

Reverse Engineering Pharo's LRUCache

G. Polito, S. Ducasse

The Task

- We need to use an LRUCache for some project
- We need to understand
 - What it is
 - How it is used
- A bit the implementation just in case

How are we going to do?

FOCUS

BACKLOG

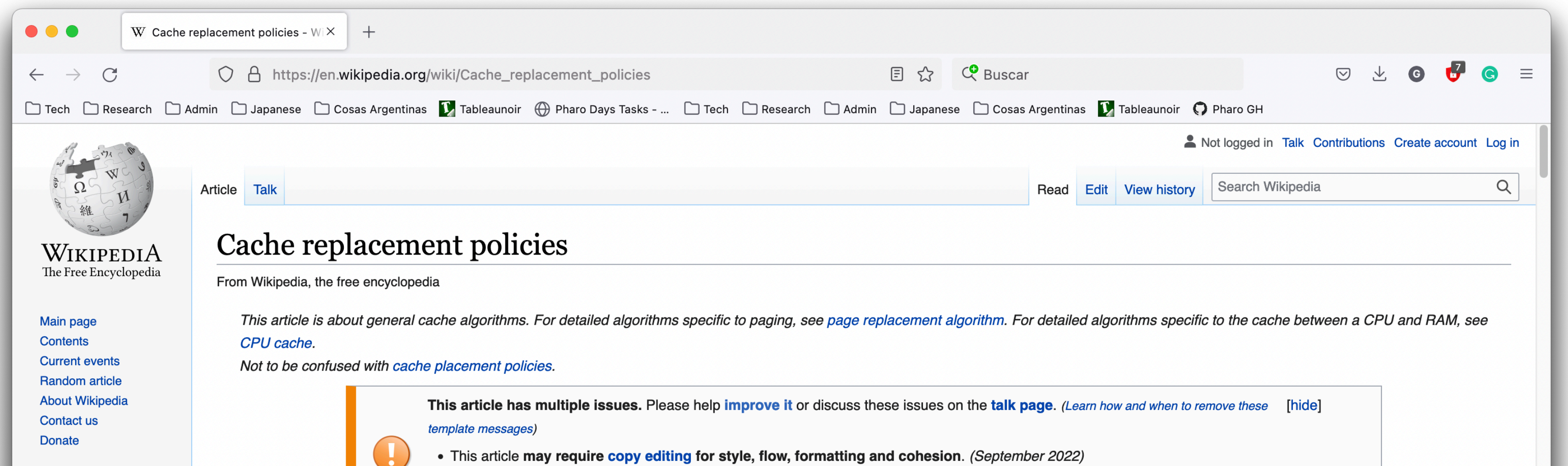
a.k.a.

the
ignore
list

First: What is an LRU Cache?

FOCUS: High-level view

- A cache => known from previous courses
- With a *least recently used* policy => it removes elements when full!
- General info on how caches work, from wikipedia

