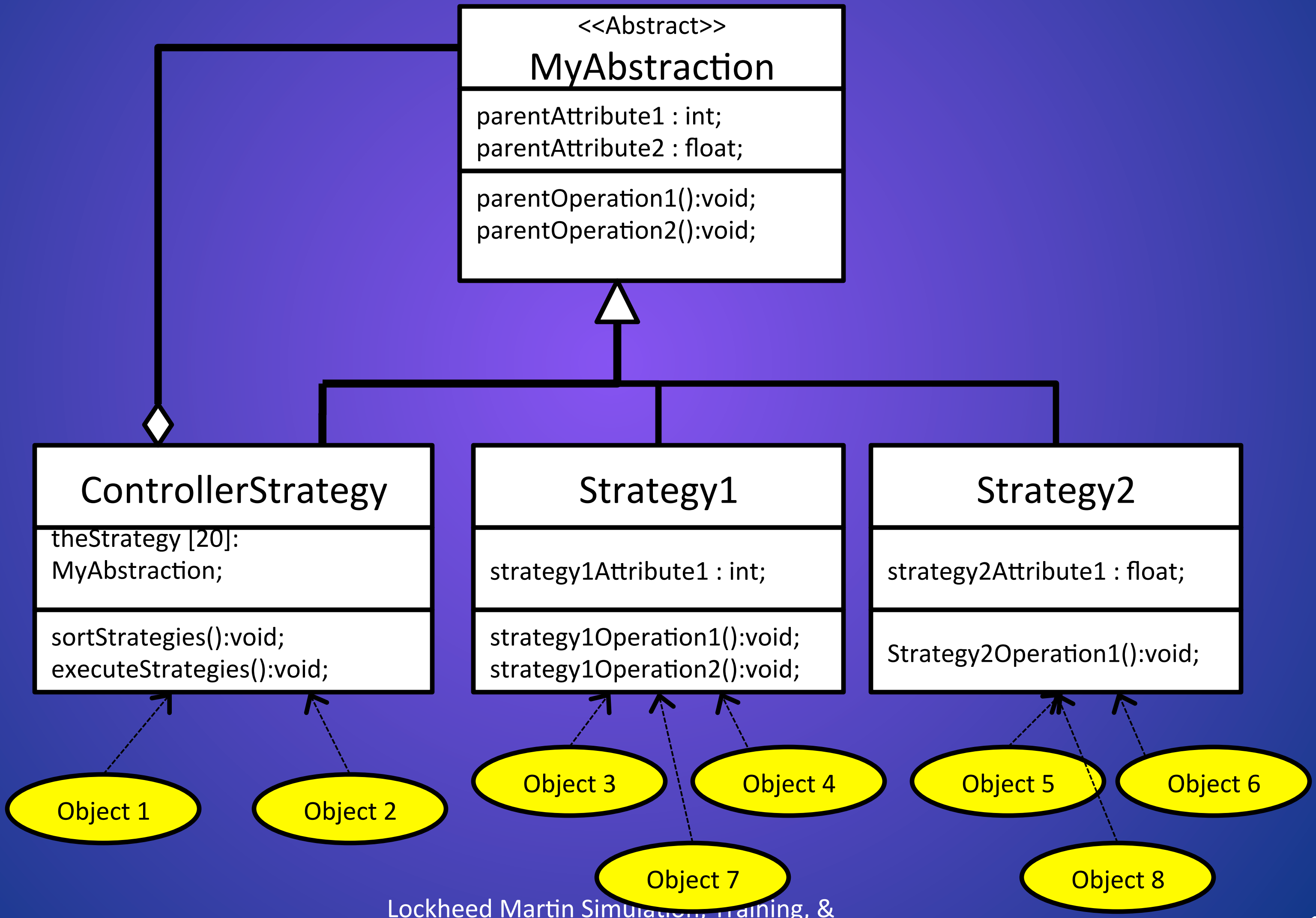
<<Abstract>>
MyAbstraction
parentAttribute1 : int;
parentAttribute2 : float;
parentOperation1():void;
parentOperation2():void;

