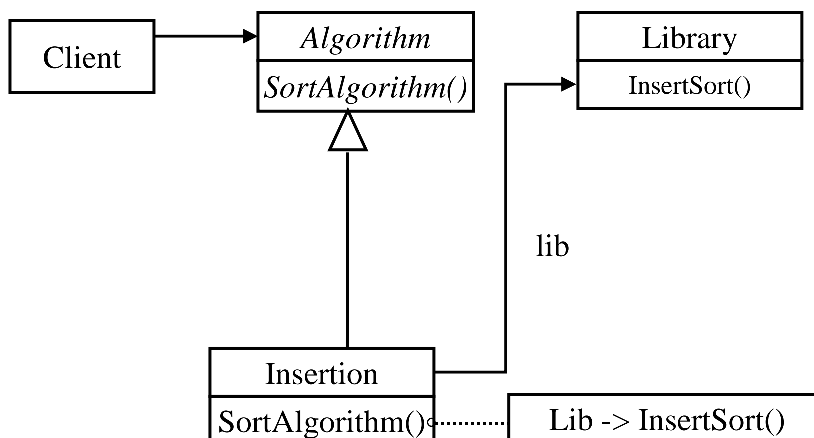
Sorting Example

- **Requirement 3:** since there is a library function that implements insertion sort, we want to reuse this function. However, the interface of this function is different.
- Use the Adapter pattern to solve the problem

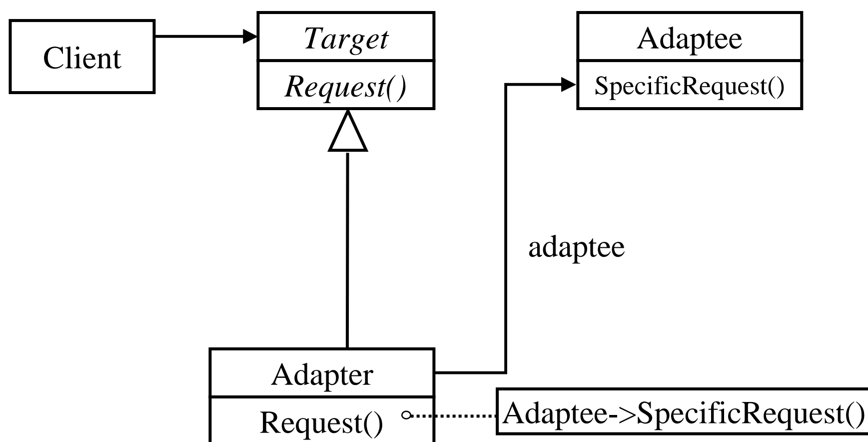
Adapter Pattern



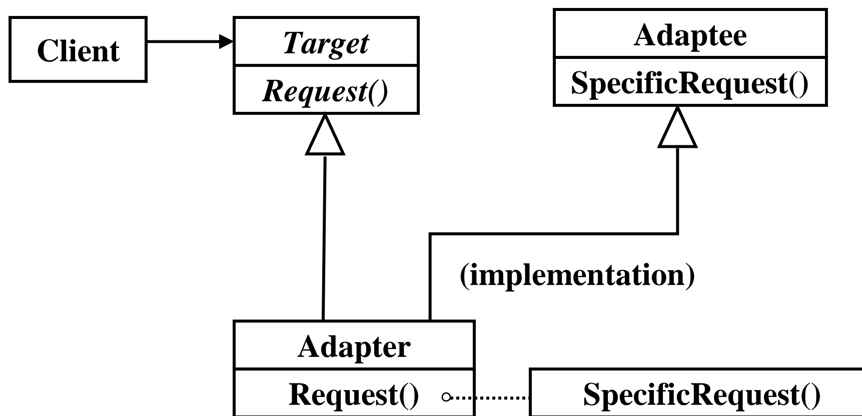
Adapter Pattern

- **Intent:** Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
- **Motivation:** When we want to reuse classes in an application that expects classes with a different interface, we do not want (and often cannot) to change the reusable classes to suit our application.

Adapter (Composition)



Adapter (Multi-inheritance)



43

Participants of Adapter

- **Target:** defines the application-specific interface that clients use.
- **Client:** collaborates with objects conforming to the target interface.
- **Adaptee:** defines an existing interface that needs adapting.
- **Adapter:** adapts the interface of the adaptee to the target interface.

