Design Patterns

Bridge

Dr. Chad Williams
Central Connecticut State University

Design pattern: Bridge

- **Category**: Structural design pattern
- **Intent**:
  - Decouple an abstraction from its implementation so the two can vary independently
- **Motivation**
  - When abstraction can have multiple different variations of implementation, common approach is abstraction with multiple child implementations. However, may require more flexibility - implementation may need to change at runtime or be a combination of implementations

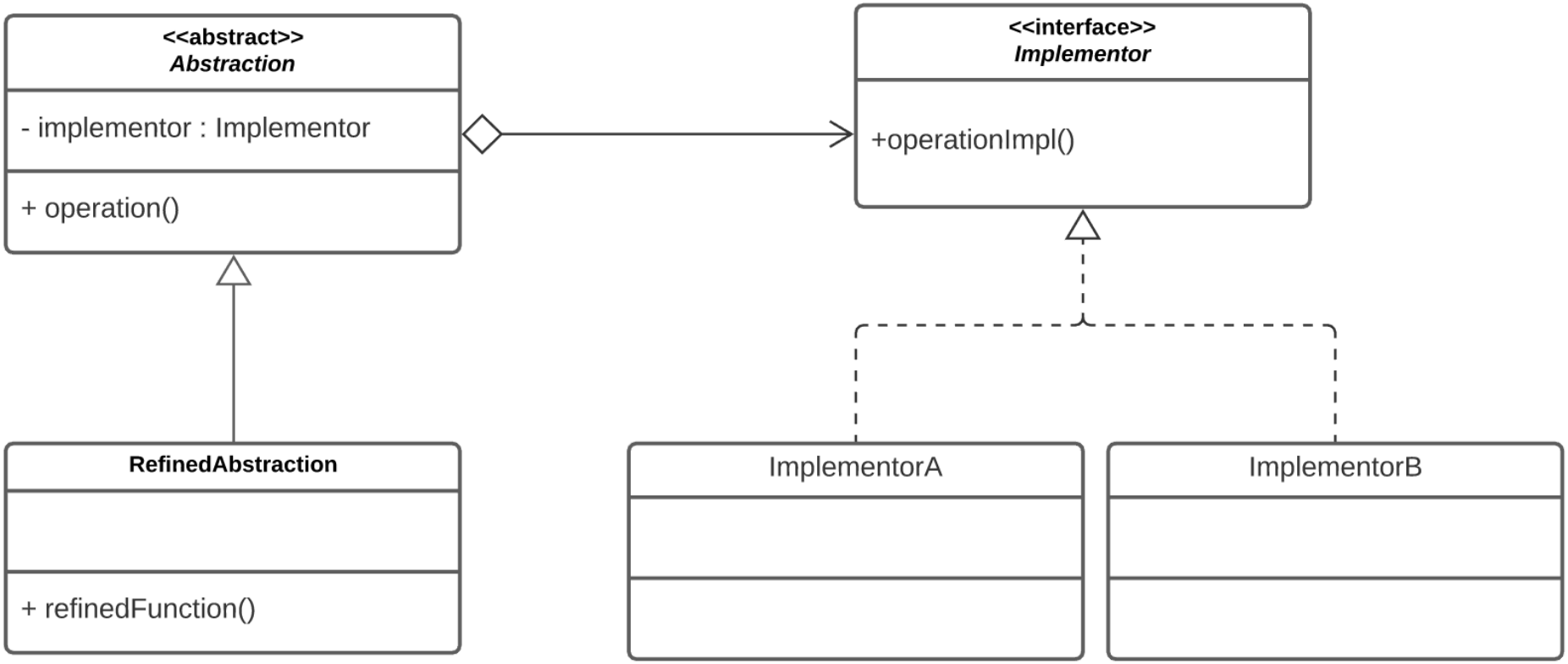
Applicability

- Use Bridge pattern when:
  - Want to avoid permanent binding between an abstraction and an implementation – such as selected/switched at runtime
  - Both abstraction and implementations should be extensible by subclassing – allow to extend independently
  - Want to share implementation among multiple objects that don't share a parent

Participants

- Abstraction (abstract class)
  - Defines the abstraction interface
  - Maintains reference in Implementor(s)
- Refined Abstraction
  - Extends interface defined by abstraction
- Implementor (interface)
  - Define interface for the implementing classes, can be some of same methods as abstraction, or more primitive methods and abstraction provides higher order operations on the primatives
- Concrete Implementor
  - Defines concrete implementation

Bridge UML


<<abstract>>
Abstraction
- implementor : Implementor
+ operation()
<<interface>>
Implementor
+operationImpl()
RefinedAbstraction
+ refinedFunction()
ImplementorA
ImplementorB

