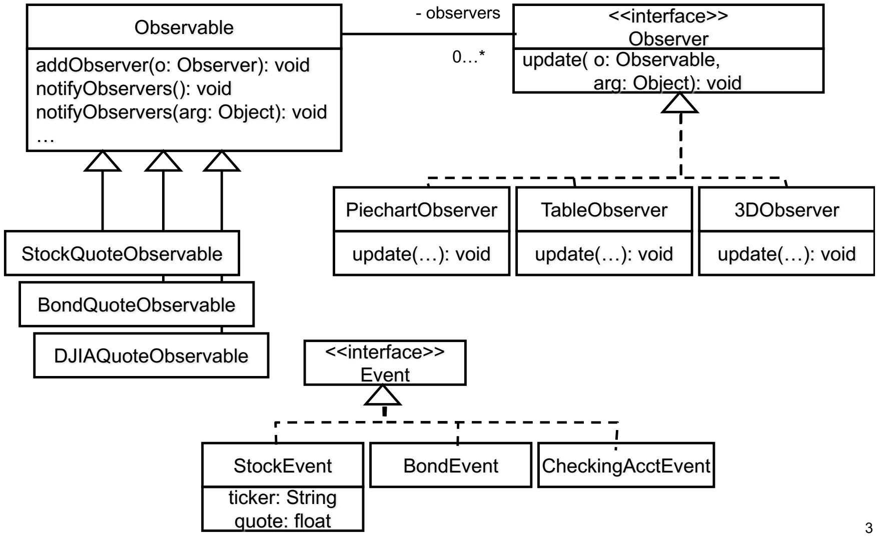
Template Method Design Pattern

Template Method

- Intent
  - Define the template (or abstract flow) of an algorithm (or logic/procedure) in a superclass's method
    - Leave implementation details of some steps to subclasses.
    - Subclasses implements those steps without changing the template/flow that their superclass defines.

Recap: *Observer*


Observable
addObserver(o: Observer): void
notifyObservers(): void
notifyObservers(arg: Object): void
…

- observers
0…*

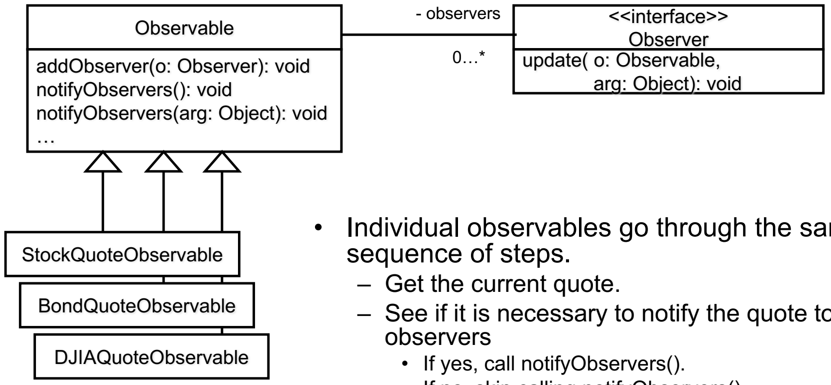
<<interface>>
Observer
update(o: Observable,
arg: Object): void

StockQuoteObservable
BondQuoteObservable
DJIAQuoteObservable

PiechartObserver
update(…): void
TableObserver
update(…): void
3DObserver
update(…): void

<<interface>>
Event

StockEvent
ticker: String
quote: float
BondEvent
CheckingAcctEvent

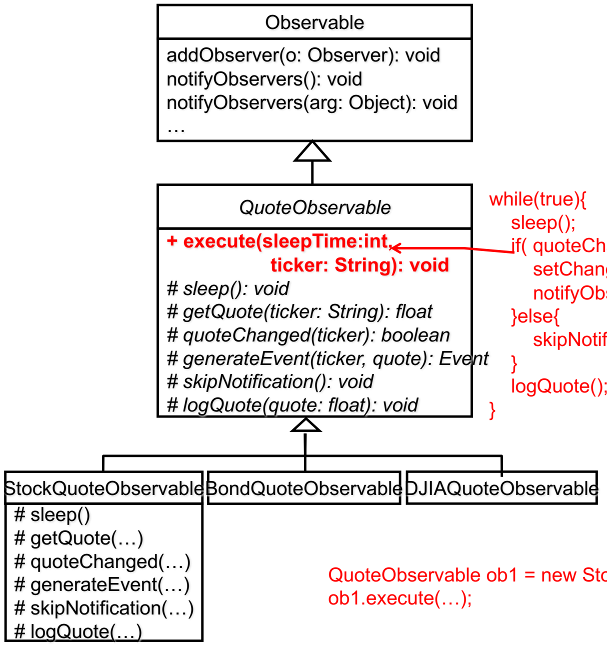

Observable
addObserver(o: Observer): void
notifyObservers(): void
notifyObservers(arg: Object): void
…

