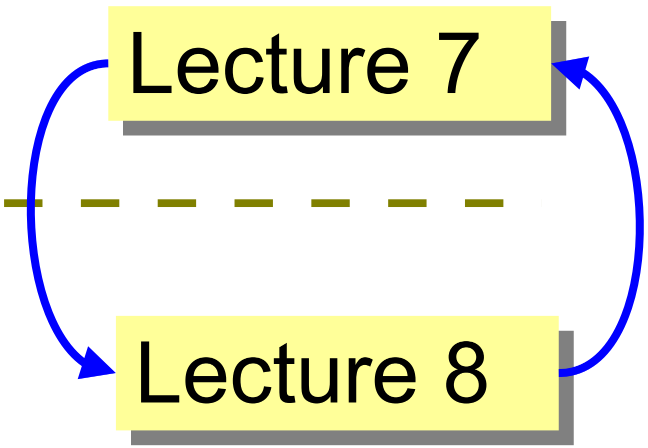
Software Engineering

Lecture 7-8:
Design evaluation &
Implementation

Outline

- Design evaluation
  - design review
  - criteria
  - walk through (informal)



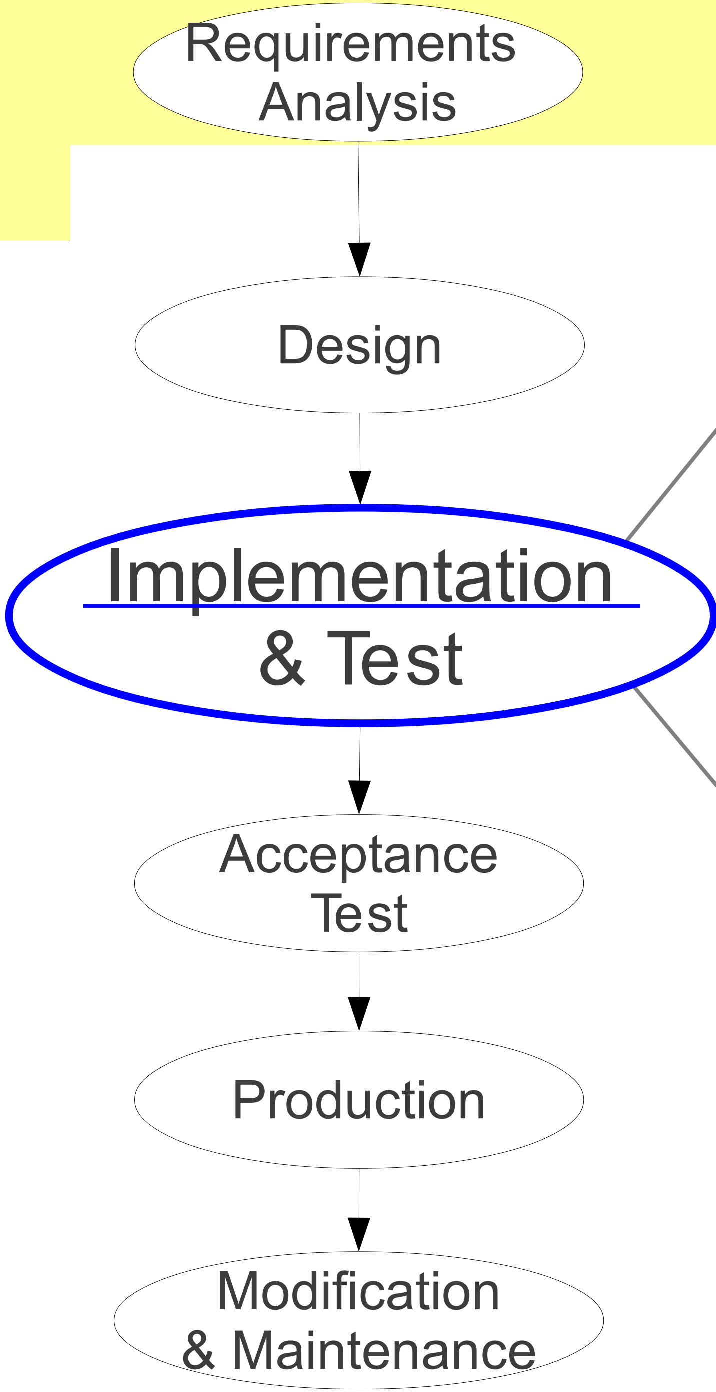
- Implementation:
  - plan: top down, bottom up, hybrid
  - code

Case study: KEngine implementation

References

- Liskov & Guttag (2001):
  - Chapter 14: Implementation

Development process



- Evaluate design
- Plan implementation
- Code the implementation
- Test the code
- Output:
  - (tested) prototypes
  - test drivers

Design evaluation

- Goal: determine whether the design is *adequate*
- Participants:
  - analysts
  - designers
  - developers

Design review

- A technique used to check design specifications against a set of criteria
- Tasks:
  - verify using symbolic test data
  - specify performance constraints when necessary
  - discuss support for modifications
- Carried out in two phases:
  - review each component
  - review how the components work together

Design criteria

- Correctness
- Performance
- Modifiability
- Modularity

Correctness

- Correct w.r.t requirement specification
- Common technique is design review
- Check:
  - sequence diagram against functional requirement
  - class diagram against data requirement
- Check design specification:
  - consistency and completeness
  - rep reflects associations with others
  - operations usage in sequence diagram(s)

Design walk through

- A design review technique
- Walk through the design using *symbolic* test data
- Two basic steps:
  - identify symbolic test cases
  - “run” the test cases (on paper) through each sequence diagram

Example: KEngine

Engine
Engine() queryfirst(String): Query queryMore(String): Query findDoc(String): Doc addDocs(String): Query

Symbolic test cases

- Doc d1 (title = t1):
  - <w1,<d1,6>>, <w2, <d1,12>>
- Doc d2 (title = t2):
  - <w1,<d2,10>>
- Doc d3 (title = t3):
  - <w2,<d3,4>>

Engine()

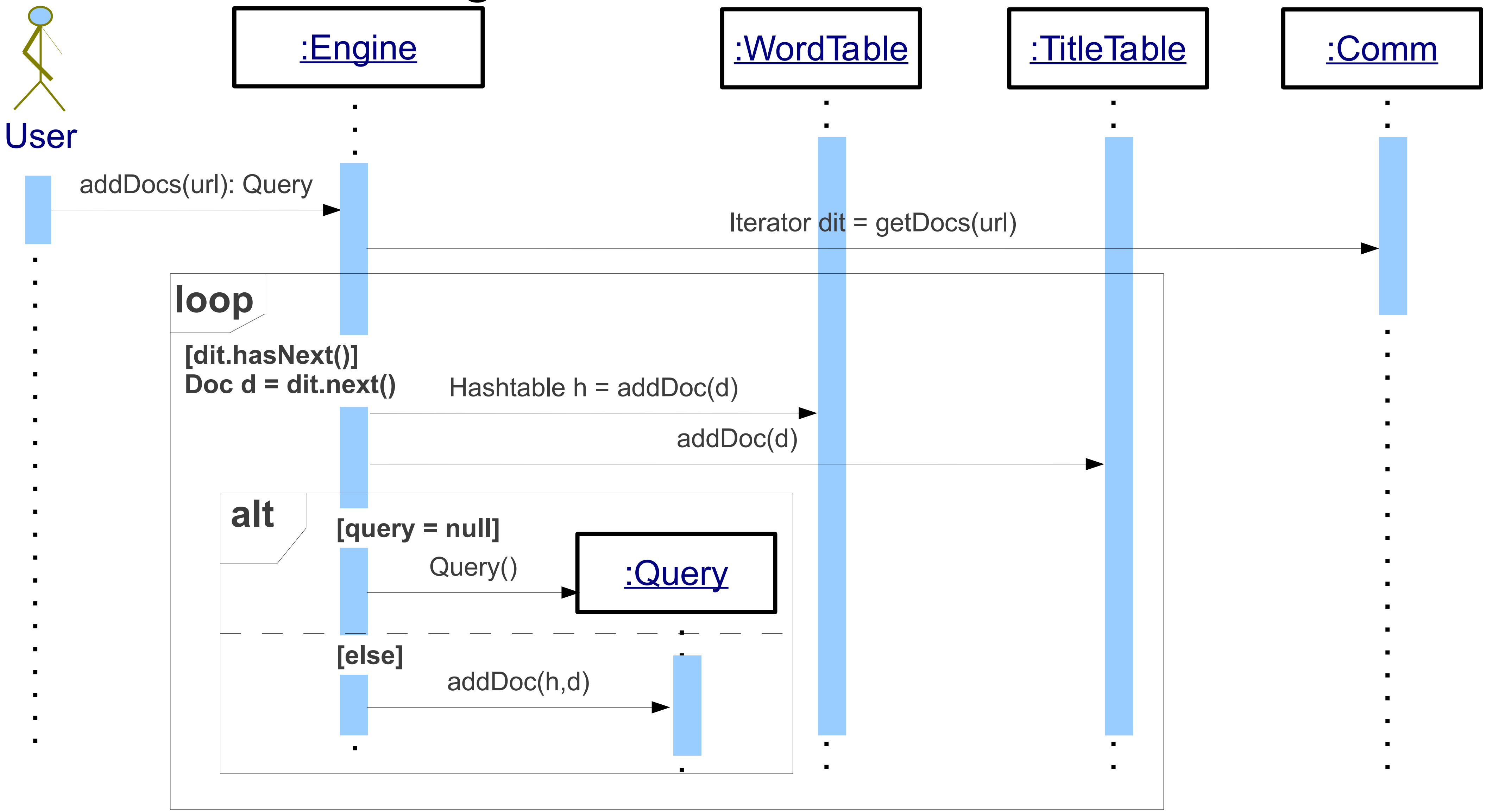
- Check the exceptional case:
  - failure to read the word file → throws exception