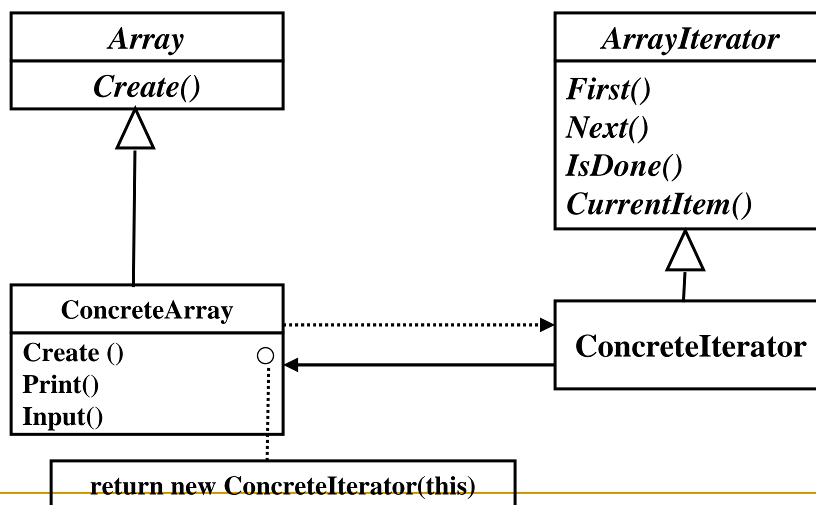
44

Sorting Example

- **Requirement 4:** we want to be able to print the sorted list without exploring the internal structure of the list
- Use the Iterator pattern

45

Iterator Pattern



46

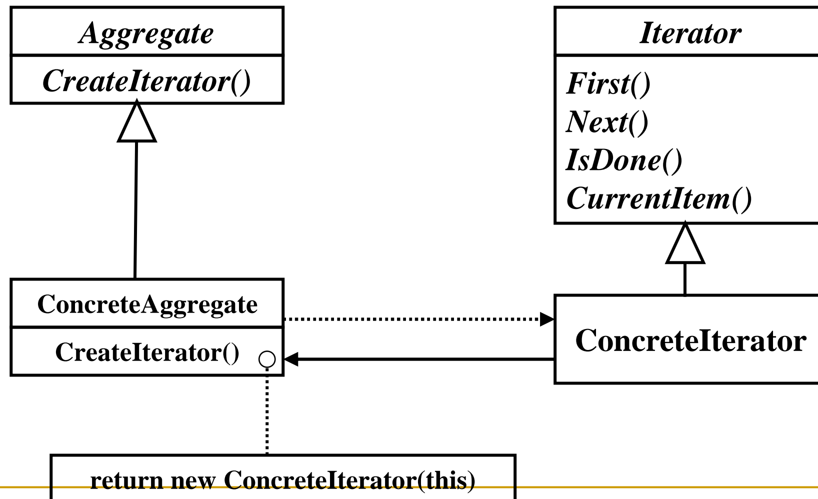
Intent of Iterator

- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- Move the responsibility for access and traversal from the aggregate object to the iterator object.

Motivation of Iterator

- One might want to traverse an aggregate object in different ways.
- One might want to have more than one traversal pending on the same aggregate object.
- Not all types of traversals can be anticipated a priori.
- One should not bloat the interface of the aggregate object with all these traversals.

Iterator Pattern



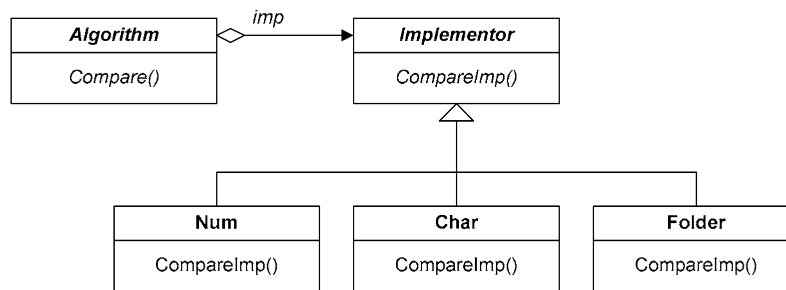
49

Participants of Iterator

- **Iterator:** defines an interface for accessing and traversing elements.
- **Concrete Iterator:** implements an iterator interface and keeps track of the current position in the traversal of the aggregate.
- **Aggregate:** defines an interface for creating an iterator object.
- **Concrete Aggregate:** implements the iterator creation interface to return an instance of the proper concrete iterator.

