*Development Process*

*sequence diagram*

*design specification*

*design class diagram*

iterative

output

design notebook

decomposition

abstraction

*sketch, alternatives, context*

*performance, modifiability*

develop detailed specifications

modular program structure

requirement specification

Iterative steps

top-down

iterations

start high-level

function

abstraction

helper

implement A

Abstraction (A) design specification

Iteration 0: initial abstractions

requirements specification: @checks, @effects

construct initial design class diagram association with *dependency* indicators

1: top-level abstractions

*analyse design spec. of each initial abstraction 🡪 new abstractions*

*Design class diagram* (UML)

concept class diagram : less *refine*d

module dependency diagram: less detailed

software classes

domain classes: representation + operations

Word, Keyword, NonKeyword 🡪 String

Build 🡨 concept class diagram / not ( scratch)

initial abstractions of KEngine

//startEngine

/\*\*

**@overview** represents keyword search engines

An **engine** holds a mutable collection of **documents** – obtained from some given URLs

The engine is able to pocess **keyword query** -> search for documents – contain **keywords**

The **matching** documents are ranked based on frequencies of keywords found in them

The engine has private file – contain list of uninteresting words

\*/

**class** Engine {

// need an abstraction -> represent the *engine*

* creates abstraction **Engine**

/\*\*  
\* **@effects**

\* If uninteresting words not retrievable

\* throws NotPossibleException

\* else

\* creates NonKeyword

\* initialises app. state appropriately

\*/

Engine() **throws** NotPossibleException

// query

/\*\*  
\* **@checks** w not in NonKeyword

\* **@effects**

\* Sets Keyword = {w}

\* makes Match – contain documents match w, ordered as required

\*

\*/

query(String w)

// need an abstraction 🡪 hold a keyword + store matches

// Keyword, NonKeyword: String

/\*\*  
\* **@effects**

\* If WORD(w) = false or w in NonKeyword

\* throws NotPossibleException

\* else

\* sets Keyword = {w}

\* performs new query

\* returns result

\*/

Query queryFirst(String w) **throws** NotPossibleException

// queryMore

/\*\*  
\* **@checks** Key != {}

\* w not in Keyword & NonKeyword

\* **@effects**

\* adds w 🡪 Keyword

\* makes Match – documents already in Match – additionaly match \* w (see above)

\* Orders Match properly.

\*/

queryMore(String w)

// need an abstraction 🡪 hold keywords + store matches

// 🡪 creates abstraction Query

// Keyword, NonKeyword: String

/\*\*  
\* **@effects**

\* If WORD(w) = false or w in Keyword/ NonKeyword or Key = {}

\* throws NotPossibleException

\* else

\* adds w to Keyword

\* returns query result

\*/

Query queryMore(String w) **throws** NotPossibleException

// findDoc

/\*\*  
\* **@checks** t is in titles

\* **@effects**

\* return d in **Document** s.t d’s title = t

\*

\*/

findDoc(String t)

// needs an abstraction 🡪 represent *Document*

// 🡪 uses abstraction *Doc*

/\*\*

\* **@effects**

\* If t not in Title

\* throws NotPossibleException

\* else

\* returns document with title t

\*/

Doc findDoc(String t) **throws** NotPossibleException

// addDocuments

/\*\*  
\* **@checks** u does not name a site in URL

\* u names a site – provide documents

\* **@effects**

\* adds u 🡪 **URL**

\* adds documents at site u with new titles to **Document**.

\* If **Keyword** – non-empty

\* adds any documents – match keywords - **Match**

\*/

addDocuments(String u)

// need an abstraction 🡪 represent *Document*

* creates abstraction Doc

/\*\*  
\* **@effects**

\* If u not a URL for a site – contain documents or u in URL

\* throws NotPossibleException

\* else

\* adds new documents 🡪 Doc

\* If no query was in progress

\* returns empty query result

\* else

\* returns query result – include any new \*matching documents

\*/

Query addDocs(String u) **throws** NotPossibleException

// end Engine

/\*\*

\* **@overview**

\* A textual document – contain a title + some text contents

\*/

**class** Doc {

} // end Doc

/\*\*

\* **@overview**

\* A query – consist of keywords - interest

\*/

**class** Query {

}

*Top-level data abstractions*

find all top-level abstraction

decomposition

sub-tasks: most significant ones first

**Engine**.queryFirst

Activity diagram

return (Query, Matches)