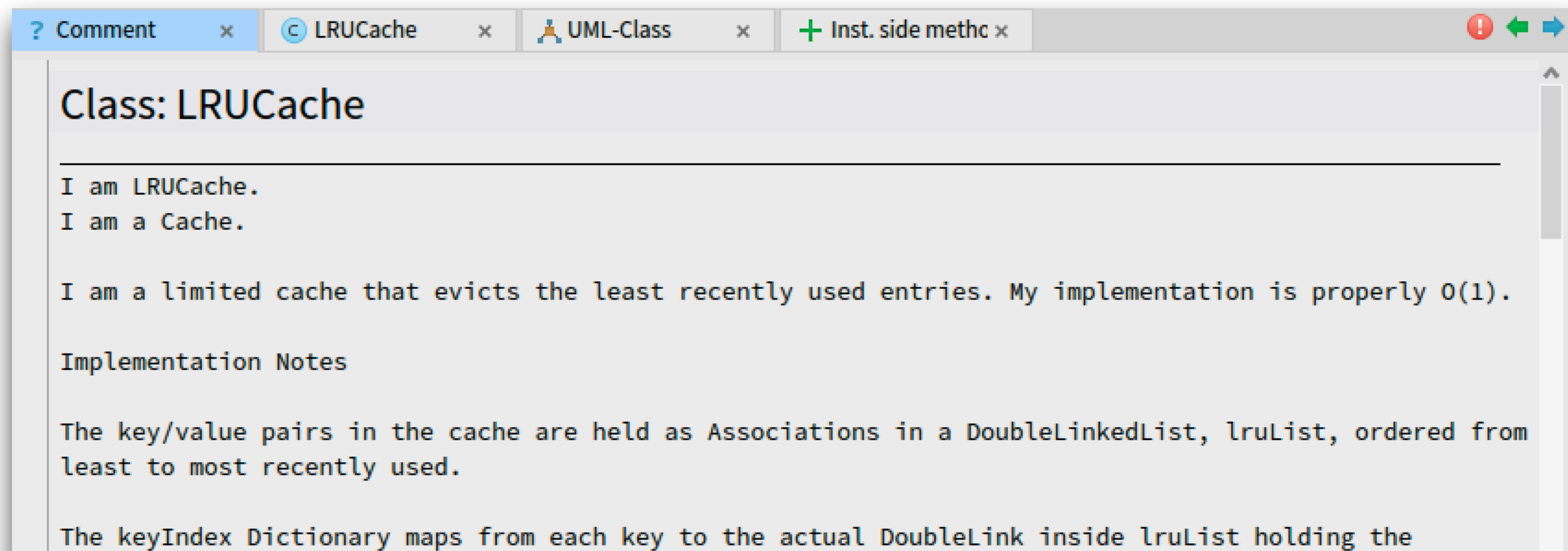
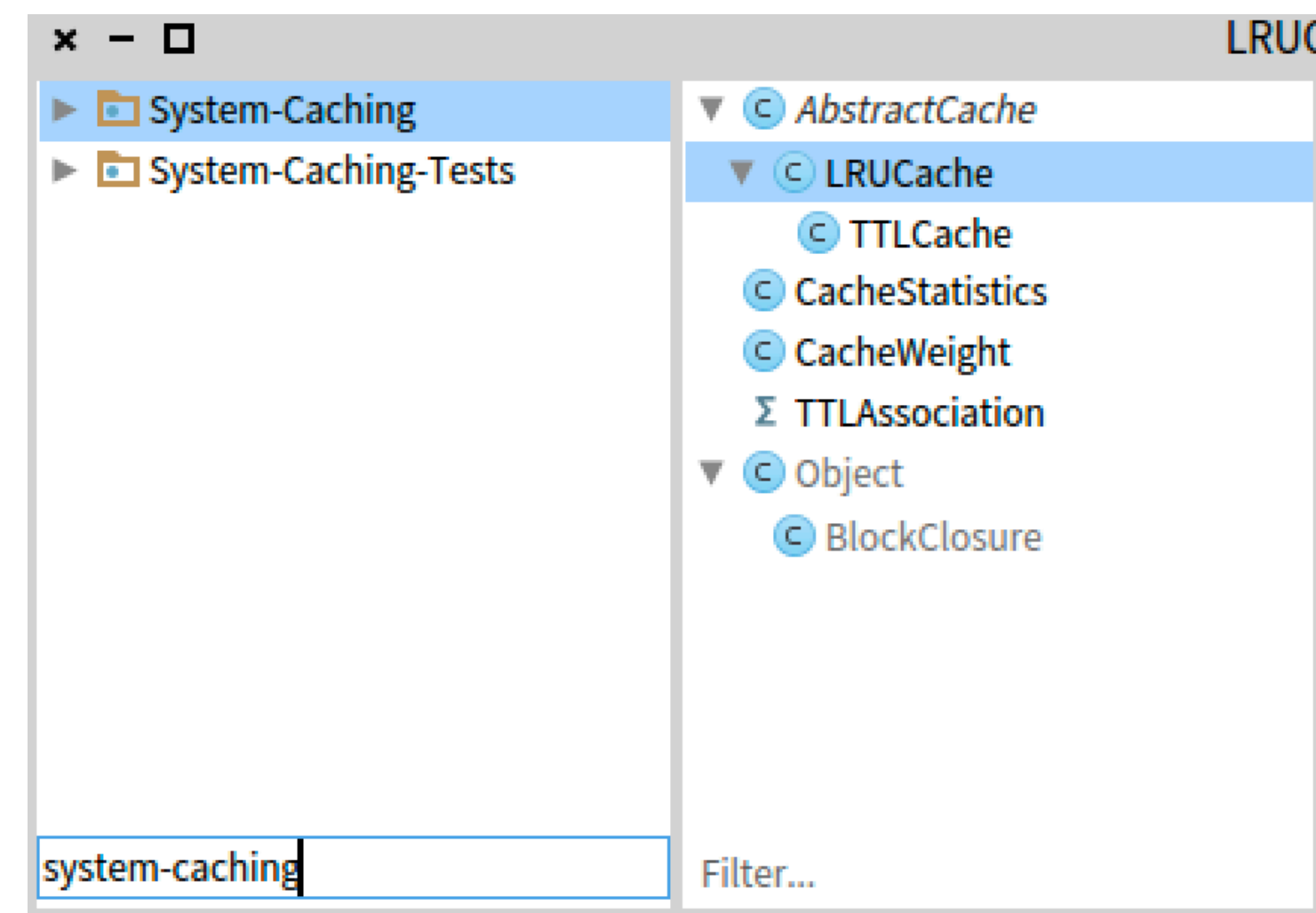


BACKLOG

A Map of the Code

FOCUS: High-level view

- Statically learn what is there
- Focus on the main class: LRUCache
- Read the comment!