50

Compare Abstraction Using Bridge Pattern



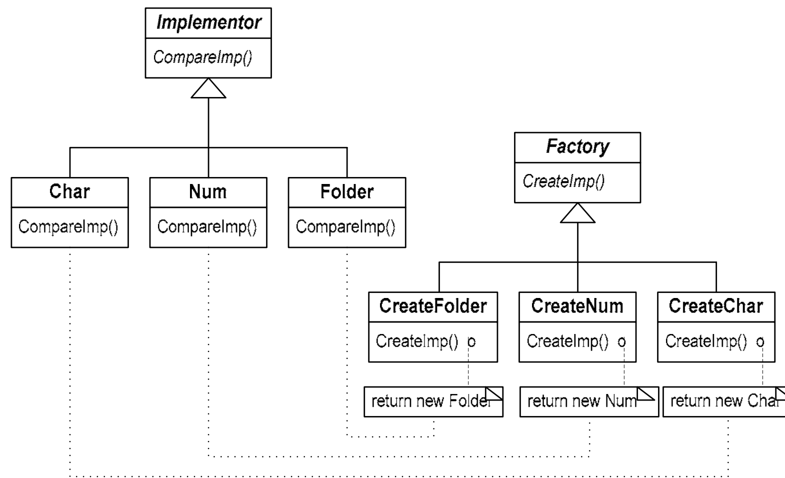
51

Sorting Example

- **Requirement 5:** we want to create families of different compare implementation for different types of elements.
- Use the Abstract Factory pattern

52

Abstract Factory Pattern



Abstract Factory Pattern

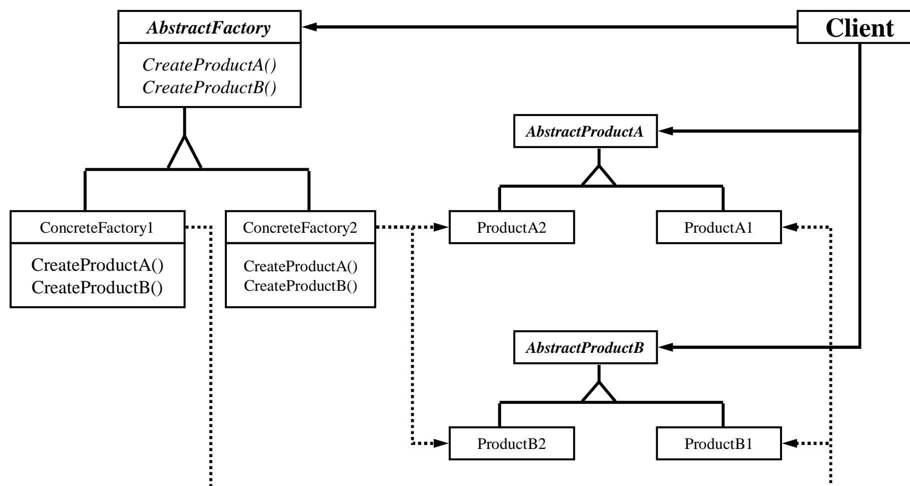
- **Intent:** Provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Abstract Factory Pattern

- **Motivation:**
  - Sometimes we have systems that support different representations depending on external factors.
  - There is an *Abstract Factory* that provides an interface for the client. In this way the client can obtain a specific object through this abstract interface.

55

Abstract Factory Pattern



56

Participants of Abstract Factory Pattern

- **Abstract factory:** declares an interface for operations that create abstract product objects.
- **Concrete factory:** implements the operations to create concrete product objects.

57

Participants of Abstract Factory Pattern

- **Abstract product:** declares an interface for a type of product object.
- **Concrete product:** defines a product object to be declared by the corresponding concrete factory. (Implements the abstract product interface).
- **Client:** uses only interfaces declared by abstract factory and abstract product classes.

58

The Integrated Design of Multi-Sorting Algorithms Example

Sorting Example