Determine keyword frequencies

Sort matching documents

match

keyword

Find matching documents

Query

interesting

Create a new query

check input String w (word)

record interesting + uninteresting words in same abstraction (WordTable)

|  |
| --- |
| WordTable |
|  |
| WordTable()  isInteresting(String): boolean  addDoc(Doc) |

*WordTable*

iteration abstraction

store words

check + maintain words set

*Doc*

|  |
| --- |
| Doc |
|  |
| Doc(String)  body(): String |

**Engine**.queryMore

filter

check

add

|  |
| --- |
| Query |
|  |
| Query(WordTable, String)  keys(): String[] // observe keywords  size(): int // retrieve match  fetch(int): Doc  addKey(String) |

**Engine**.findDoc

return

first matching document

title?

record documents + titles in *Title Table*

|  |
| --- |
| Doc |
|  |
| Doc(String)  title(): String  body():String |

|  |
| --- |
| TitleTable |
|  |
| TitleTable()  addDoc(Doc)  lookup(String): Doc |

*TitleTable*

iteration abstraction

store document

add + look up documents

**Engine**.addDocs

return query object

create empty query object

update *existing query*

add

collection

retrieve

documents

site with given URL

A new abstraction Comm: create a getDocs() 🡪 return an Iterator object for documents

|  |
| --- |
| Comm |
|  |
| getDocs(): Iterator |

|  |
| --- |
| Query |
|  |
| Query(WordTable, String)  keys(): String[]  size(): int  fetch(int): Doc  addKey(String)  addDoc(Doc) |

|  |
| --- |
| Doc |
|  |
| Doc(String)  title(): String  body(): String |

Design class diagram

|  |
| --- |
| 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 |

|  |
| --- |
| WordTable |
|  |
| WordTable()  isInteresting(String): boolean  addDoc(Doc) |

|  |
| --- |
| TitleTable |
|  |
| TitleTable() addDoc(Doc)  lookUp(String): Doc |

|  |
| --- |
| Doc |
|  |
| Doc(String)  title(): String  body(): String |

|  |
| --- |
| Engine |
| * wt: WordTable * t: TitleTable * q: Query * urls: String[] |
| Engine()  queryFirst(String): Query  queryMore(String): Query  findDoc(String): Doc  addDocs(String): Query |

/\*\*

\* **@overview** … (omitted) …

\* **@version** (iteration) 1.0

\*/

**class** Engine {

@DomainConstraint(type = “WordTable”, optional = **false**)

**private** WordTable wt;

@DomainConstraint(type = “TitleTable”, optional = **false**)

**private** TitleTable tt;

@DomainConstraint(type = “Query”)

**private** Query q;

**private** String[] urls;

///// END version 1.0

} // end Engine

/\*\*

\* **@overview** keeps track of interesting + uninteresting words.

\* uninteresting words – obtain from private file

\* records number of times each interesting word occurs in document

\* **@version** (iteration) 1.0

\*/

**class** WordTable {

/\*\*

\* **@effects**

**\*** If uninteresting-word file cannot be read

**\*** throws NotPossibleException

**\***  else

**\*** initialises this -> contain all words in file

\*

\*/

WordTable() **throws** NotPossibleException

/\*\*

\* **@effects**

**\*** If w null/ nonword/ uninteresting word

**\*** return false

**\***  else

**\*** return true

\*

\*/

**boolean** isInteresting(String w)

/\*\*

\* **@requires** d not null

**\* @modifies** this

**\* @effects**

**\***  add 🡪 this interesting words of d with their numbers of

**\*** occurrences

\*

\*/

**void** addDoc(Doc d)

} // end WordTable

/\*\*

\* **@overview**

\* provides in4 🡪 **keywords** of query + documents – **match** those

\* Documents accessed indexes: 0 – size

\* Documents **ordered** by **number of matches** they contain

\* document 0th contain the most matches

\*

\* **@version** (iteration) 1.0

\*/

**class** Query {

/\*\*  
 \* **@effects**  returns an empty query

\*/

Query()