ControllerStrategy
theStrategy [20]:
MyAbstraction;
sortStrategies():void;
executeStrategies():void;

Strategy1
strategy1Attribute1 : int;
strategy1Operation1():void;
strategy1Operation2():void;

Strategy2
strategy2Attribute1 : float;
Strategy2Operation1():void;

How does this provide extension



<<Abstract>>
MyAbstraction

parentAttribute1 : int;
parentAttribute2 : float;

parentOperation1():void;
parentOperation2():void;

ControllerStrategy

theStrategy [20]:
MyAbstraction;

sortStrategies():void;
executeStrategies():void;

Strategy1

strategy1Attribute1 : int;

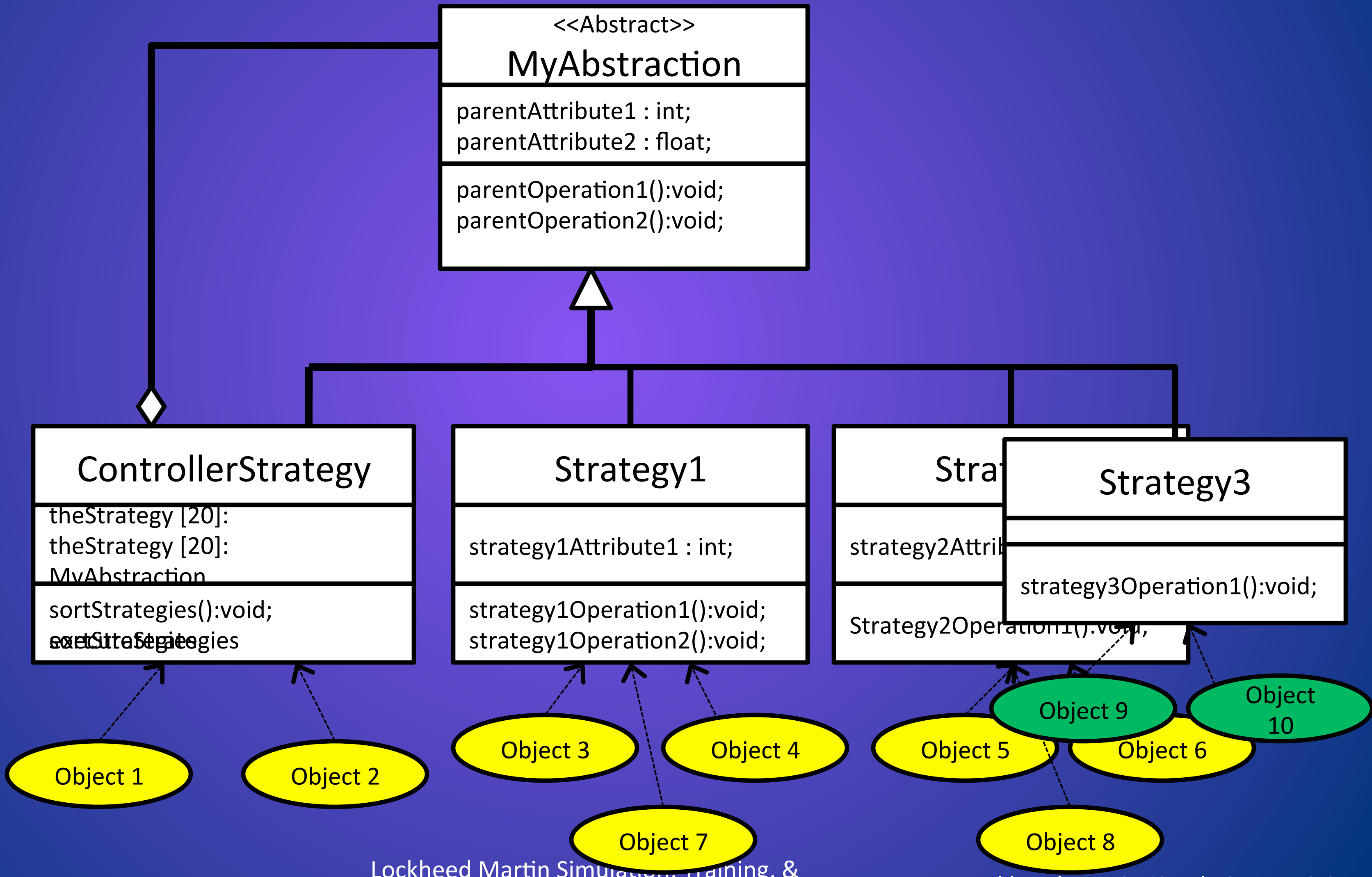
strategy1Operation1():void;
strategy1Operation2():void;