Bridge examples

- Serializable class
  - Family of classes that you want to extend for functionality, ability to use common mechanism for serialization that should be changeable (XML,JSON,binary)
- Logging class
  - Abstract class that provides generic logging interfaces, allows logging mechanism to be switchable (console/file/alert), class extendable to support multiple

Consequences

+ Improved extensibility
  - Generalizations and sub classing can be done of implementations sub classing

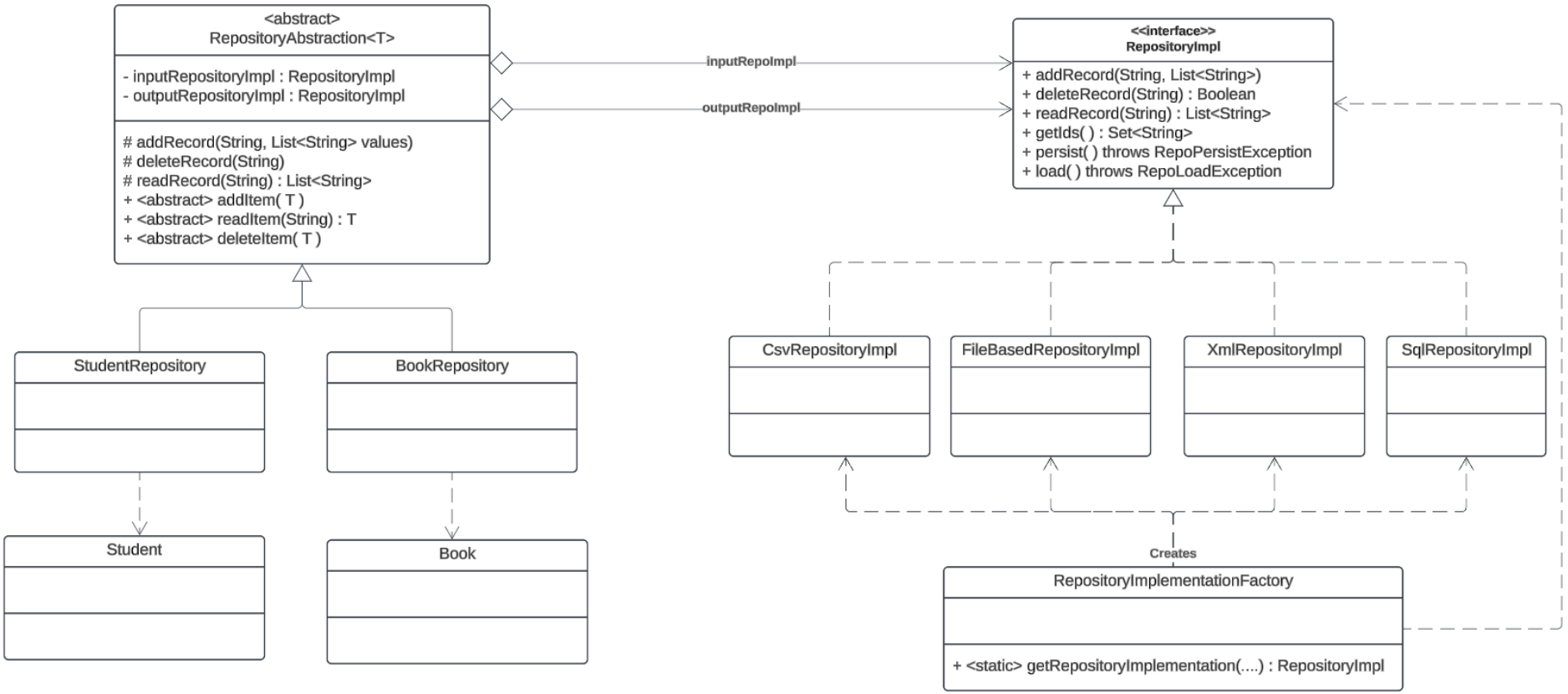
+ Can shield clients from complexity of changing implementations

- More complex to understand/read, so may add unnecessary complexity if abstraction and implementation are not expected to vary independently

Implementation considerations

- Implementor
  - Pattern is suited to have multiple different implementors so components of specific aspects of implementations can be varied at runtime independently
  - Creating implementor
    - Can have refined abstraction specify default implementation in constructor
    - Alternatively use Abstract Factory to make abstraction completely independent from selection of implementation

Bridge pattern example



Bridge vs Strategy vs State

- Bridge has a lot of similarity in UML to Strategy and State
  - Key differences:
    - State - you are changing the behavior of the entire StateContext class, the context is just a vessel for the current State
    - Strategy – Providing a switchable algorithm for a step for the context
    - Bridge – Provide commonality that can change for subclasses of the abstraction