

CS560

## Lecture: Design Patterns II

Includes slides by E. Gamma et al., 1995

## Classification of GoF Design Pattern

Creational	Structural	Behavioural
Factory Method Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Flyweight Facade Proxy	Interpreter Template Method Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

# Observer Pattern I

- *Name:* Observer (Behavioural)
- *Problem:* A side-effect of partitioning a system into cooperating classes is the need to maintain consistency between related objects. You don't want to make the classes too tightly coupled.
- *Intent:* Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- *Context:* Many graphical user interfaces separate the presentational aspects of the user interface from underlying application data. When the user changes information in the application the user objects need to be informed.
- *Example:* In an auction, each bidder possesses a numbered paddle that is used to indicate a bid. The auctioneer starts the bidding, and observes when a paddle is raised to accept the bid. Bid acceptance changes the bid price, which is broadcast to all of the bidders in the form of a new bid.

# Observer Pattern II

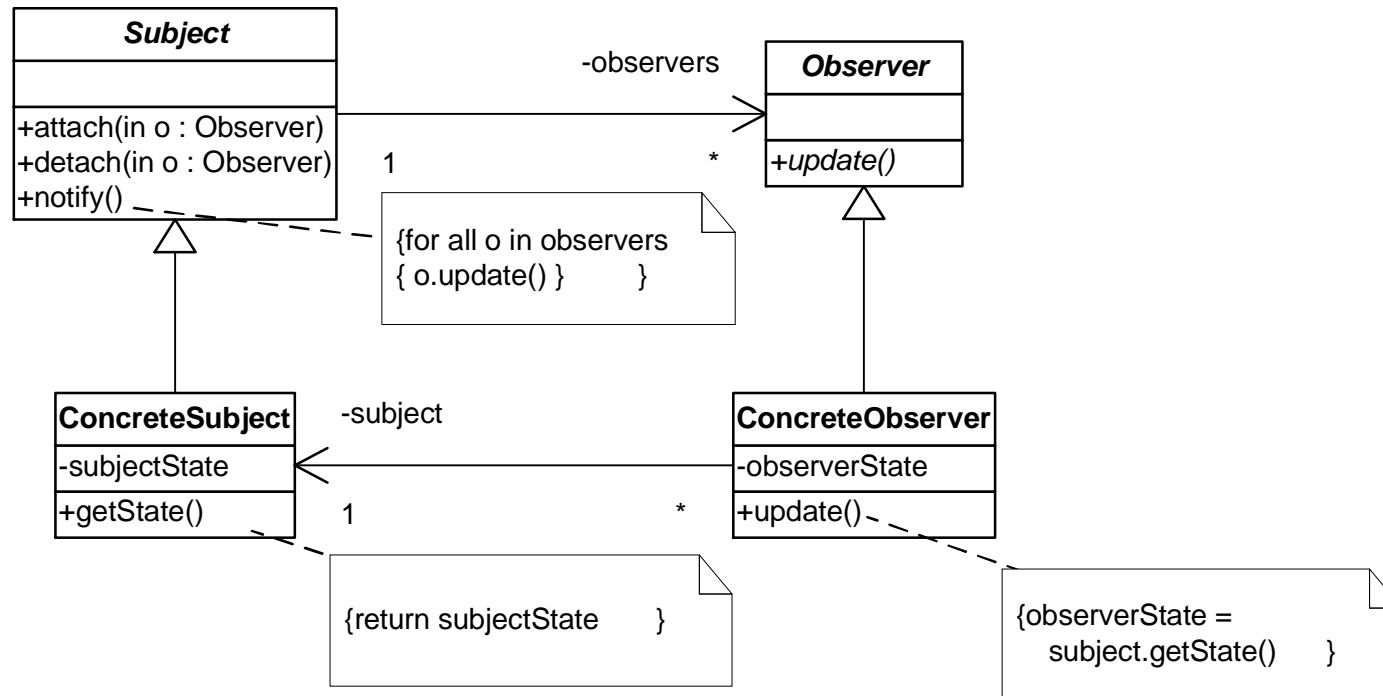
- *Solution:* Any number of observers observe a Subject. Observer class defines an updating interface for objects that should be notified of changes in a subject.
- *Resulting context:*
  - Coupling between Subject and Observer
  - Broadcast communication.
- *Consequences:*
  - Modularity: subject and observers may vary independently
  - Extensibility: can define and add any number of observers
  - Customizability: different observers provide different views of subject
- Implementation
  - Subject-observer mapping
  - Registering modifications of interest explicitly