Strategy2

strategy2Attribute1 : float;

Strategy2Operation1():void;

How does this provide extension



<<Abstract>>
MyAbstraction

parentAttribute1 : int;
parentAttribute2 : float;

parentOperation1():void;
parentOperation2():void;

ControllerStrategy

theStrategy [20]:
theStrategy [20]:
MyAbstraction

sortStrategies():void;
executeStrategies

Strategy1

strategy1Attribute1 : int;

strategy1Operation1():void;
strategy1Operation2():void;

Stra

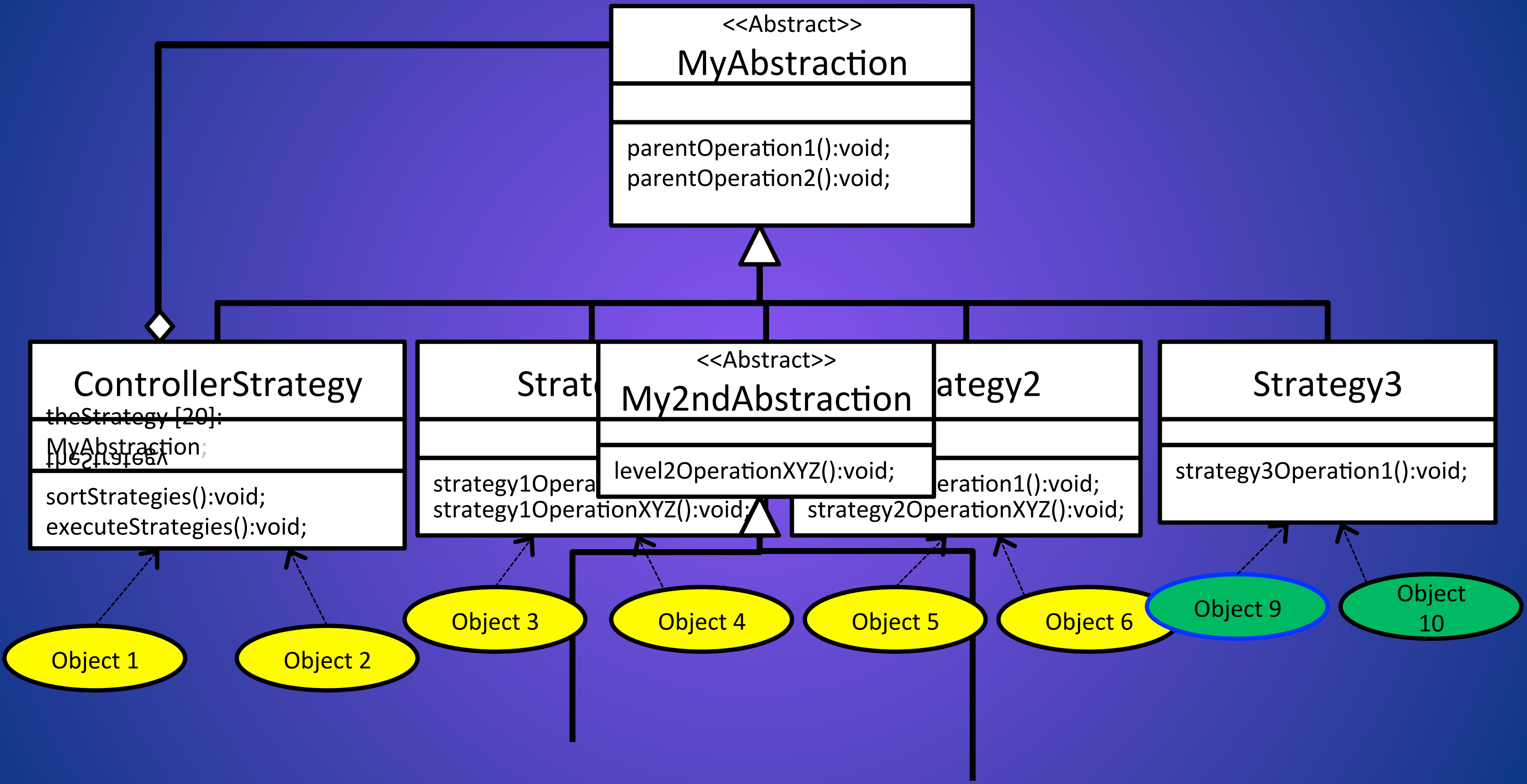
strategy2Attrib

Strategy2Operation1():void;

Strategy3

strategy3Operation1():void;

How does the architecture mature



<<Abstract>>
MyAbstraction

parentOperation1():void;
parentOperation2():void;

ControllerStrategy

theStrategy [20]: MyAbstraction;

sortStrategies():void;
executeStrategies():void;

Strat

strategy1Opera
strategy1OperationXYZ():void;

<<Abstract>>
My2ndAbstraction

level2OperationXYZ():void;