/\*\*  
 \* **@effects**  returns a count of documents – match query

\*/

**int** size()

/\*\*  
 \* **@effects**

\* If 0 <= i < size

\* returns ith matching document

\* else

\* throws IndexOutOfBoundException

\*/

Doc fetch(**int** i) **throws** IndexOutOfBoundException

/\*\*  
 \* **@effects**  returns keywords of this

\*/

String[] keys()

/\*\*

\* **@requires** w not null

**\* @modifies** this  
 \* **@effects**

\* If this empty/ w already a keyword in this

\* throws NotPossibleException

\* else

\* modifies this 🡪 contain w + all keywords in this

\*/

**void** addKey(String w) **throws** NotPossibleException

/\*\*

\* **@requires** d not null

**\* @modifies** this  
 \* **@effects**

\* If this empty & d contain all keywords of this

\* adds d -> this as query result

\* else

\* do nothing

\*/

**void** addDoc(Doc d)

} // end Query

/\*\*

\* **@overview** keeps track of documents + titles

\*

\* **@author** dmle

\*

\* **@version** (iteration) 1.0

\*/

**class** TitleTable {

/\*\*  
 \* **@effects**  intialises this to be empty

\*/

TitleTable()

/\*\*

\* **@requires** d not null

**\* @modifies** this  
 \* **@effects**

\* If a document with d’s title already in this

\* throws DuplicateException

\* else

\* adds d with its title to this

\*/

**void** addDoc(Doc d) **throws** DuplicateException

/\*\*

\* **@effects**

\* If t null/ no document with title t in this

\* throws NotPossibleException

\* else

\* returns document with title t

\*/

Doc lookup(String t) **throws** NotPossibleException

} // end TitleTable

/\*\*

\* **@overview**

\* represents communication module responsible 🡪 obtain documents

\* from remote sites

\* **@version** (iteration) 1.0

\*/

**public class** Comm {

/\*\*

\* **@effects**

\* If u not valid URL/ site it names fails -> respond

\* throws NotPossibleException

\* else

\* returns a generator -> documents from site u

\* (as strings)

\*/

**static** Iterator getDocs(String u) **throws** NotPossibleException {

} // end Comm

*Refinement*

Abstraction selection criteria

* complete spec. (not yet refined)
* uncertainty
* more 🡪 desing

Abstraction: TitleTable Query Comm.getDocs(library)

TitleTable

lookup

addDoc

find a document

extract title

document titles: re-used many times

* data structure map Docs 🡪 Strings
* *java.util.Hashtable*

|  |
| --- |
| WordTable |
|  |
| WordTable()  isInteresting(String): boolean  lookUp(String): Vector  addDoc(Doc) |

Query

Query(WordTable, String)

frequencies

observe

occurrence

sort

find

keyword

documents

|  |
| --- |
| Query |
| -k: WordTable  -keys: String // store keywords |
| Query(WordTable, String)  keys(): String[] // keep track of keywords  size(): int  fetch(int): Doc  addKey(String)  addDoc(Doc) |

|  |
| --- |
| WordTable |
|  |
| WordTable()  isInteresting(String): boolean  lookUp(String): Vector  addDoc(Doc) |

ith document

retrieve

fetch(int): (current matches)

* index-based collection 🡪 store (Vector)

addKey(String)

sumFreq

sort

new

find

duplicate

check

maintain sumFreq 🡪 each match 🡪 sort matches

* DocCnt<Document, Count> abstraction for matches
* Vector 🡪 store matches (DocCnt objects)
* quick-sort

|  |
| --- |
| Sorting |
|  |
| quicksort(Vector) |

|  |
| --- |
| Comparable |
| compareTo(Object o) |
|  |

|  |
| --- |
| DocCnt |
| -d: Doc  -cnt: int |
| DocCnt(Doc, int)  getDoc(): Doc  getCount: int  toString(): String |

|  |
| --- |
| Query |
| - k: WordTable  - matches: Vector  - keys: String[] |
| Query(WordTable, String)  keys():String[]  size(): int  fetch(int): Doc  addKey(String)  addDoc(Doc)  toString(): String |

addDoc(Doc)

update sorting

add doc

check

matches

current keyword

update WordTable.addDoc 🡪 return HashTable

(map keywords – frequencies)