# Observer Pattern III

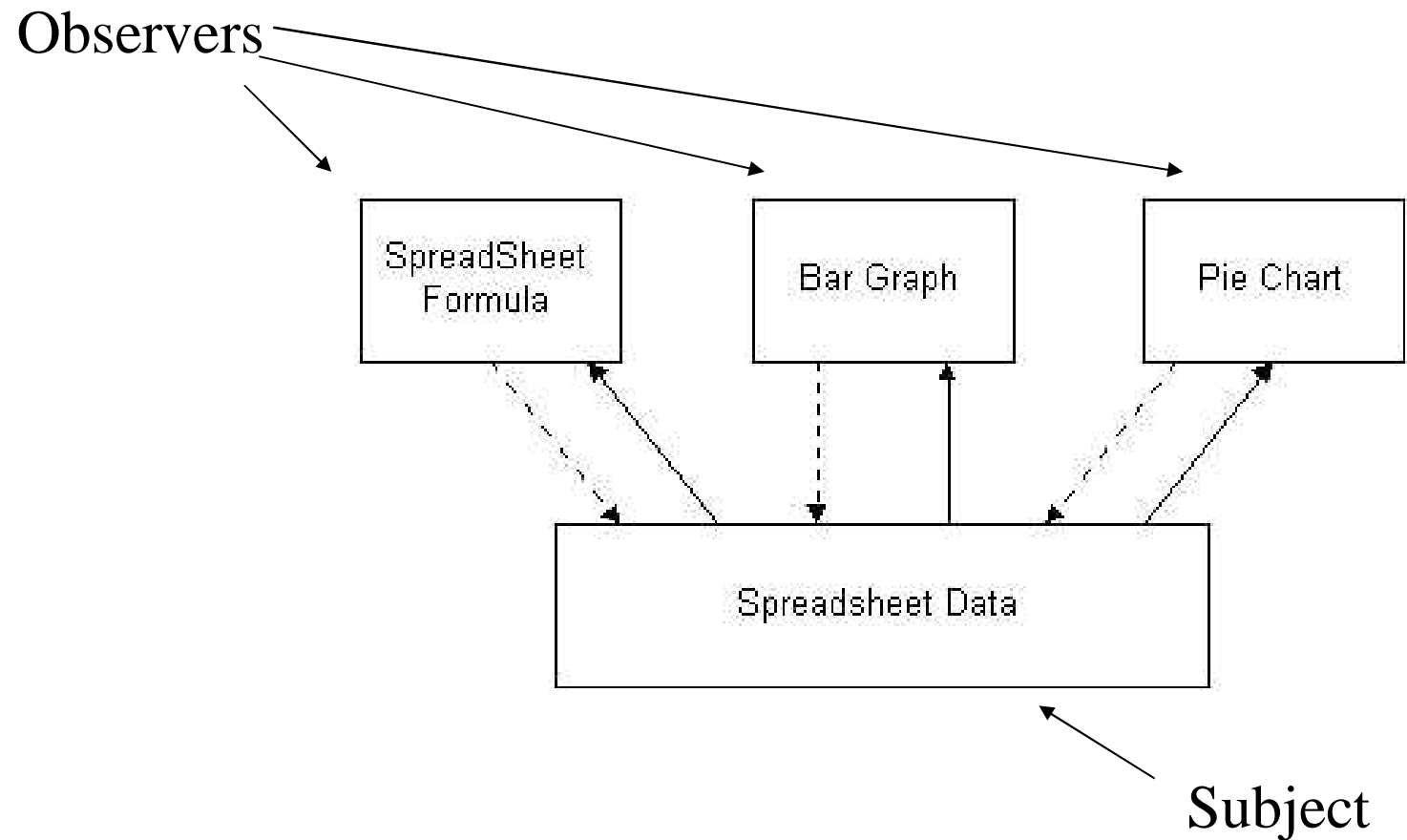
- *Also Known As:* Publish-subscribe, Dependents
- *Known Uses:* CORBA event notification; Cache coherency algorithms; Model-View separation in MVC; ET++, etc.; Interviews (subjects and views); Andrew file system (data objects and views)
- *Related Pattern:* Mediator (A ChangeManager encapsulates complex update semantics between subjects and objects.)
- *Ref:* Design Patterns, E. Gamma et al., Addison-Wesley, 1994, pp 293-303.