addDocs

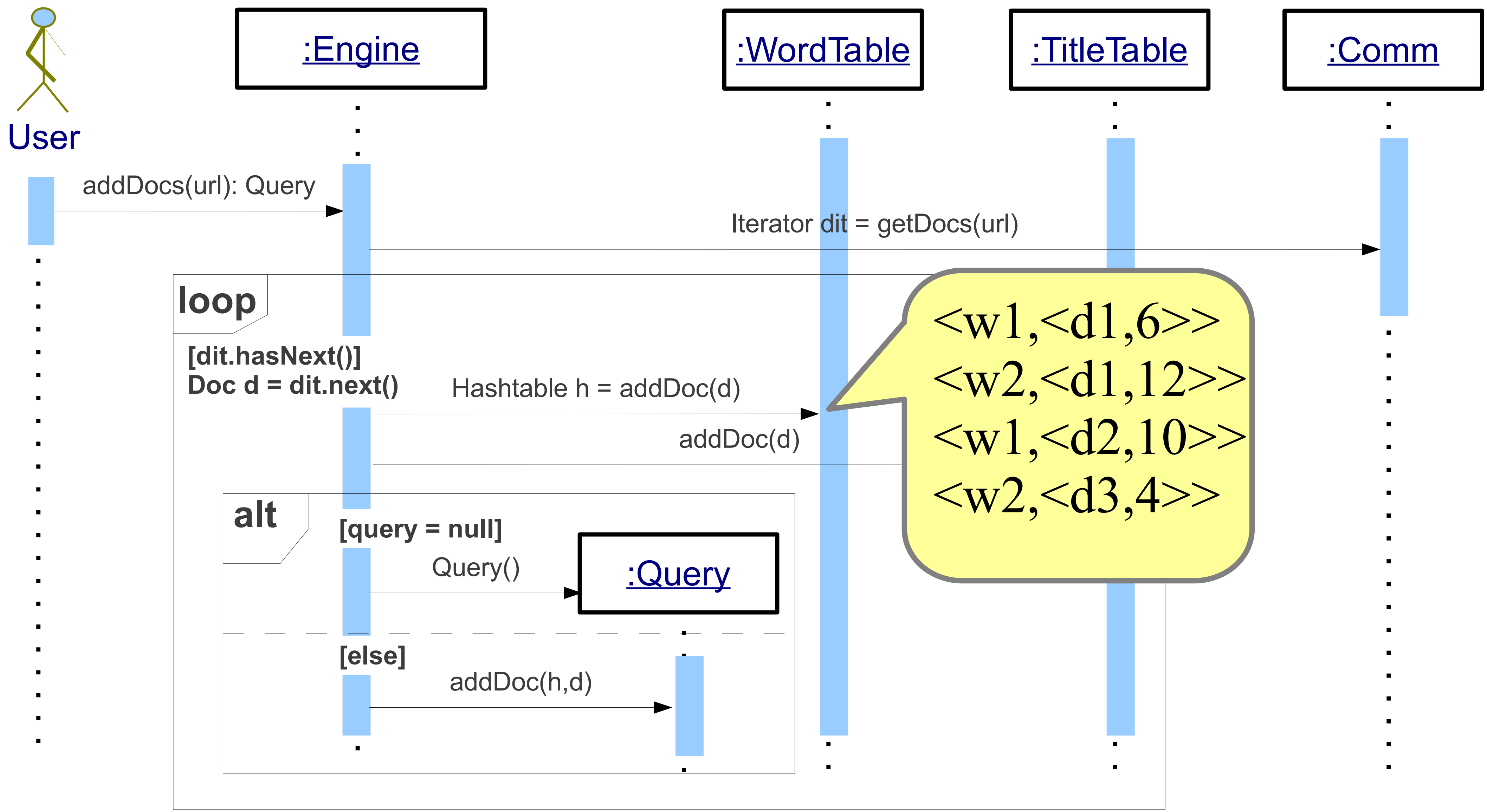
- Create 3 Doc objects: d1, d2, d3
- Doc objects are added to TitleTable:
  - <t1,d1>, <t2,d2>, <t3,d3>
- Doc objects are added to WordTable:
  - <w1,<d1,6>>, <w2,<d1,12>>
  - <w1,<d2,10>>
  - <w2,<d3,4>>

sd.addDocs (1)

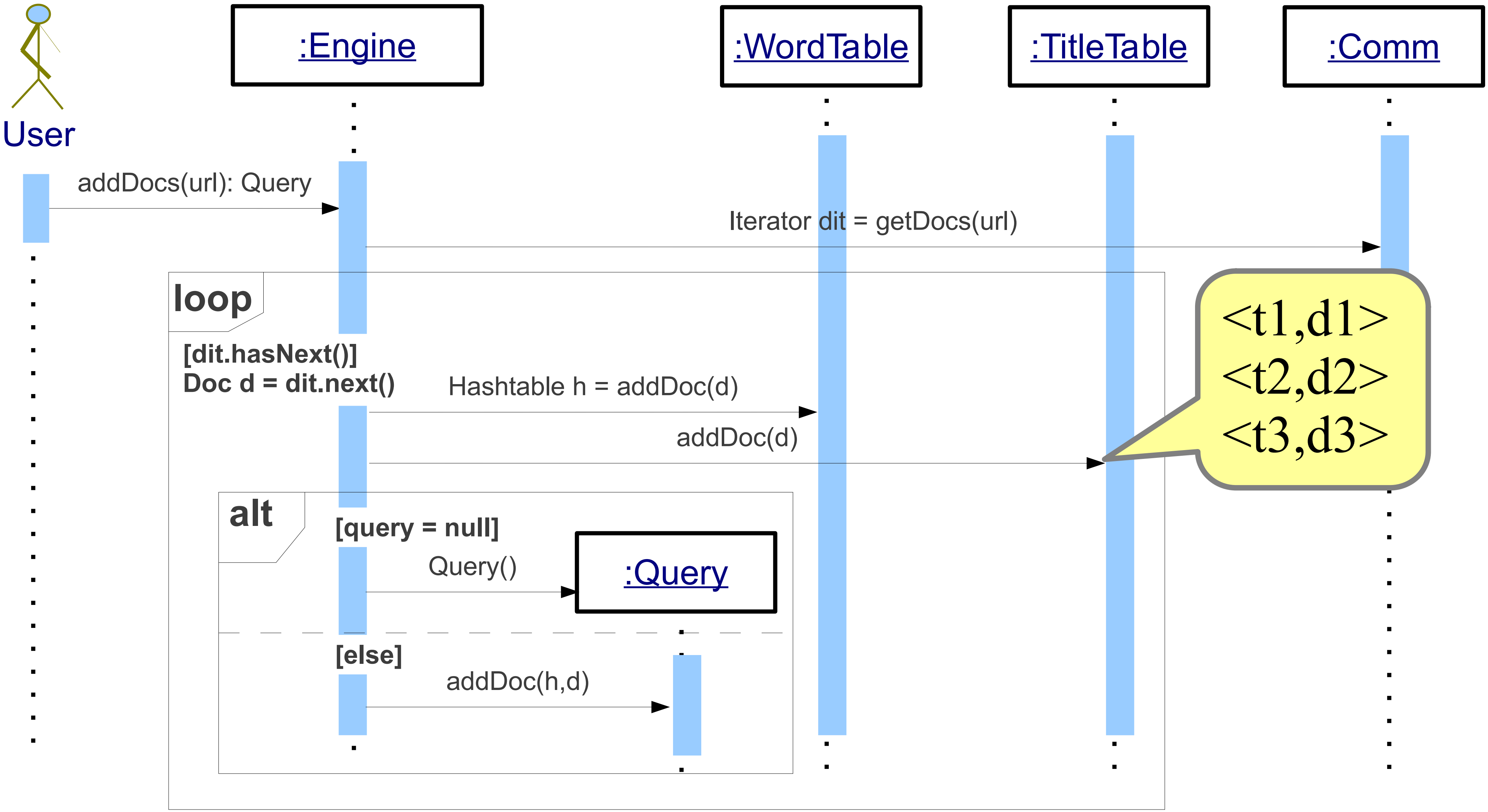
- Verified using sd.addDocs



sd.addDocs (2)

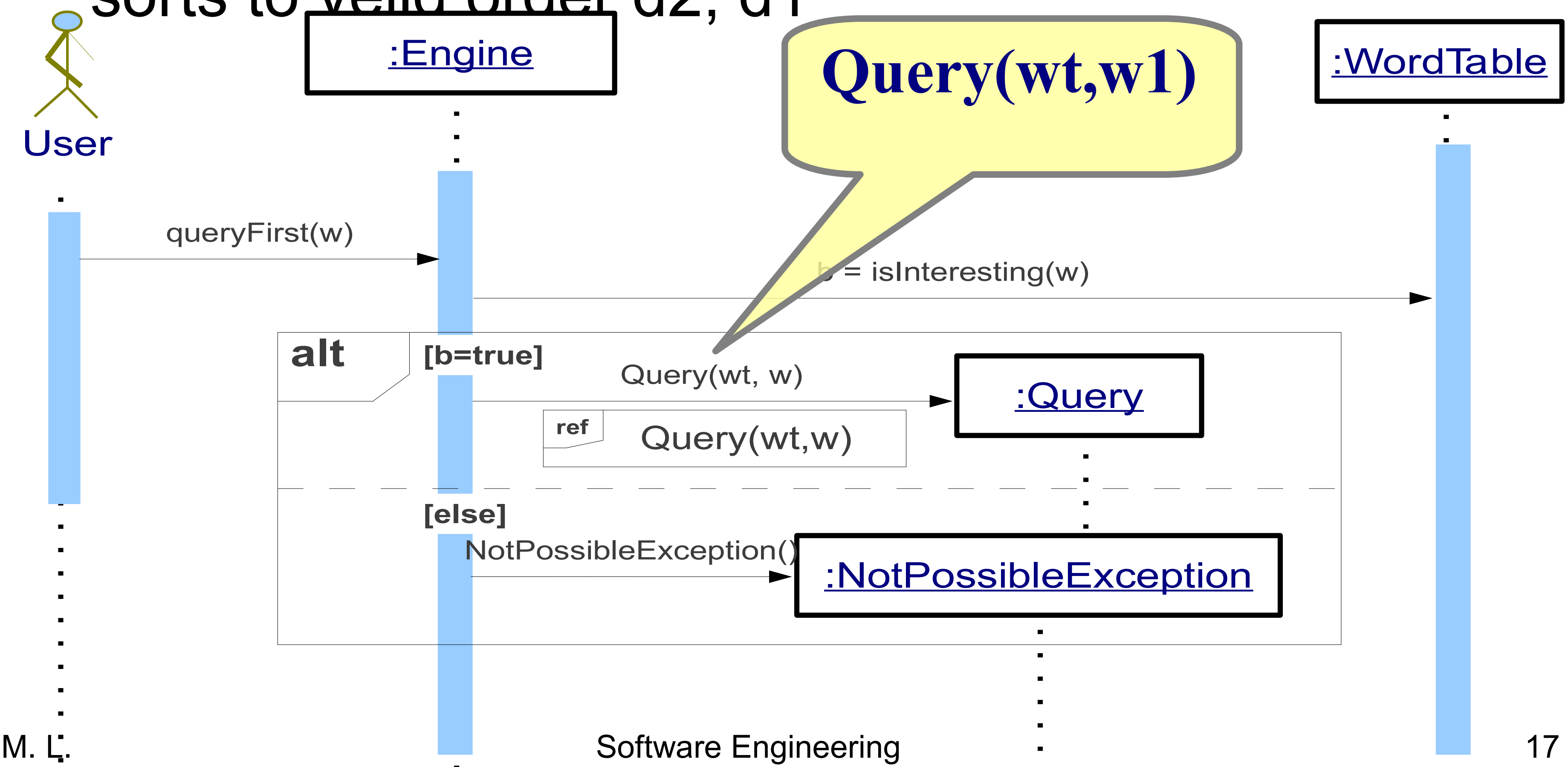

User
:Engine
:WordTable
:TitleTable
:Comm
addDocs(url): Query
Iterator dit = getDocs(url)
loop
[dit.hasNext()]
Doc d = dit.next()
Hashtable h = addDoc(d)
addDoc(d)
alt
[query = null]
Query()
:Query
[else]
addDoc(h,d)
<w1,<d1,6>>
<w2,<d1,12>>
<w1,<d2,10>>
<w2,<d3,4>>

sd.addDocs (3)