Query.addDoc(Doc) 🡪 Query.addDoc(Doc, Hashtable)

|  |
| --- |
| Query |
|  |
| -k: WordTable  -matches: Vector  -keys: String[] |
| Query(WordTable, String)  keys(): String[]  size(): int  fetch(int): Doc  addKey(String)  addDoc(Hashtable, Doc) |

|  |
| --- |
| WordTable |
|  |
| WordTable()  isInteresting(String): boolean  lookUp(String): Vector  addDoc(Doc): Hashtable |

addDoc(Doc)

WordTable

create

Doc: iterator method 🡪 iterate over all words

* Doc.words(): Iterator method

record each keyword Set of DocCnt objects

* WordTable.table: keyword – Vector DocCnts

canonical word forms

* Helpers.canon: convert words 🡪 common format

|  |
| --- |
| Doc |
|  |
| Doc(String)  title(): String  body(): String  words(): Iterator |

|  |
| --- |
| WordTable |
| -table: Hashtable |
| WordTable()  isInteresting(String): boolean  lookUp(String): Vector  addDoc(Doc): Hashtable |

|  |
| --- |
| Helpers |
|  |
| canon(String): String |

|  |
| --- |
| Query |
|  |
| Query()  Query(WordTable, String)  keys(): String[]  size(): int  fetch(int): Doc  addKey(String) addDocs(Doc) |

|  |
| --- |
| Engine |
|  |
| Engine()  queryFirst(String): Query  queryMore(String): Query  findDoc(String): Doc  addDocs(String): Query |

|  |
| --- |
| Comm |
|  |
| getDocs(): Iterator |

|  |
| --- |
| TitleTable |
| Hashtable docs |
| TitleTable()  addDoc(Doc)  lookUp(String): Doc |

|  |
| --- |
| Comparable |
|  |
|  |

|  |
| --- |
| Sorting |
|  |
| quicksort(Vector) |

|  |
| --- |
| WordTable |
| Hashtable table |
| WordTable()  isInteresting(String): boolean  lookUp(String): Vector  addDoc(Doc): Hashtable |

|  |
| --- |
| Helpers |
|  |
| canon(String): String |

|  |
| --- |
| DocCnt |
| Doc d  int freqCount |
|  |

|  |
| --- |
| Doc |
|  |
| Doc(String)  title(): String  body(): String  words(): Iterator |

*Design class diagram*

*Query implementation sketches*

/\*\*  
\* **@requires** wt & w not null

\* **@effects** initialises this – contain w

\*

\* **@pseudocode** <pre> --- implement sketch -----

\* lookup key in WordTable

\* sort matches 🡪 quicksort

\* </pre>

\*/

Query(WordTable wt, String w)

/\*\*

\* **@requires** …

\* **@modifies** …

\* **@effects** …

\*

\* **@pseudocode** <pre> --- implement sketch -----

lookup new key in WordTable

store information 🡪 matches in hash table

for each current match,

if document in hash table

look up

store in vector sort 🡪 quicksort

</pre>

\*/

**void** addKey(String w) **throws** NotPossibleException

/\*\*

\* **@requires** …

\* **@modifies** …

\* **@effects** …

\*

\* **@pseudo** <pre> --- implementation sketch -----

use argument table 🡪 get number of occurrences of each current key

if document has all keywords

compute sum

insert (doc, sum) pair in vector of matches

</pre>

\*/

**void** addDoc(Doc d)

*Design process*

Top-down design approach: decomposition by abstraction (create, refine as needed)

design diagram / sequence diagram 🡪 design updates

Design: iterative

more details

(later iterations program structure)

sequence diagram 🡪 validated