# Observer Class Diagram

- Structure



# Schematic Observer Example




# Observer - C++ Sample Code

```
class Subject {
public:
    virtual ~Subject();
    virtual void Attach(Observer* o);
    virtual void Detach(Observer* o);
    virtual void Notify();
protected:
    Subject();
private:
    List<Observer*> *_observers;
};

void Subject::Attach (Observer* o) {
    _observers->Insert(_observers->end(), o);
}

void Subject::Detach (Observer* o) {
    _observers->remove(o);
}

void Subject::Notify () {
    ListIterator<Observer*>i(_observers);
    for (i.First(); !i.IsDone(); i.Next()) {
        i.CurrentItem()->Update(this);
    }
}
```



```
class Observer {
public:
    virtual ~Observer();
    virtual void Update(Subject* theChangeSubject) = 0;
protected:
    Observer();
};

class ClockTimer : public Subject {
public:
    ClockTimer();
    virtual int GetHour();
    virtual int GetMinute();
    virtual int GetSecond();
    void Tick();
};

void ClockTimer::Tick() {
    // update internal time-keeping state
    // ...
    Notify();
}
```



# Observer – C++ Sample Code

```
class DigitalClock: public Observer {
public:
    DigitalClock(ClockTimer *);
    ~DigitalClock();
    void Update(Subject *);
    void Draw();
private:
    ClockTimer *_subject;
};

DigitalClock::DigitalClock (ClockTimer *s)
{
    _subject = s;
    _subject->Attach(this);
}

DigitalClock::~DigitalClock ()
{
    _subject->Detach(this);
}

void DigitalClock::Update (Subject *theChangedSubject)
{
    if(theChangedSubject == _subject)
        draw();
}

void DigitalClock::Draw ()
{
    int hour = _subject->GetHour();
    int minute = _subject->GetMinute();
    int second = _subject->GetSecond();
    // draw operation
}
```

```
class AnalogClock: public Observer {
public:
    AnalogClock(ClockTimer *);
    ~AnalogClock();
    void Update(Subject *);
    void Draw();
private:
    ClockTimer *_subject;
};

int main(void)
{
    ClockTimer *timer = new ClockTimer;
    AnalogClock *analogClock =
        new AnalogClock(timer);
    DigitalClock *digitalClock =
        new DigitalClock(timer);
    timer->Tick();
    return 0;
}
```