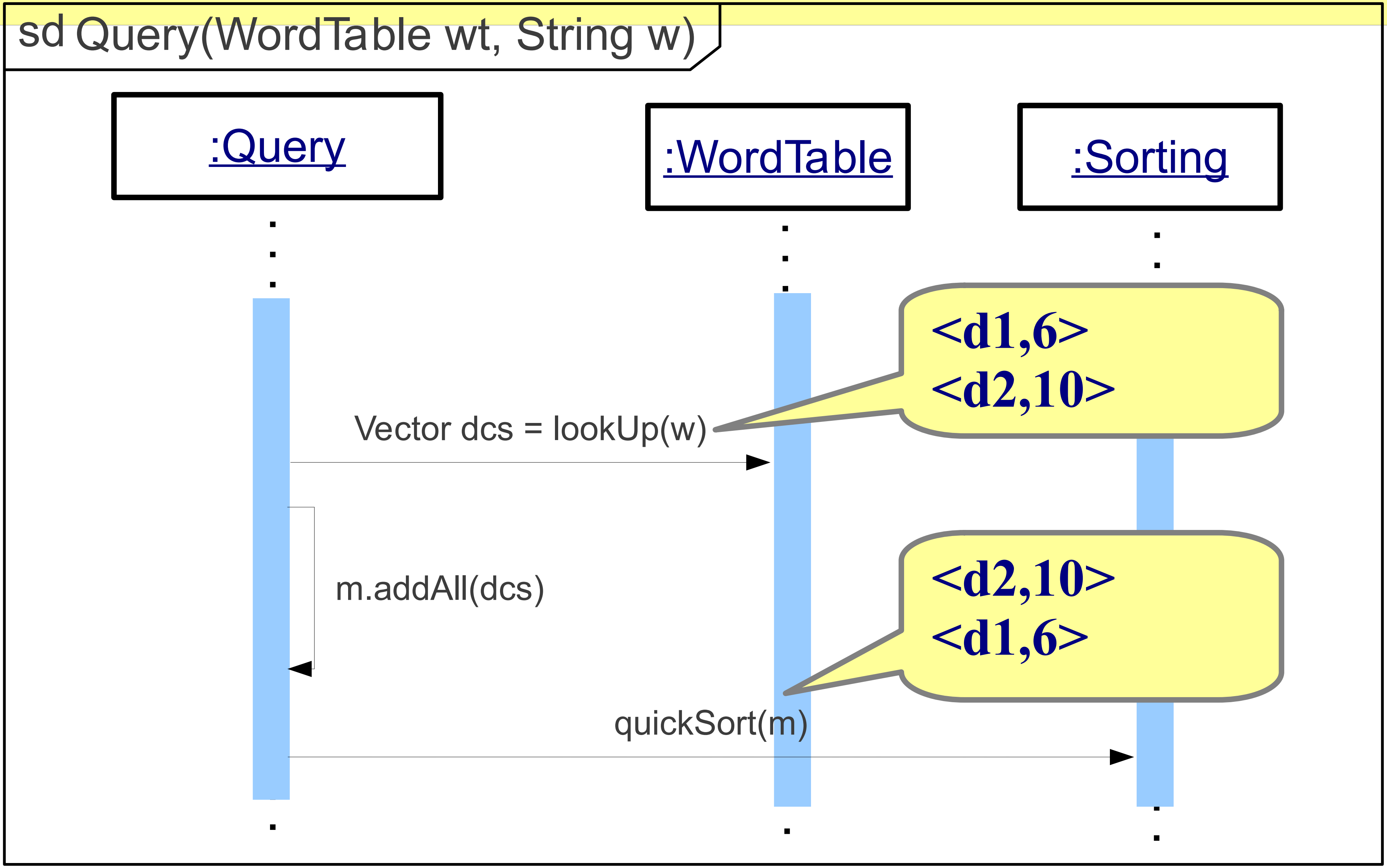

User
:Engine
:WordTable
:TitleTable
:Comm
addDocs(url): Query
Iterator dit = getDocs(url)
loop
[dit.hasNext()]
Doc d = dit.next()
Hashtable h = addDoc(d)
addDoc(d)
<t1,d1>
<t2,d2>
<t3,d3>
alt
[query = null]
Query()
:Query
[else]
addDoc(h,d)

queryFirst(w1) (1)

- Query(wt, w1) is invoked
  - looks up w1 in wt and finds <d1,6> and <d2,10>
  - sorts to veild order d2, d1

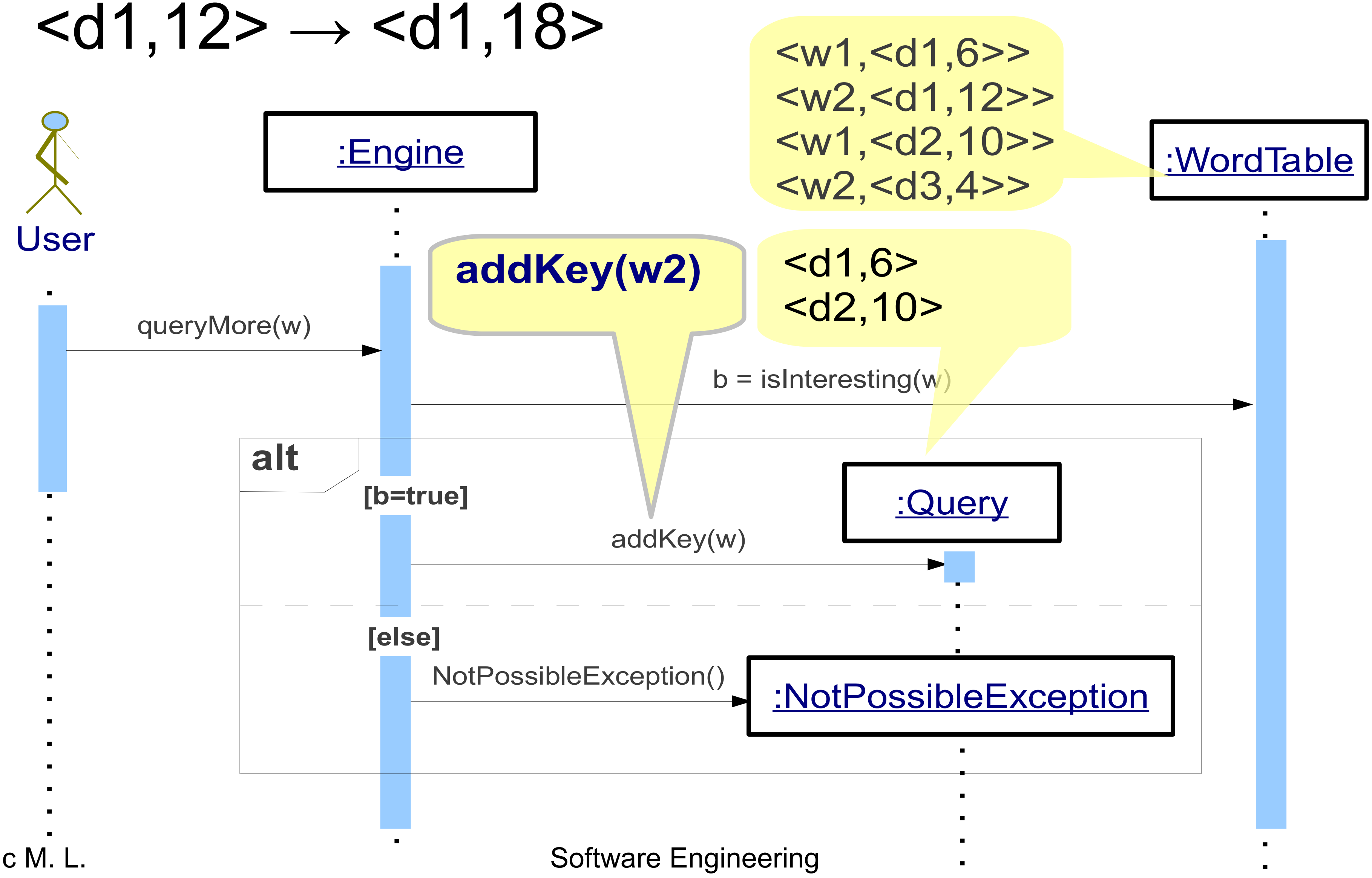


queryFirst(w1) (2)

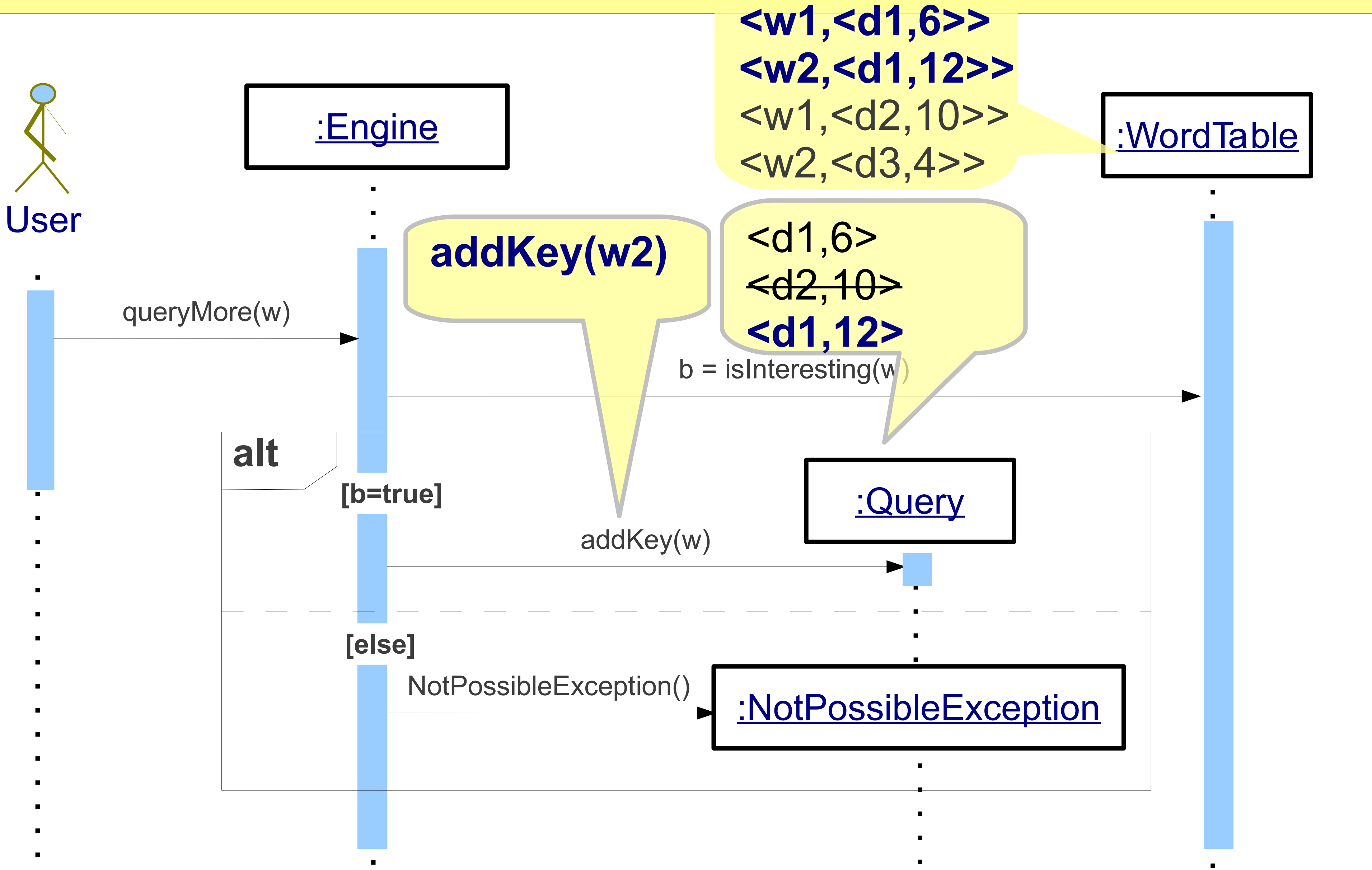


queryMore(w2) (1)