- observers
0…*

<<interface>>
Observer
update(o: Observable,
arg: Object): void

StockQuoteObservable
BondQuoteObservable
DJIAQuoteObservable

- Individual observables go through the same sequence of steps.
  - Get the current quote.
  - See if it is necessary to notify the quote to observers
    - If yes, call notifyObservers().
    - If no, skip calling notifyObservers().
  - Log/record the quote.
- However, different observables may want to implement certain steps in different ways.
  - How to get a quote?
  - What to do if no events are notified?
  - How to log/record a quote?
  - How long interval to repeat this sequence?



Observable

addObserver(o: Observer): void
notifyObservers(): void
notifyObservers(arg: Object): void
…

QuoteObservable

+ execute(sleepTime:int, ticker: String): void
# sleep(): void
# getQuote(ticker: String): float
# quoteChanged(ticker): boolean
# generateEvent(ticker, quote): Event
# skipNotification(): void
# logQuote(quote: float): void

StockQuoteObservable

# sleep()
# getQuote(…)
# quoteChanged(…)
# generateEvent(…)
# skipNotification(…)
# logQuote(…)

BondQuoteObservable

DJIAQuoteObservable

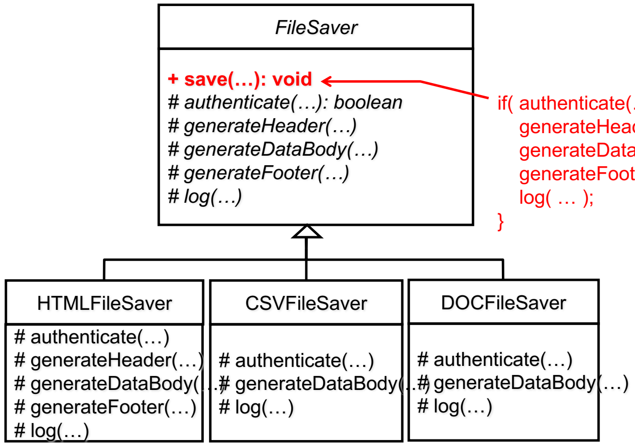
```
while(true){
  sleep();
  if( quoteChanged( getQuote(ticker) ){
    setChanged();
    notifyObservers( generateEvent(…) );
  }else{
    skipNotification();
  }
  logQuote();
}
```

```
QuoteObservable ob1 = new StockQuoteObservable(…);
ob1.execute(…);
```

Points

- Can avoid copying and pasting similar yet slightly different code across different Observable classes.
  - Intend to improve maintainability
- A template method can be a “final” method.

Saving Data to a File



FileSaver

+ save(…): void
# authenticate(…): boolean
# generateHeader(…)
# generateDataBody(…)
# generateFooter(…)
# log(…)

HTMLFileSaver

# authenticate(…)
# generateHeader(…)
# generateDataBody(…)
# generateFooter(…)
# log(…)

CSVFileSaver

# authenticate(…)
# generateDataBody(…)
# log(…)

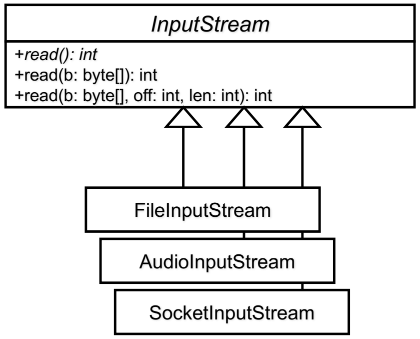
DOCFileSaver

# authenticate(…)
# generateDataBody(…)
# log(…)

```
if( authenticate(…) ){
  generateHeader(…);
  generateDataBody(…);
  generateFooter(…);
  log( … );
}
```

InputStream in Java API

- InputStream
  - Abstract class that represents input byte streams
- InputStream classes
  - AudioInputStream
  - FileInputStream
  - ObjectInputStream
  - PipedInputStream
  - …etc.
  - SocketInputStream (hidden)