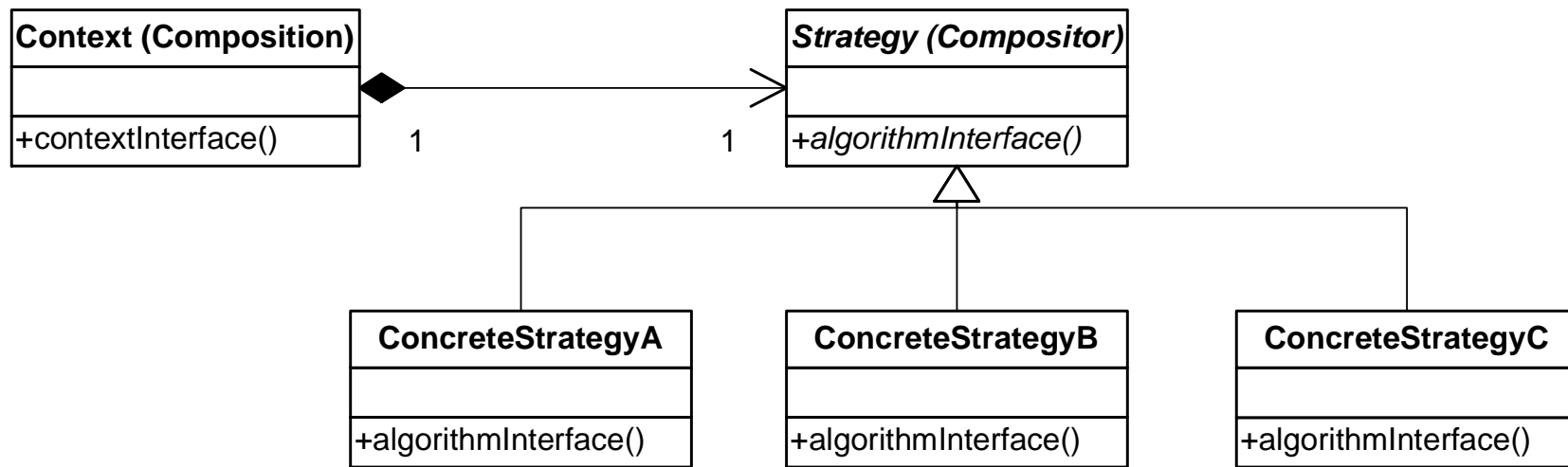
# Strategy (*Behavioural*)

- *Intent*: Define a family of algorithms, encapsulate each one, and make them interchangeable to let clients and algorithms vary independently
- *Applicability*:
  - When an object should be configurable with one of several algorithms,  
*and* all algorithms can be encapsulated,  
*and* one interface covers all encapsulations