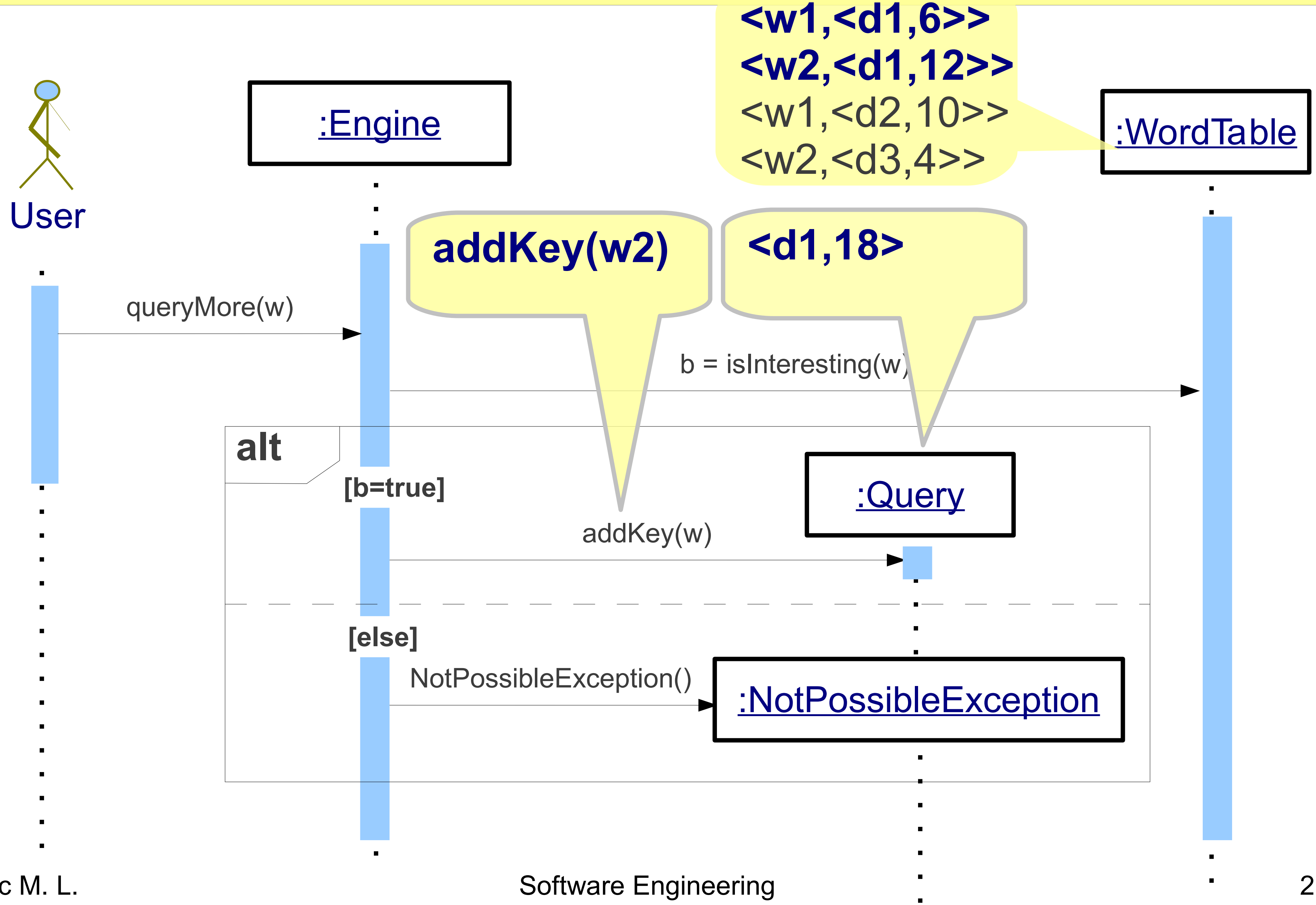
- update the current matches to yeild <d1,6> & <d1,12> → <d1,18>



queryMore(w2) (2)

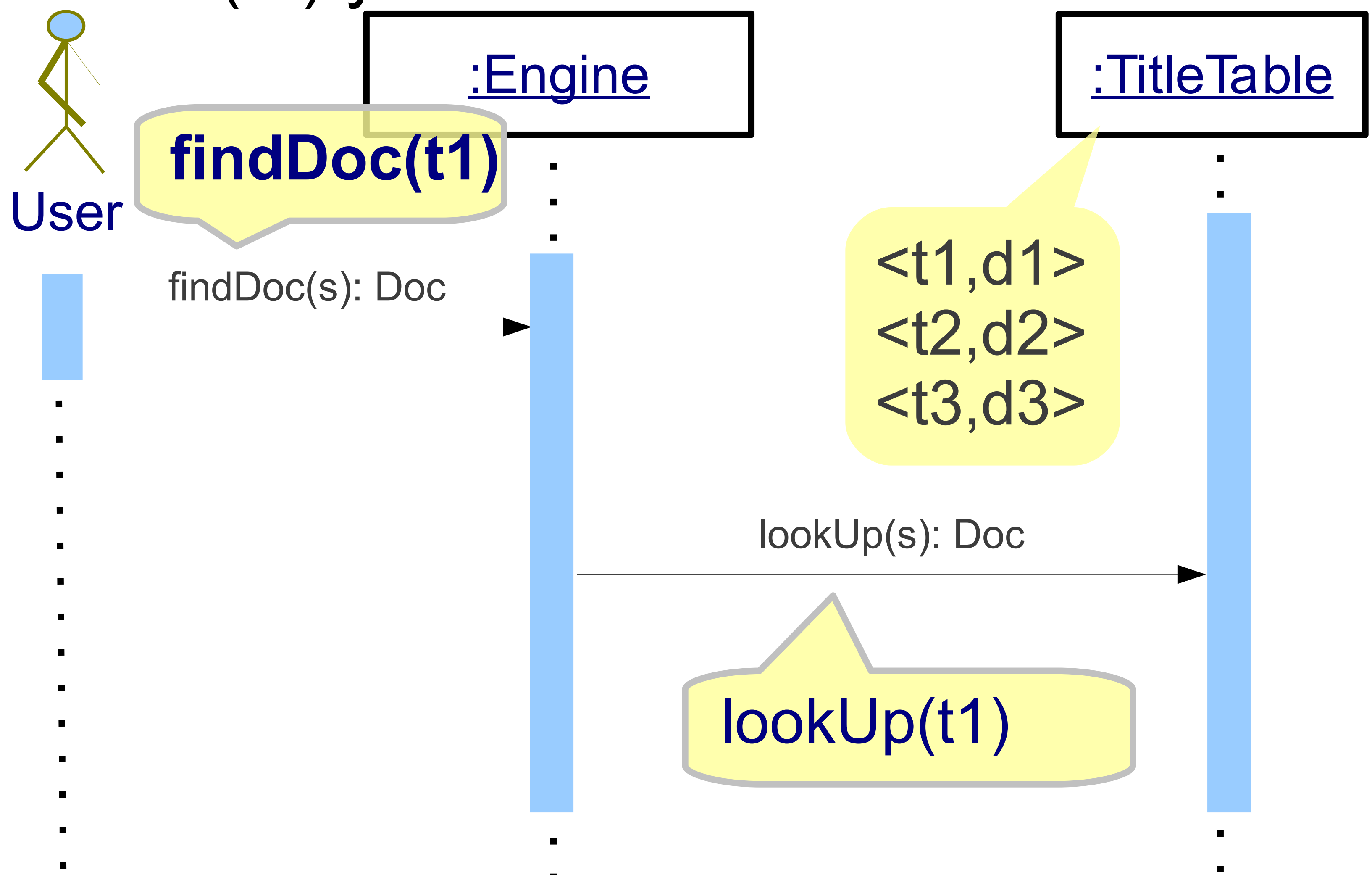


queryMore(w2) (3)

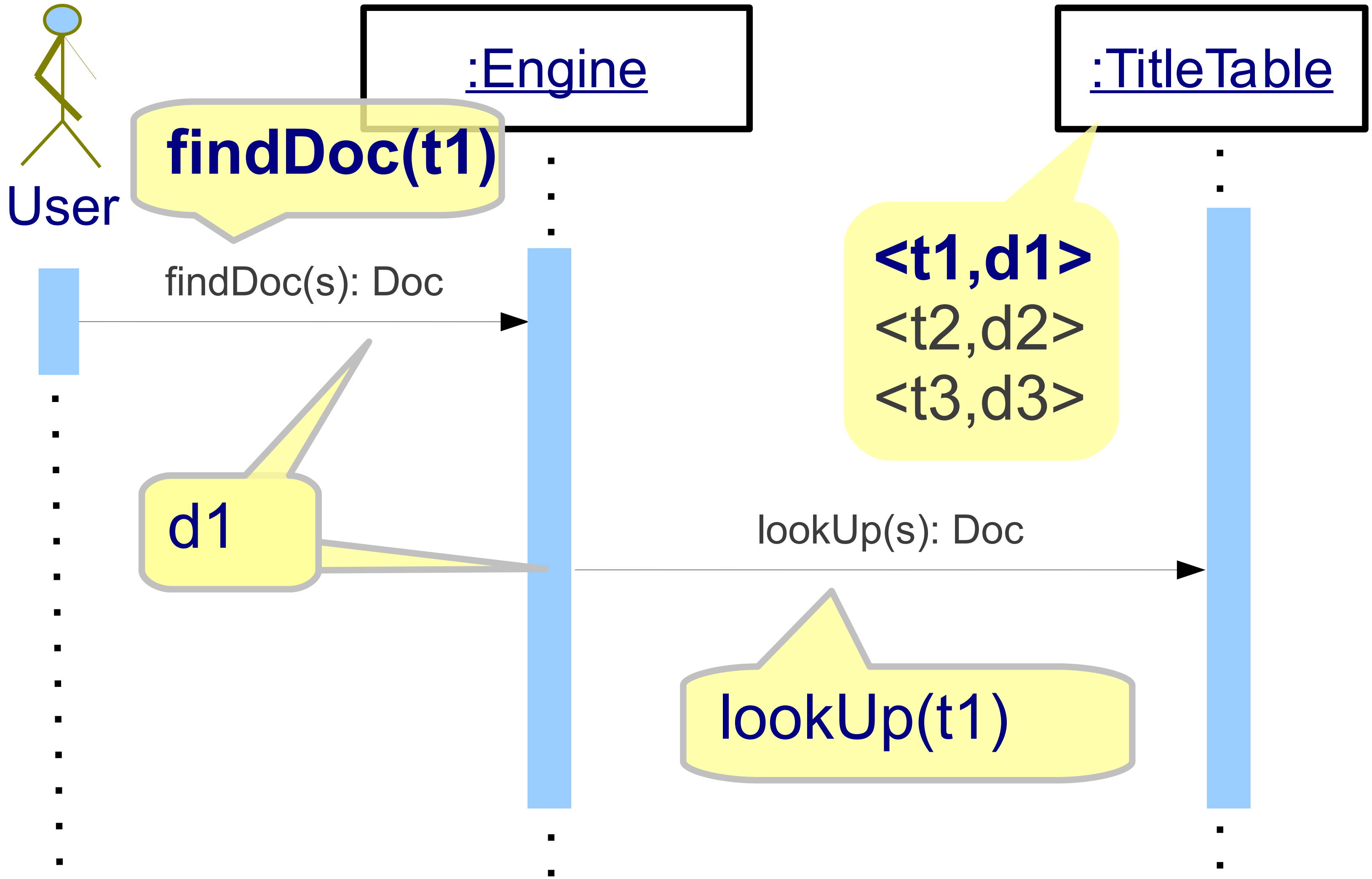


findDoc (1)

- findDoc(t1) yeilds d1



findDoc (2)


User
:Engine
:TitleTable
findDoc(t1)
findDoc(s): Doc
<t1,d1>
<t2,d2>
<t3,d3>
d1
lookUp(s): Doc
lookUp(t1)

Performance

- Translate performance requirements (if any) into performance constraints in spec
- Performance constraint: an expression over input size(s) or an upper bound
- For example: `sort(int[] a)` constraints:
  - worst case *time* = n*log(n) (n = a.length)
  - max *storage* allocated is a small constant