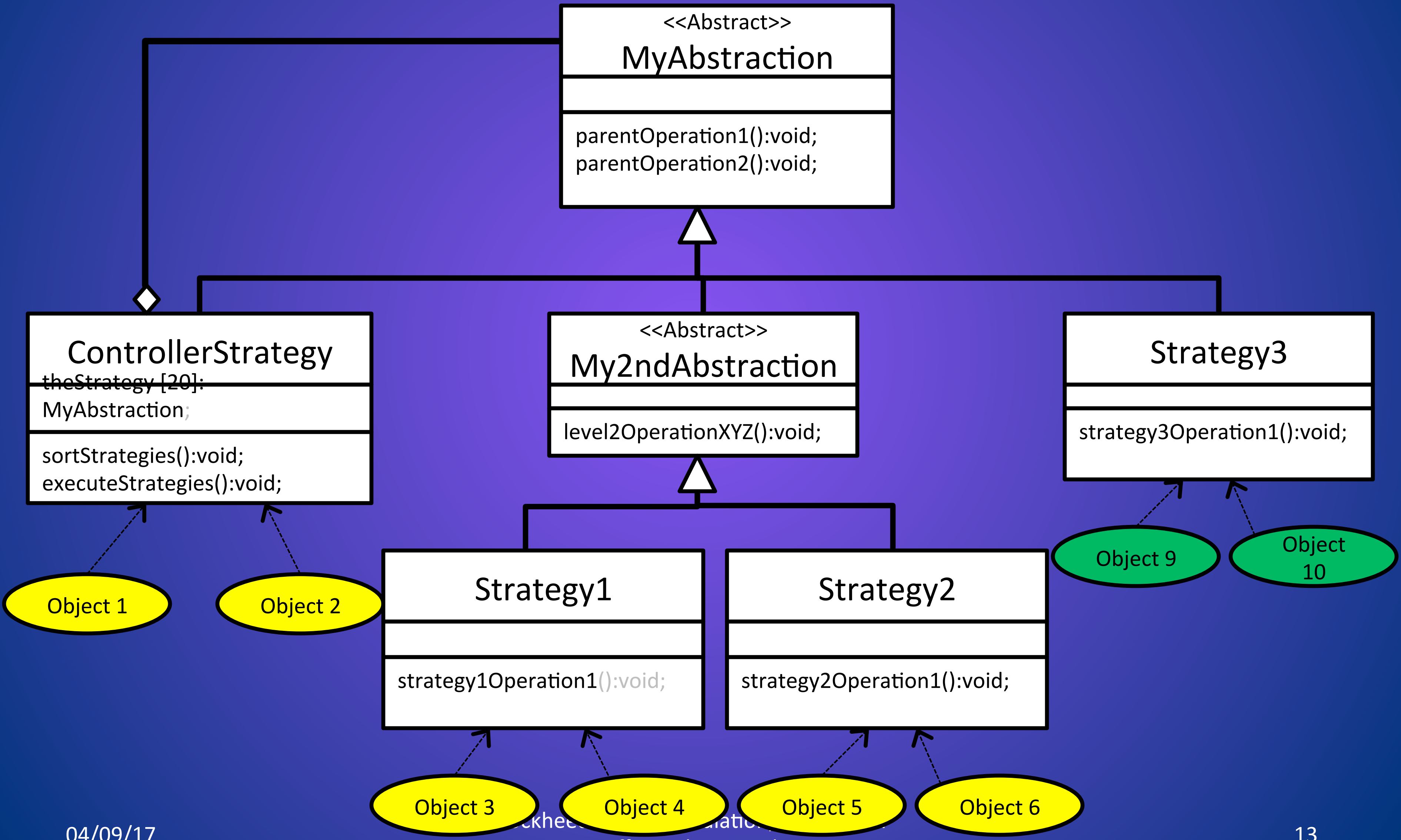
ategy2

eration1():void;
strategy2OperationXYZ():void;

Strategy3

strategy3Operation1():void;

How does the architecture mature

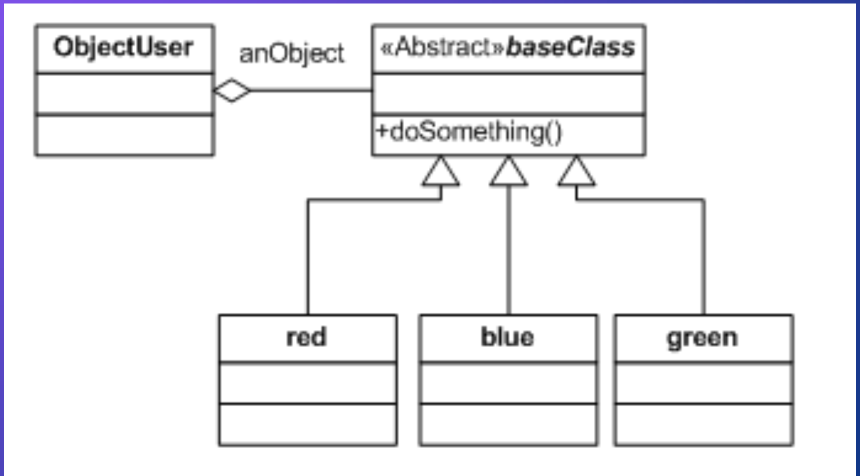


How do we apply polymorphic behavior

- Structured Logic

```
Case value is
 when red =>
    doSomething();
when blue =>
    doSomethingElse();
when green =>
    doAnotherthing();
end case;
```

- Polymorphic Logic
  - The ‘value’ translates to the abstract interface reference.
  - Each enumerated value (red, blue, and green) becomes a child classes derived from the abstract interface.



Metrics

- At this time there are only metrics to identify if these patterns are being used.
  - Counting the relationships of:
    - Shared aggregation to
    - Defined abstraction with
    - Inheritance
- Opportunities for using these patterns can also be identified
  - Number of Case statement type logic in the system can be identified

Value Added

- Extendable/Reusable System Designs
- Dynamic reconfiguration
- Satisfy Requirements with the System Architecture
- An architecture that matures over time instead of becoming obsolete

Take Away

- Polymorphism can be applied to both software and non-software aspects of the system
- This architecture allows for specific types of requirements to be satisfied at the system level
  - The most valuable of these is the ability to create a reusable system that can mature with age.

Questions/Comments

- ?