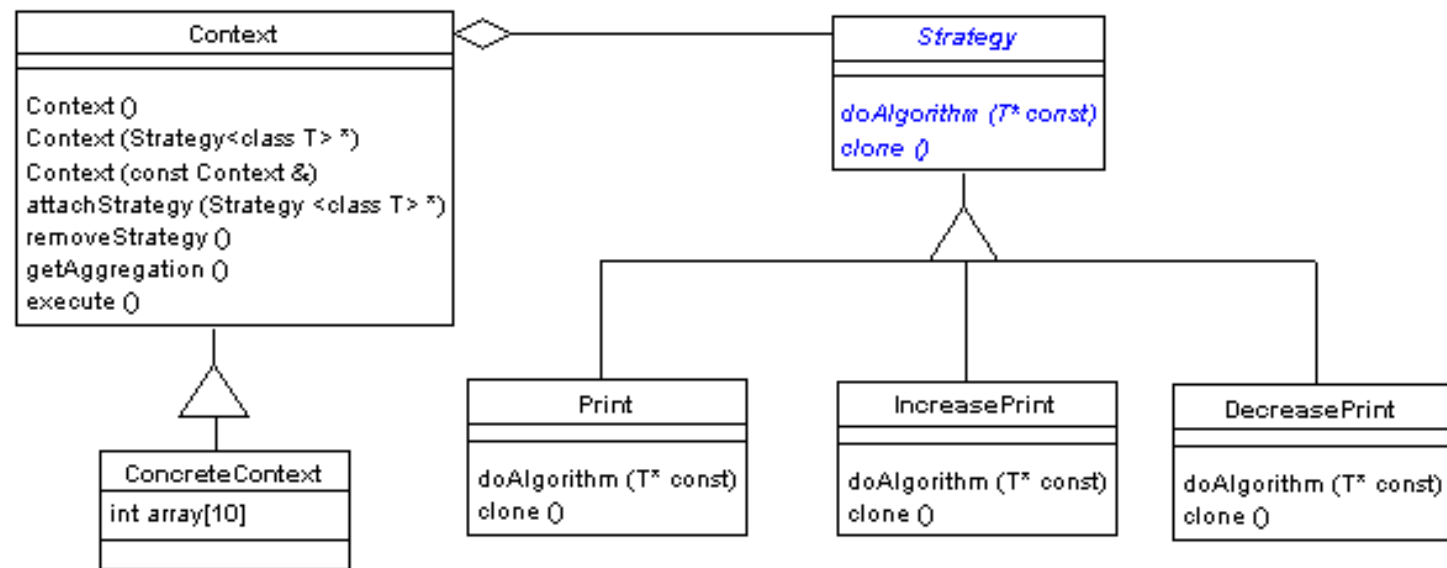
# Strategy II

- *Consequences:*
  - Greater flexibility, reuse
  - Can change algorithms dynamically
  - Strategy creation & communication overhead
  - Inflexible strategy interface
- *Implementation:*
  - Exchanging information between a strategy and its context
  - Static strategy selection via templates
- *Also Known As:* Policy
- *Known uses:*
  - Interviews text formatting
  - IC design - RTL register allocation & scheduling strategies
  - ET++ SwapsManager calculation engines