Modifiability

- The extent to which the design is changed to accommodate a modification
- Crude measure of impact:
  - number of abstractions affected
- Best design(s) have impact measure = 1

Example: KEngine

- Modification:
  - store only document URLs not the documents themselves
- Impact measure = 5

Change impact 1

- Doc:
  - replaces body by an URL attribute

Change impact 2

- FullDoc: a new abstraction *containing* the original Doc:
  - document body
  - words(): Iterator
  - getDoc(): Doc

Change impact 3

- Engine.addDocs:
  - uses FullDoc to parse and process documents
  - pass FullDocs (objects) to WordTable.addDoc and TitleTable.addDoc:
    - store Docs and not FullDocs into the tables
    - discard FullDocs afterwards

Change impact 4 & 5

- WordTable.addDoc
  - change parameter type to FullDoc
  - processes FullDoc but maps keywords to Doc
- TitleTable.addDoc
  - change parameter type to FullDoc
  - processes FullDoc but maps titles to Doc

Modularity

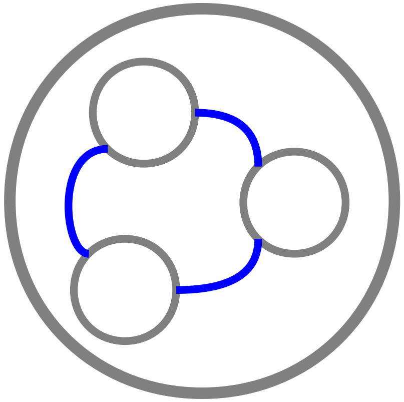
- Modular design eases implementation and maintenance
- Two modularity criteria:
  - cohesion
  - coupling

Cohesion

- Extent to which parts of an abstraction perform the same task
- Applies to both procedural abstraction and data abstraction

Cohesion of operation

- Performs a single task w.r.t arguments
  - determined based on the specification
- Two types:
  - conjunctive
  - disjunctive



Conjunctive coherence

- Performs a combination of logically related tasks

  `A && B && C && ...`

- Example: `WordTable.isInteresting():`
  - conjunctive: two related checks for non-word and uninteresting
  - non-conjunctive if also canonicalise

Disjunctive coherence

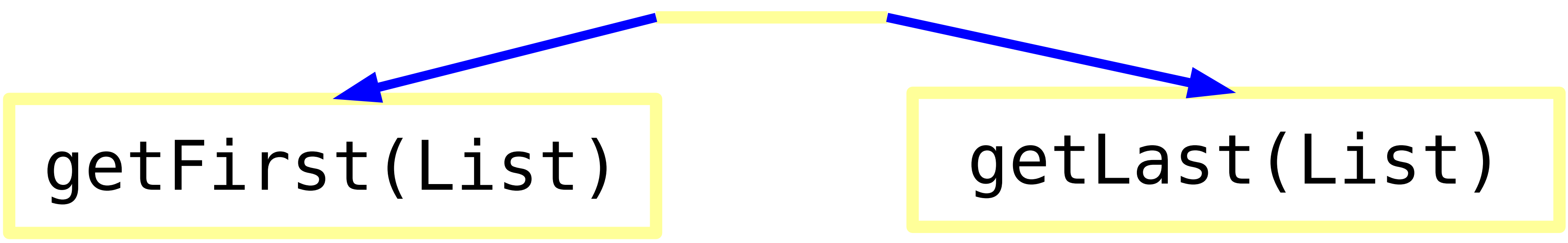
- @effects clause states a disjunction of tasks

  `A || B || ...`

- Avoid if:
  - operation is not a proper generalisation of the tasks
  - tasks can be separated

Bad disjunctive coherence

```java
/**
 * @requires <tt>0 < j < 3 and all
 *   elements of a are Integers</tt>
 * @effects  <pre>
 *   If j = 1
 *     returns first element of a
 *   else
 *     returns the last element of a </pre>
 */
public static int getEnd(List a, int j)
```

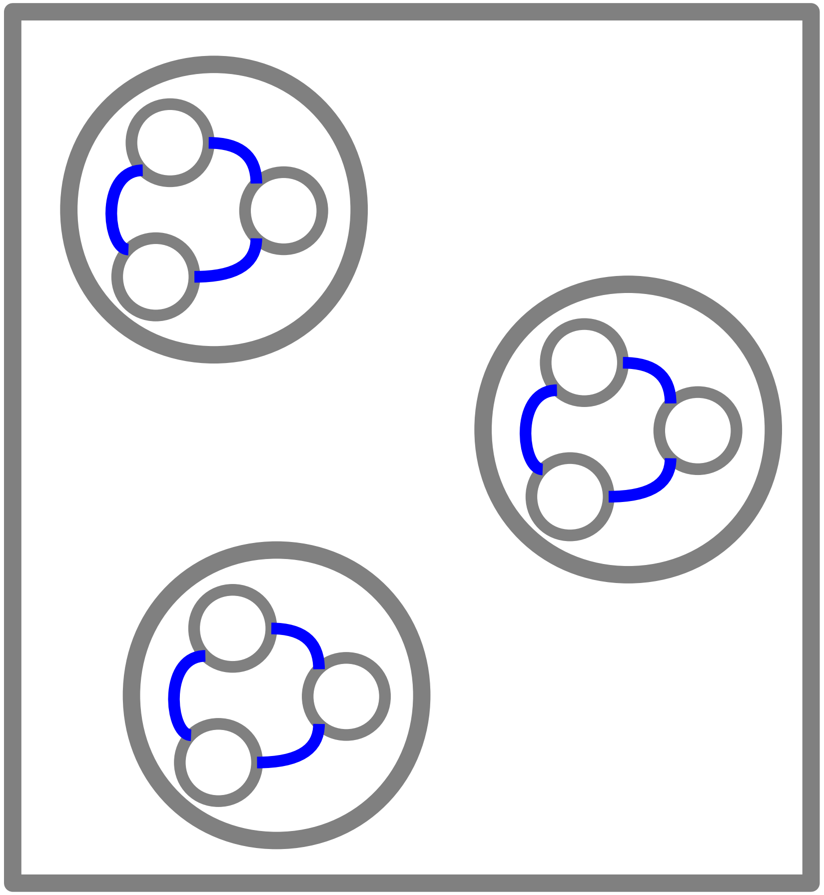


Acceptable disjunctive coherence

```
/**
 * @requires <tt>a is not null && a is not
 *        empty && all elements of a are
 *        integers && 0 <= j < a.size </tt>
 * @effects <pre>
 *   if a is a List
 *    returns element a[j]
 *   else
 *    returns the jth element in
 *    a.iterator</pre>
 */
public static int getElement(Collection a,
int j)
```

Cohesion of type

- Each operation is coherent
- All operations belong to the type
  - adequate for common use

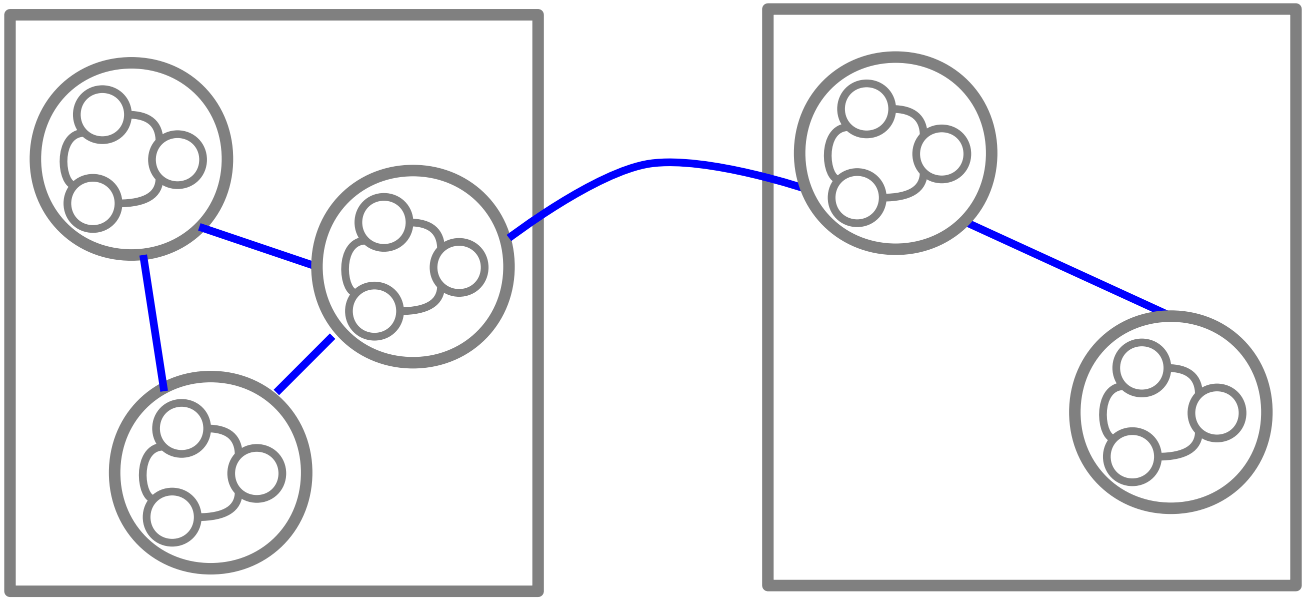


Example

- `Stack.sqrtTop()`:
  - returns the square root of the top element
- Should this be defined in `Stack`?
  - no, because `Stack` is adequate without it

Coupling

- Extent to which related operations/types exchange information
- Applies to both PA and DA



Coupling of operations

- Operations with a “narrow” interface
- Narrow interface:
  - consisted of input parameters (no global variables)
  - use data abstractions as parameter types where applicable
  - use “narrow” data abstractions where possible