InputStream

+read(): int
+read(b: byte[]): int
+read(b: byte[], off: int, len: int): int

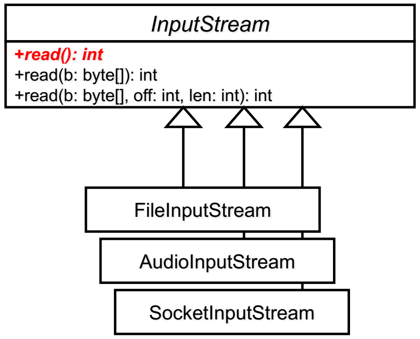
FileInputStream

AudioInputStream

SocketInputStream

InputStream classes

- read()
  - Reads the next byte of data from an input stream. The value byte is returned in the range from 0 to 255.
  - Returns -1 if no byte is available because the end of the stream has been reached.
  - Defined as an abstract method. Each subclass must implement it.



InputStream

+read(): int

+read(b: byte[]): int

+read(b: byte[], off: int, len: int): int

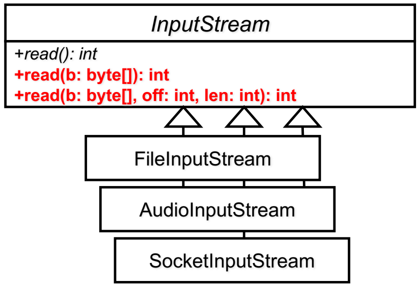
FileInputStream

AudioInputStream

SocketInputStream

InputStream classes

- read()
  - Reads the next byte of data from an input stream. The value byte is returned in the range from 0 to 255.
  - Returns -1 if no byte is available because the end of the stream has been reached.



InputStream

+read(): int

+read(b: byte[]): int

+read(b: byte[], off: int, len: int): int

FileInputStream

AudioInputStream

SocketInputStream

- Template methods
  - read(byte[] b)
    - Reads multiple bytes

    -
      ```
      int i = 0;
      while( i < b.length ){
        read the next byte with read();
        if (EOS is reached) break;
        copy the byte to b[i];
        i++;
      }
      return i;
      ```
    - Equivalent to call read(b, 0, b.length)
  - read(byte[] b, int off, int len)
    - Reads multiple bytes

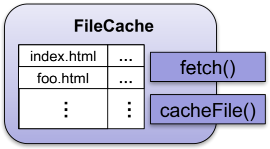
    -
      ```
      int i = 0;
      while( i < len ){
        if (EOS is reached) break;
        read the next byte with read();
        copy the byte to b[off+i];
        i++;
      }
      return i;
      ```

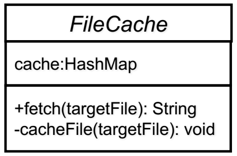
Template Method* and *Factory Method

- *Factory method* is a variant of *Template Method*
  - *Factory method*
    - Specializes in defining a template for instance creation/initialization
  - *Template method*
    - Can be used to define a template for any logic.

File Caching

- **`FileCache`**
  - Maintains a map that pairs a relative file path and string data of the file.
    - Assume **`java.util.HashMap`**
  - **`fetch()`**
    - accepts a file path and gets the content of the requested file from the HashMap.
  - **`cacheFile()`**
    - accepts a file path and its content to the HashMap.





FileCache

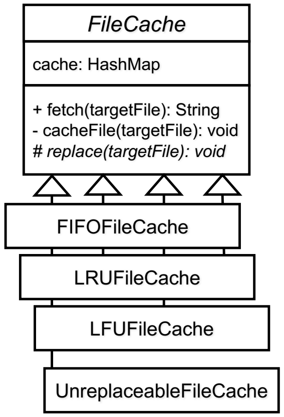
cache:HashMap

+fetch(targetFile): String

-cacheFile(targetFile): void

- **FileCache**
  - **public String fetch(targetFile)**
    - targetFile: a relative file path
      - String or java.nio.Path
    - Returns a requested file’s content
    - Template method

    -
      ```
      if ( targetFile is cached ){
          return targetFile’s (cached) content.
      } else if {
          read targetFile from the disk.
          if( cache is not full )
              cacheFile(targetFile);
          else if
             replace( targetFile ); }
      ```



FileCache

cache: HashMap

+ fetch(targetFile): String
- cacheFile(targetFile): void
# *replace(targetFile): void*

FIFOFileCache

LRUFileCache

LFUFileCache

UnreplaceableFileCache

HW 12

- Implement **FileCache** and its two subclasses with *Template Method*
  - **FIFOFileCache** and **UnreplaceableFileCache**
    - **UnreplaceableFileCache**: no cache replacement in replace()
      - c.f. **NullReplacement** in HW 7
      - c.f. Lecture notes #10 and #11 (*Strategy*)