# Strategy - Structure

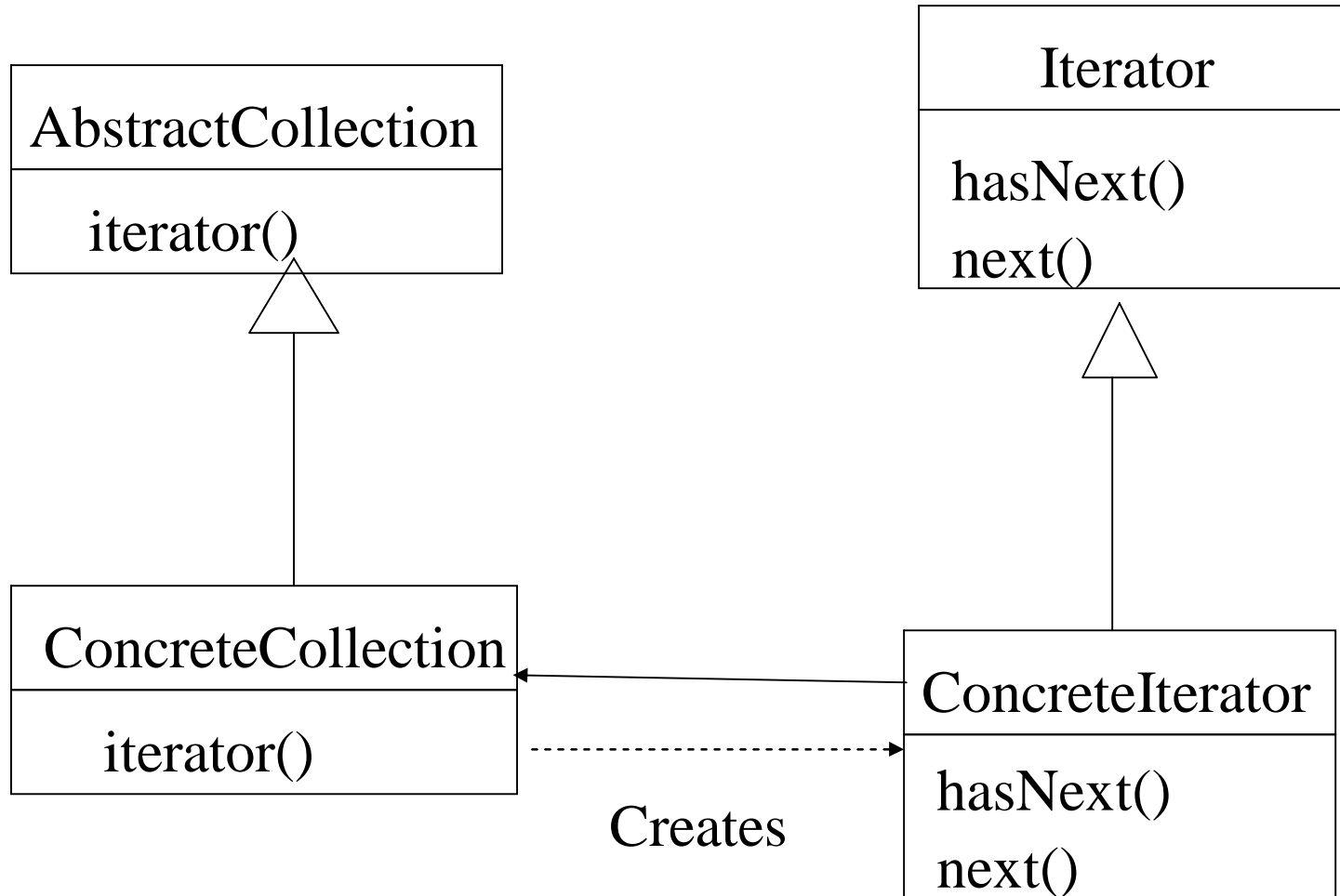


# Strategy - Example

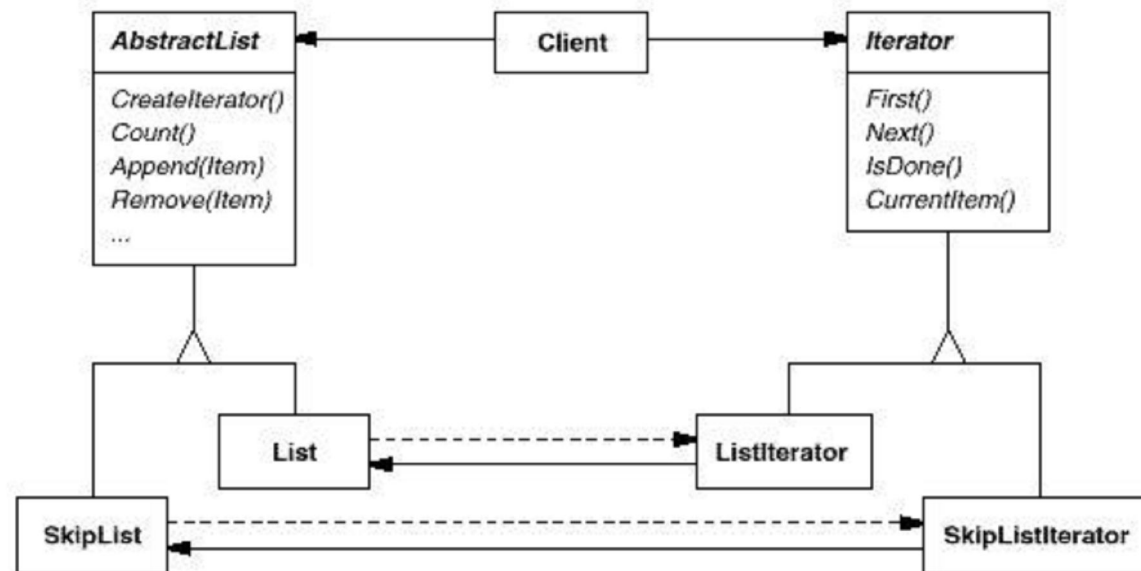


# Iterator (Behavioural)

- *Intent:* Provide a way to access the elements of a collection sequentially
- *Applicability:*
  - To access the contents of a collection without exposing its internal representation
  - To support multiple traversals of collection
  - To provide a uniform interface for traversing different collections (i.e. to support polymorphic iteration)
- *Also Known As:* Cursor
- *Know Uses:* Widespread in access to data structures; Directly supported in languages such as Java and C++ STL



# List Example





# Template Method (Behavioural)

- *Intent:* Define the skeleton of an algorithm in a method, deferring some steps to subclasses, thus allowing the subclasses to redefine certain steps of the algorithm.
- *Applicability:*
  - To implement the invariant parts of an algorithm once and leave it up to the subclasses to implement the behaviour that can vary.
  - To factorize and localize the common behaviour among subclasses to avoid code duplication.