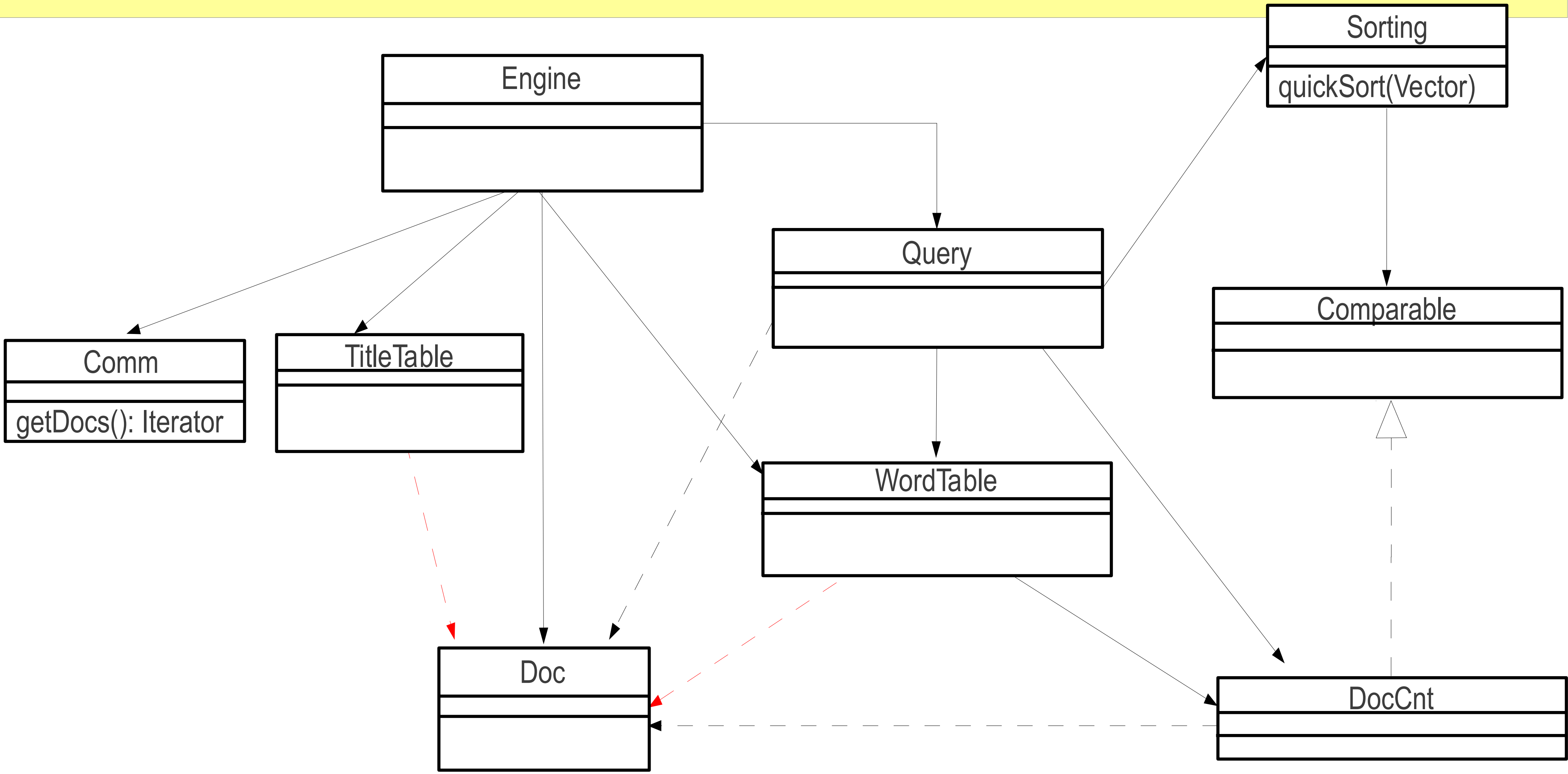
Coupling of data abstractions

- Degree of dependency: strong or weak
- To minimise dependencies:
  - use well-defined design solutions (design patterns)
  - convert strong dependency into weak
- Example: strong to weak conversion
  - `WordTable.addDoc(Doc)` →

    `addDoc(Iterator, Doc)`

    move invocation of `Doc.words()` *out of* `WordTable` *(dependent) and pass the result (an* `Iterator` *object) in as input*

Updated design diagram



Implementation

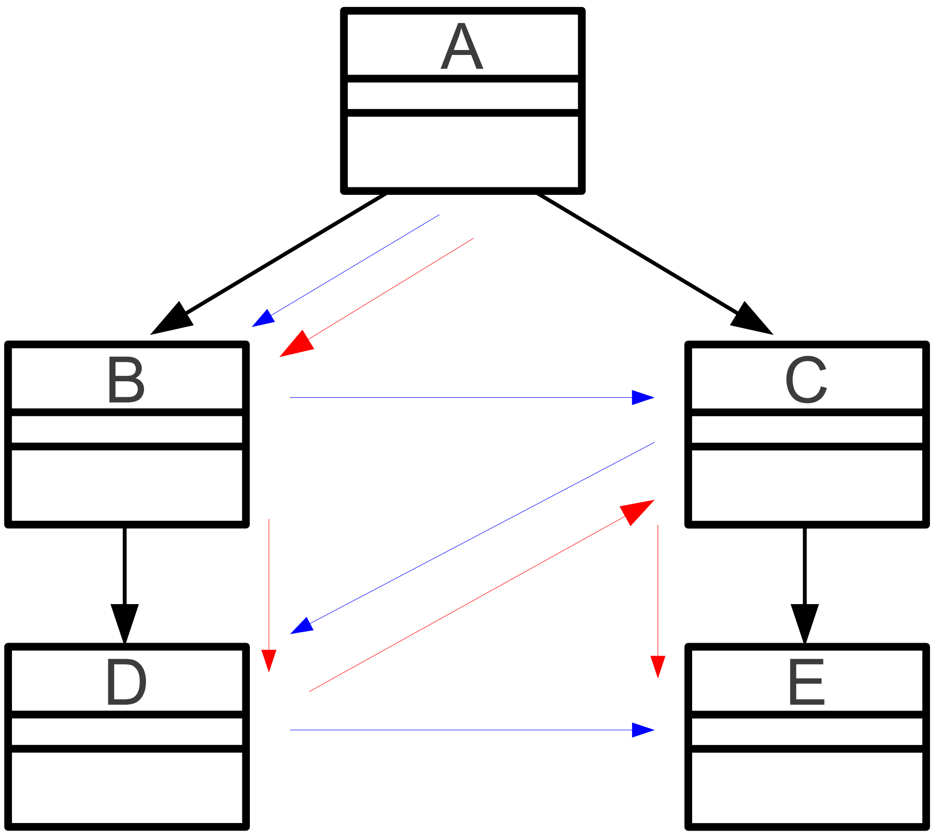
- Implementation & test =
  - transform design into code &
  - test code
- Implementation in small iterations, in tandem with design's
- Three methods for structuring implementation iterations:
  - top-down
  - bottom-up
  - hybrid

Top-down implementation

- Implement a module before those that it uses
- Applicable if implementation is performed after design
  - required for type hierarchy
- Features:
  - integrate one component at a time
  - use *stubs* for lower level components
- A ***stub*** is a fully specified program unit that has a dummy implementation:
  - empty body or body that returns a default value

Top down example

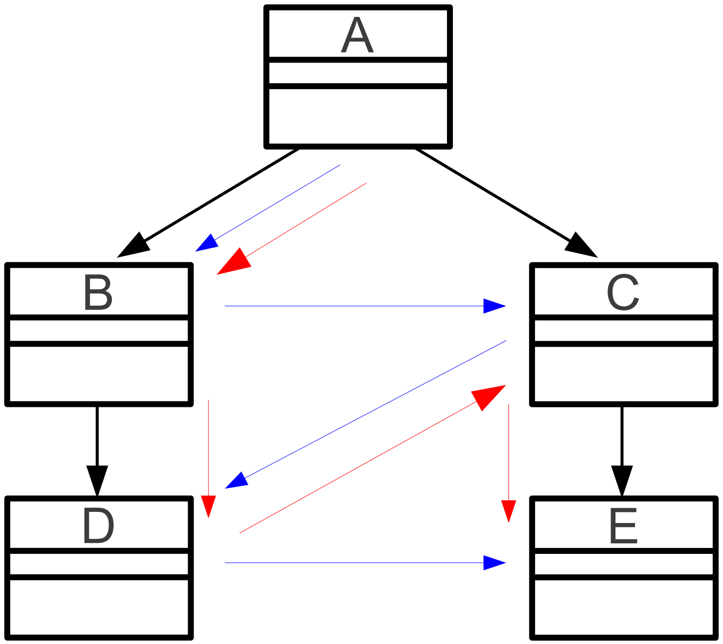
- A, B, C, D, E
- A, B, D, C, E



Class diagram

Pros & Cons

- Advantages
  - early detection of design errors
  - eases testing: test driver re-use
  - early prototypes of the system
- Disadvantages
  - more resource up front

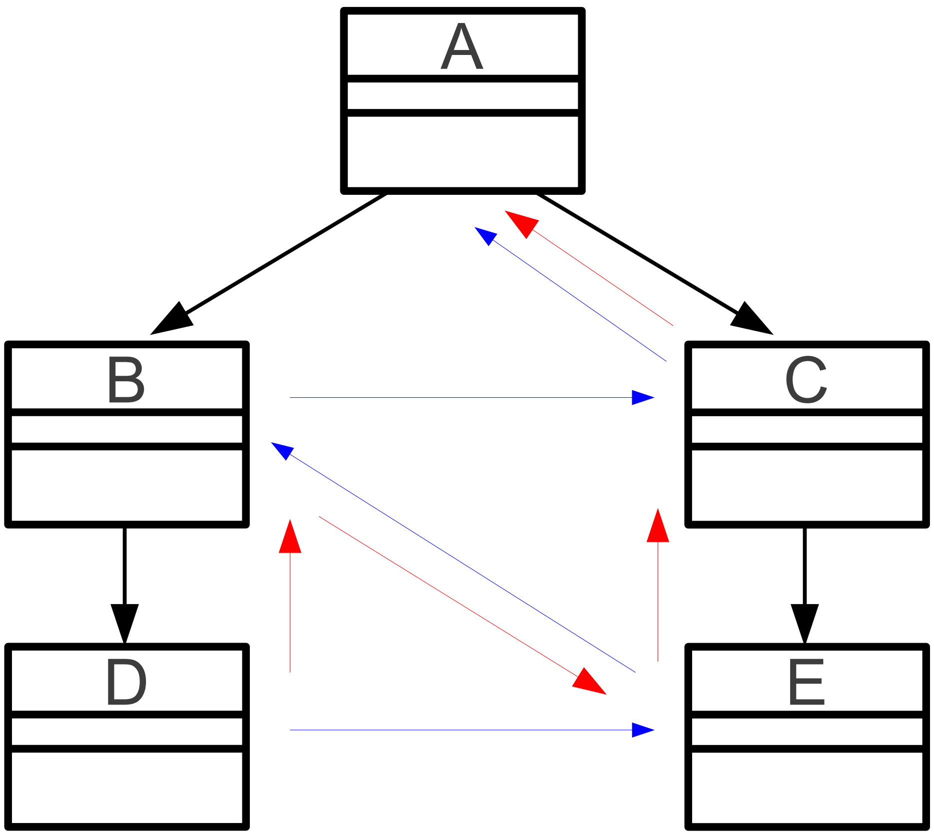


Bottom-up implementation

- Implement a module before those that use it
- Applicable if design is fully completed OR for bottom-level abstractions that are completed early

Bottom up example

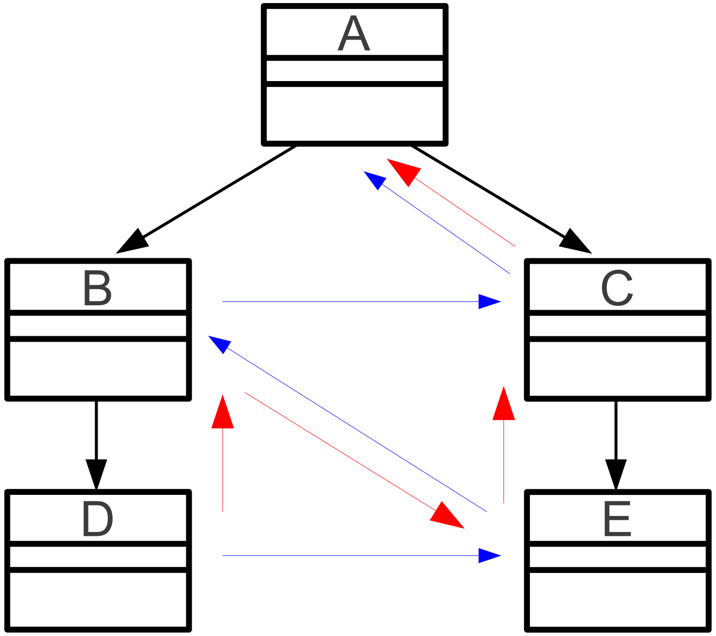
- D, E, B, C, A
- D, B, E, C, A



Pros & Cons

- Advantages:
  - less resource up front
  - early prototypes of sub-systems
- Disadvantages:
  - late detection of design errors
  - more test driver coding:

    one driver per component

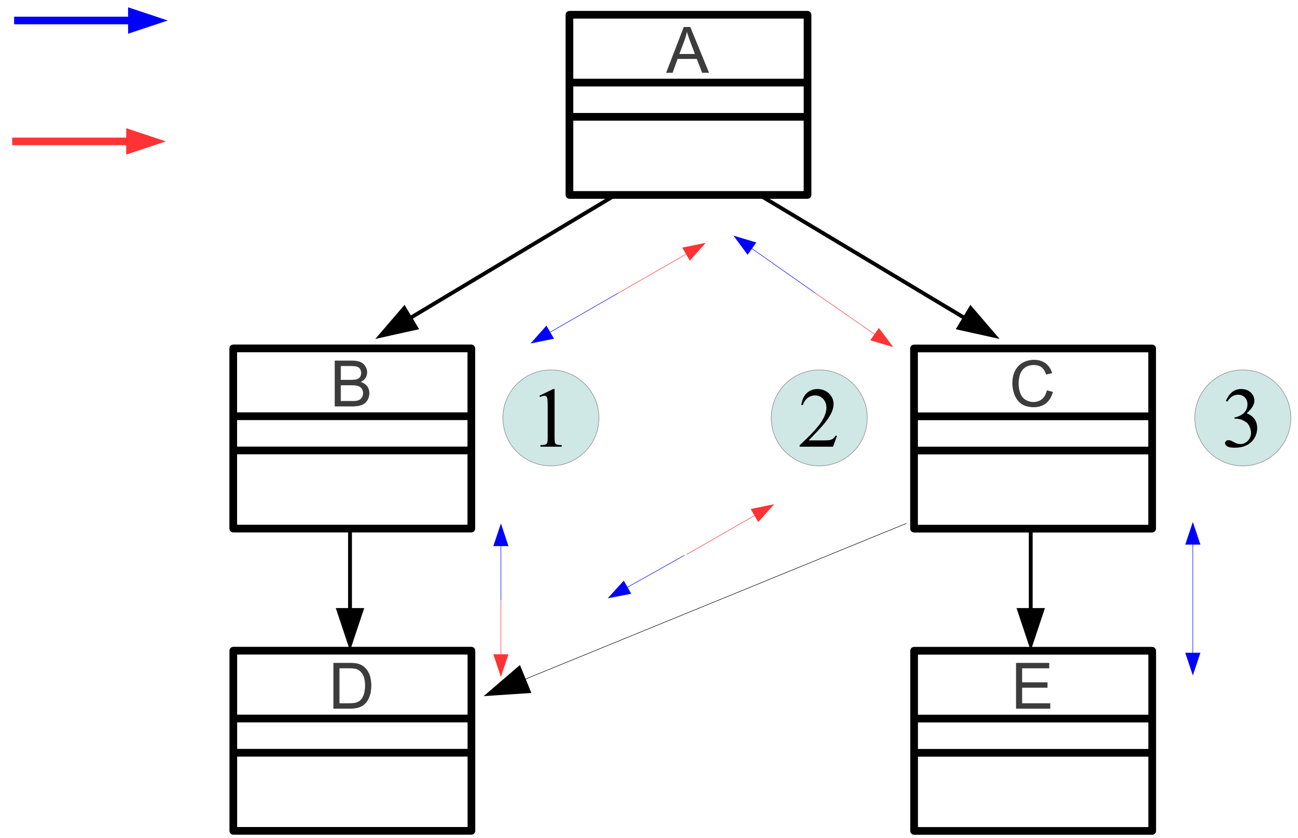


Hybrid implementation

- Mix top down and bottom up to minimise slag time
- Basically a top-down strategy that:
  - uses concrete implementations (instead of stubs) ***where possible***
  - components may be partially implemented
- Example: KEngine

Hybrid example

- completed
- partial



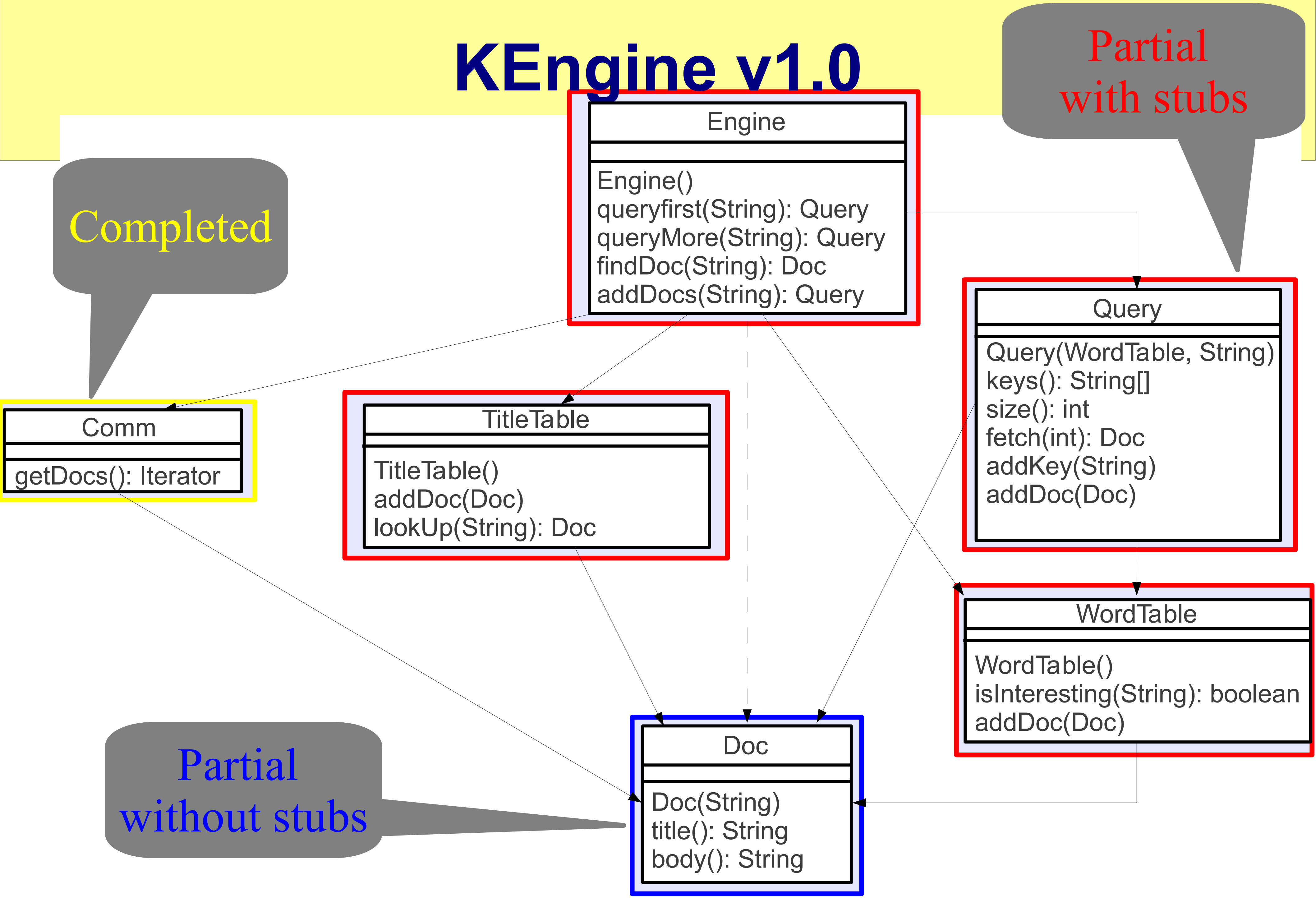
KEngine hybrid implementation

- Four iterations producing 4 versions:
  - 1.0
  - 2.0
  - 3.0
  - 4.0
- *May be* carried out in tandem with design
- A version is a system prototype

KEngine v1.0

- Scope: design iteration 1
- Components:
  - `Doc`: partial (without stubs)
    - not yet extract words
  - **`Comm`**: completed
  - `Query`: partial with stubs
  - `TitleTable`: partial with stubs
  - `WordTable`: partial with stubs
  - `Engine`: partial w.r.t other abstractions
    - empty query (words assumed uninteresting)

KEngine v1.0

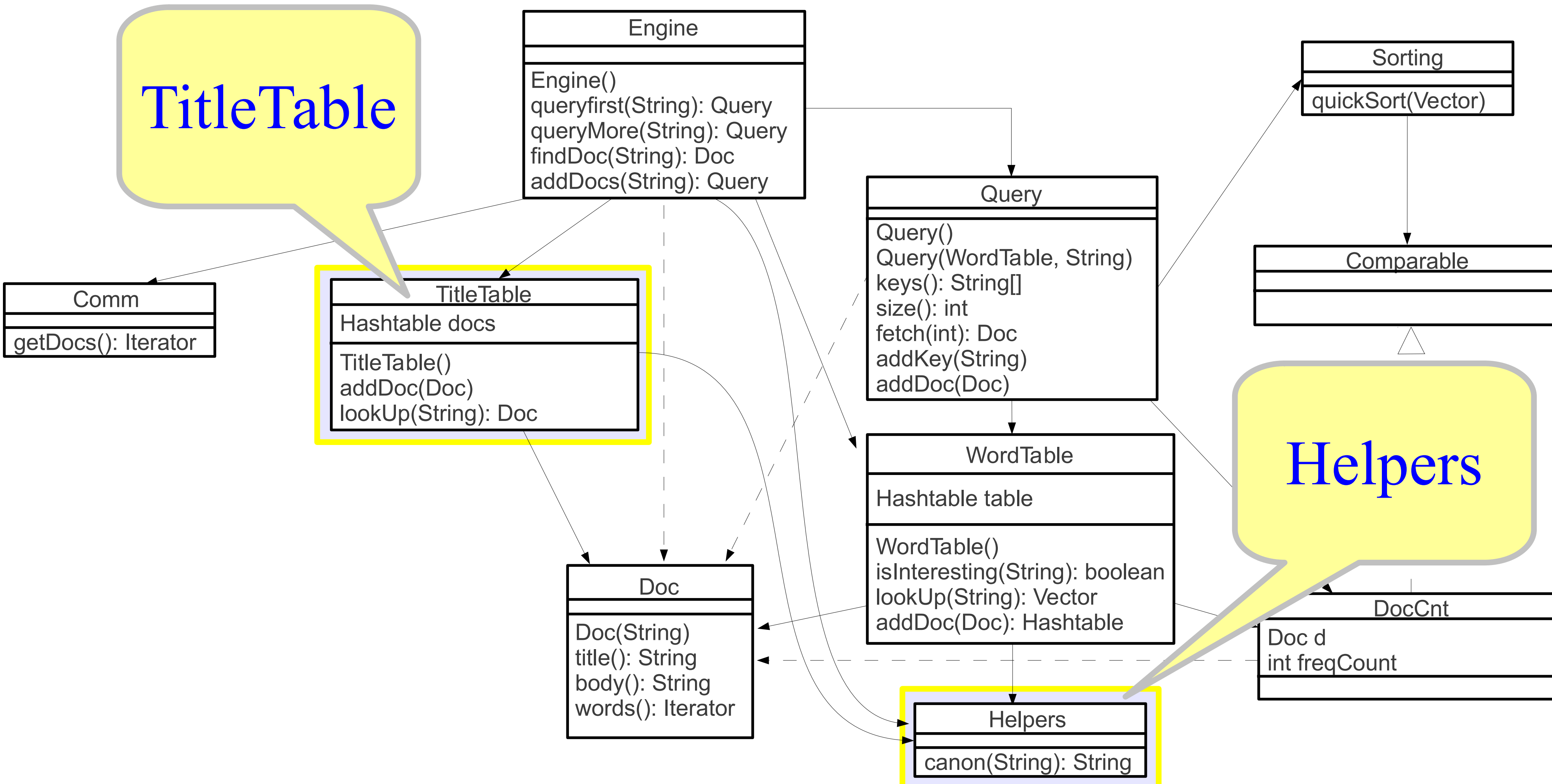


Completed
Partial with stubs
Partial without stubs
Engine
Engine()
queryfirst(String): Query
queryMore(String): Query
findDoc(String): Doc
addDocs(String): Query
Query
Query(WordTable, String)
keys(): String[]
size(): int
fetch(int): Doc
addKey(String)
addDoc(Doc)
Comm
getDocs(): Iterator
TitleTable
TitleTable()
addDoc(Doc)
lookUp(String): Doc
WordTable
WordTable()
isInteresting(String): boolean
addDoc(Doc)
Doc
Doc(String)
title(): String
body(): String

KEngine v2.0

- Scope: design iteration 2, part 1
- Components:
  - **`TitleTable`**: completed
  - **`Helpers`**: completed

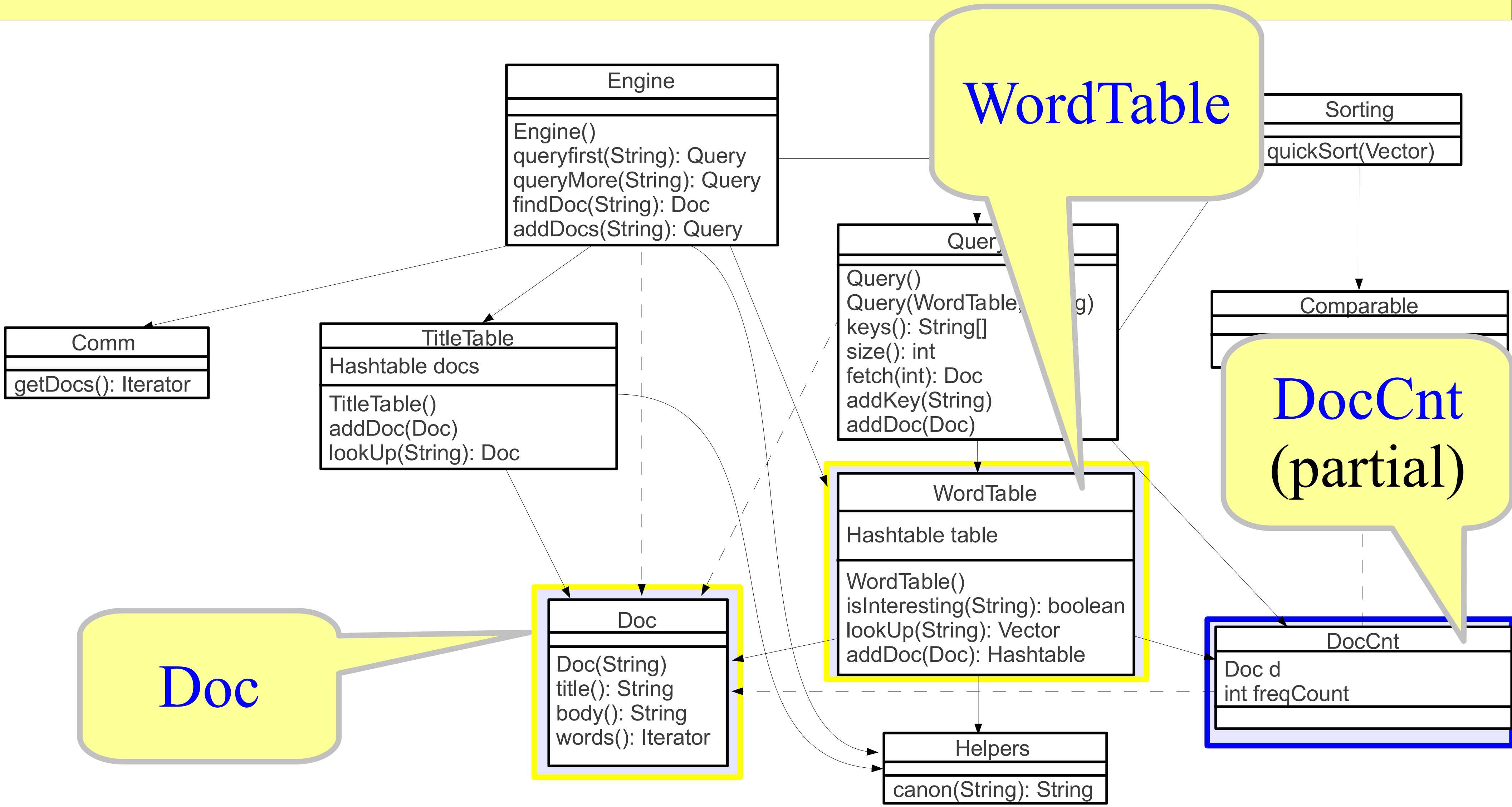
KEngine v2.0


TitleTable
Engine
Engine()
queryfirst(String): Query
queryMore(String): Query
findDoc(String): Doc
addDocs(String): Query
Sorting
quickSort(Vector)
Query
Query()
Query(WordTable, String)
keys(): String[]
size(): int
fetch(int): Doc
addKey(String)
addDoc(Doc)
Comparable
Comm
getDocs(): Iterator
TitleTable
Hashtable docs
TitleTable()
addDoc(Doc)
lookUp(String): Doc
Helpers
WordTable
Hashtable table
WordTable()
isInteresting(String): boolean
lookUp(String): Vector
addDoc(Doc): Hashtable
Doc
Doc(String)
title(): String
body(): String
words(): Iterator
DocCnt
Doc d
int freqCount
Helpers
canon(String): String

KEngine v3.0

- Scope: design iteration 2, part 2
- Components:
  - **`WordTable`**: completed
  - **`Doc`**: completed
  - `DocCnt`: partial
    - not yet implement `Comparable` interface

KEngine v3.0

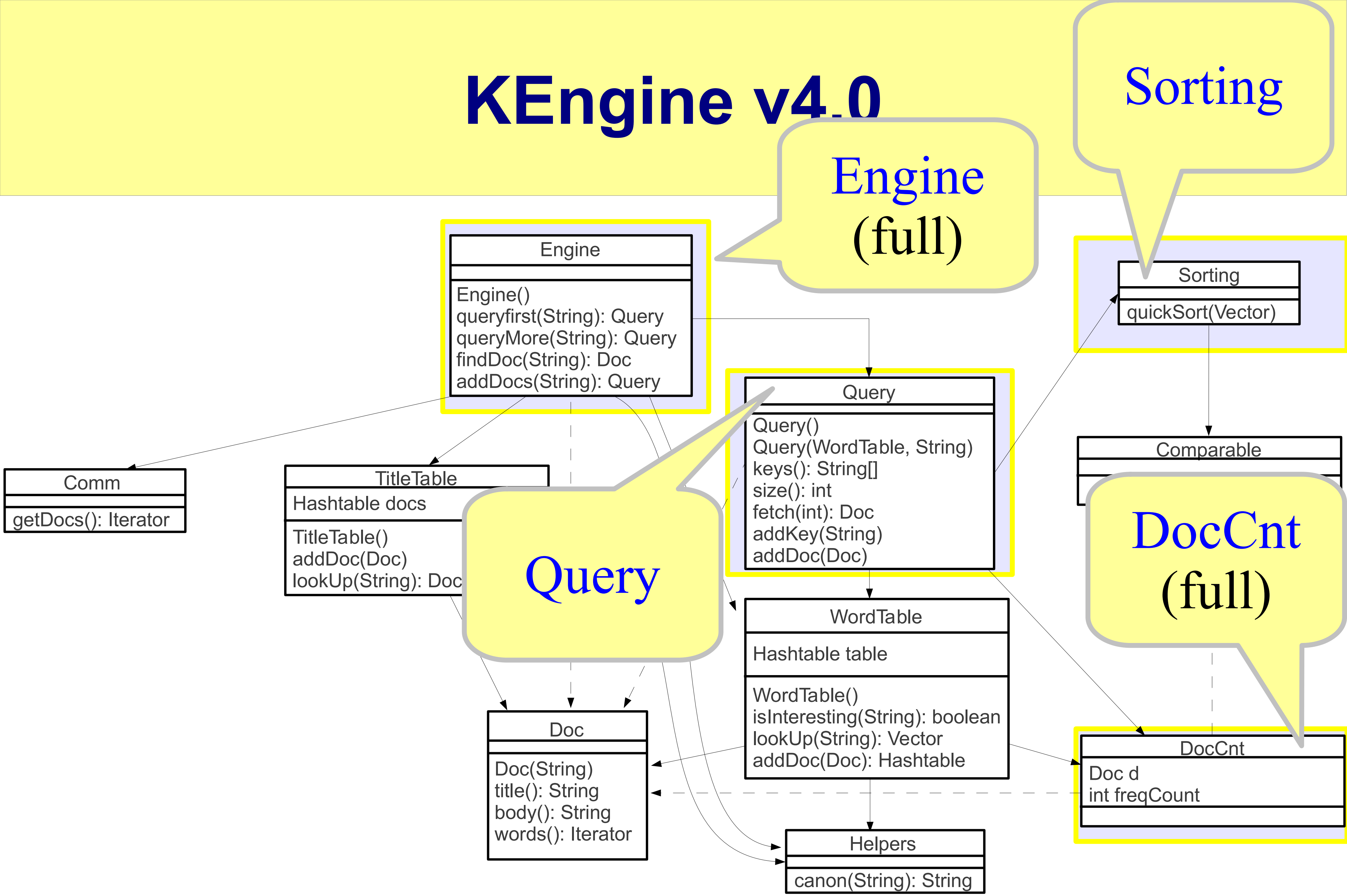


Engine
Engine()
queryfirst(String): Query
queryMore(String): Query
findDoc(String): Doc
addDocs(String): Query
WordTable
Sorting
quickSort(Vector)
Query
Query()
Query(WordTable, g)
keys(): String[]
size(): int
fetch(int): Doc
addKey(String)
addDoc(Doc)
Comparable
Comm
getDocs(): Iterator
TitleTable
Hashtable docs
TitleTable()
addDoc(Doc)
lookUp(String): Doc
DocCnt
(partial)
WordTable
Hashtable table
WordTable()
isInteresting(String): boolean
lookUp(String): Vector
addDoc(Doc): Hashtable
Doc
Doc
Doc(String)
title(): String
body(): String
words(): Iterator
DocCnt
Doc d
int freqCount
Helpers
canon(String): String

KEngine v4.0

- Scope: design iteration 2, part 3
- Components:
  - **`Engine`**: completed
  - **`Query`**
  - **`Sorting`**
  - **`DocCnt`**: completed
    - implement `Comparable`

KEngine v4.0


Sorting
Engine (full)
Engine
Engine()
queryfirst(String): Query
queryMore(String): Query
findDoc(String): Doc
addDocs(String): Query
Sorting
quickSort(Vector)
Comparable
Comm
getDocs(): Iterator
TitleTable
Hashtable docs
TitleTable()
addDoc(Doc)
lookUp(String): Doc
Query
Query
Query()
Query(WordTable, String)
keys(): String[]
size(): int
fetch(int): Doc
addKey(String)
addDoc(Doc)
DocCnt (full)
WordTable
Hashtable table
WordTable()
isInteresting(String): boolean
lookUp(String): Vector
addDoc(Doc): Hashtable
Doc
Doc(String)
title(): String
body(): String
words(): Iterator
DocCnt
Doc d
int freqCount
Helpers
canon(String): String

DEMO

KEngine

EXer
cise

Tutorial

- Program trio: implementation

Summary

- Design review helps informally evaluate a design before implementation
- Design is evaluated for correctness, performance and modularity
- Design walk through using symbolic test cases
- Implementation can be carried out in top-down, bottom-up or a mixture of the two (hybrid)

Q & A