# Template Method

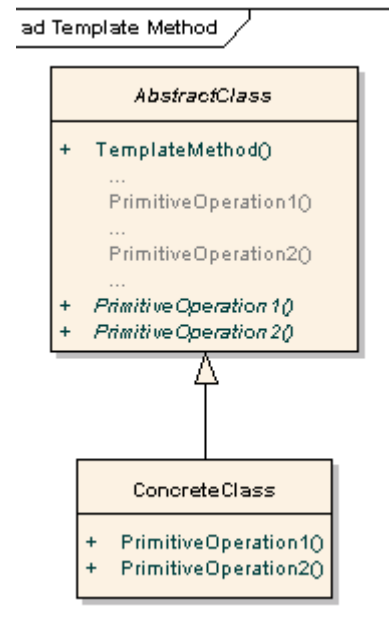
## Generic class

- implements a template method that defines the skeleton of the algorithm
- the template method calls the hook methods
- hook methods are abstract methods

## Concrete class

- implements the hook methods to carry out the subclass specific parts of algorithm

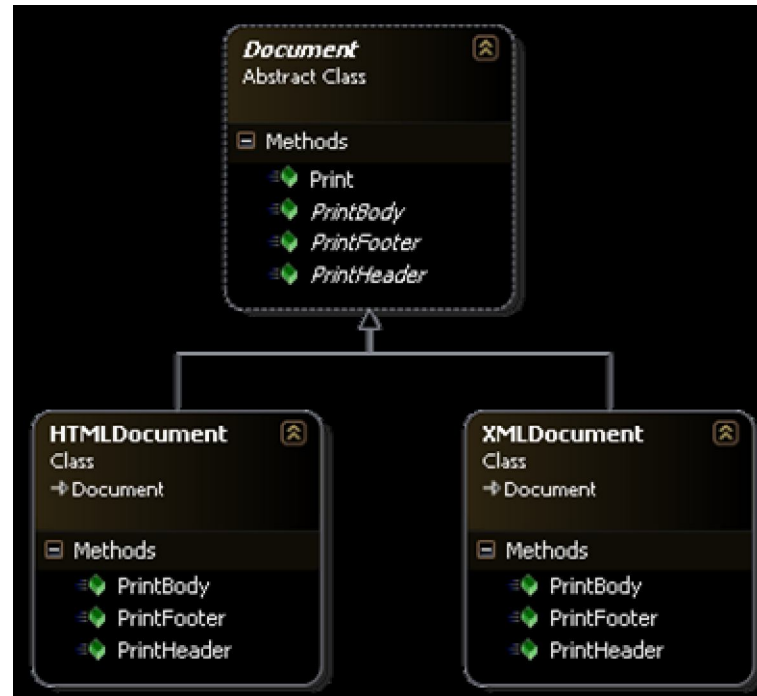
# Template Method



- Diagram ref: Design pattern in simple examples, Shabbir, 2007.

# Template Method Example

- Document Processing Example



# Template Method example - Java

```
abstract public class TemplateMethod {  
  
    public TemplateMethod() {  
    }  
  
    // implementation of processMessage is left to the subclasses.  
  
    public abstract Object processMessage (Object message);  
}
```

# Patterns at Different Levels

- Applicable in most stages of the OO lifecycle
  - Analysis, design, implementation, reviews, documentation, reuse, and refactoring
- Analysis patterns
  - Typical solutions to recurring analysis problems
  - See Analysis Patterns, Fowler; Addison-Wesley, 1996
- Architectural patterns
  - See POSA for example (Pattern-Oriented Software Architecture, Buschmann, et al.; Wiley, 1996)
- Design patterns
  - GoF design patterns (Design Patterns, Gamma, et al.; Addison-Wesley, 1995)
- Programming Idioms
  - Smalltalk Best Practice Patterns, Beck; Prentice Hall, 1997
  - Concurrent Programming in Java, Lea; Addison-Wesley, 1997
  - Advanced C++, Coplien, Addison-Wesley, 1992
  - Effective C++: 50 Specific Ways to Improve Your Programs and Design (2nd Edition), Scott Meyers, Addison-Wesley, 1997
  - More Effective C++: 35 New Ways to Improve Your Programs and Designs, Scott Meyers, Addison-Wesley, 1995