BACKLOG

TTLCache
Weight
Statistics
Extensions?

User perspective - How does it work?

FOCUS: API

- Learned from the comment:
 - It *seems* to work as a dictionary and store (key, value) pairs
 - There are examples!
 - Important method: **at:ifAbsentPut:**

```
primeFactorsCache := LRUCache new.
```

```
50 timesRepeat: [  
  | n |  
  n := 100 atRandom.  
  primeFactorsCache  
    at: n ifAbsentPut: [ n primeFactors ] ].
```

BACKLOG

TTLCache
Weight
Statistics
Extensions?

Pharo things:
atRandom
primes

User perspective - How is it used?

FOCUS: API

- Senders of `at:ifAbsentPut:`
 - Bad idea! same API as Dictionary, *lots* of false positives

Class	Method	Category	Package
AbstractCache	at:	accessing	System-Caching
AthensCairoSurface	cacheAt:ifAbsentPut:	caching	Athens-Cairo
AthensCanvas	setPaint:	paint	Athens-Core
Behavior	ensureProperties	accessing - properties	Kernel
Behavior	propertyAt:put:	accessing - properties	Kernel
Class class	allSuperclassesFor:cache:	file in/out	Kernel
BitBlt	colorConvertingMap:from:to:keepSu	private	Graphics-Primitives
BlockClosure	memoizedUsing:	*System-Caching	System-Caching
CCodeGenerator	checkForGlobalUsage:in:	utilities	Slang
CCodeGenerator	localizeGlobalVariables	utilities	Slang
CairoBackendCache	at:ifAbsentPut:	accessing	Athens-Cairo
CairoBackendCache	for:at:ifAbsentPut:	accessing	Athens-Cairo

BACKLOG

TTLCache
Weight
Statistics
Extensions?

Pharo things:
atRandom
primes
classes

User perspective - How is it used? Try #2

FOCUS: API

- References of the LRUCache class
 - Better! Only 11 users. We can read them all!

Sender	Method	Package	Project
AthensCanvas	paintCache	paint	Athens-Core
BlockClosureTest	testMemoizedLRUCache	*System-Caching-Tests	System-Caching-Tests
CoSession	initialize	initialization	HeuristicCompletion-Model
GradientFillStyle class	initPixelRampCache	private - initialization	Graphics-Canvas
IceRepository	commitsInPackageCache	private - commits	Iceberg
IceLibgitRepository	commitCache	private - commits	Iceberg-Libgit
LRUCacheTest	newCache	accessing	System-Caching-Tests
MCTool class	mcVersionCache	accessing	MonticelloGUI
MicHTTPResourceReference class	resourcesCache	testing	Microdown
UITheme	createArrowImagesCache	scrollbars	Polymorph-Widgets
UITheme	createBoxImagesCache	scrollbars	Polymorph-Widgets

BACKLOG

TTLCache
Weight
Statistics
Extensions?

Pharo things:
atRandom
primes
classes

Analysis of Class Users

FOCUS: API

- 2 tests, 9 “other” legitimate users
- Other 9 usages use two new API methods we did not check

```
LRUCache new  
  maximumWeight: 20
```

```
LRUCache new  
  maximumWeight: 20;  
  factory: [ :key | ... ];  
  yourself
```

BACKLOG

TTLCache
Weight
Statistics
Extensions?

Pharo things:
atRandom
primes
classes

A Decision: Where to Continue?

FOCUS: API

- 2 tests, 9 “other” legitimate users
- Other 9 usages use two new API methods we did not check

```
LRUCache new  
  maximumWeight: 20
```

```
LRUCache new  
  maximumWeight: 20.  
  factory: [ :key | ... ];  
  yourself
```

For later

**at:ifAbsentPut: seems enough
to add elements for now**

BACKLOG

TTLCache
Weight
Statistics
Extensions?
factory:

Pharo things:
atRandom
primes
classes

Advanced Usage - Maximum Weight

FOCUS: API - weight

- **The comment says nothing** about the weight!
- Let's check the implementation => read some code

- `maximumWeight`: is not in `LRUCache`
 - => check the **superclass**
 - **Found** in `AbstractCache`



BACKLOG

TTLCache

★ *Weight*

Statistics

Extensions?

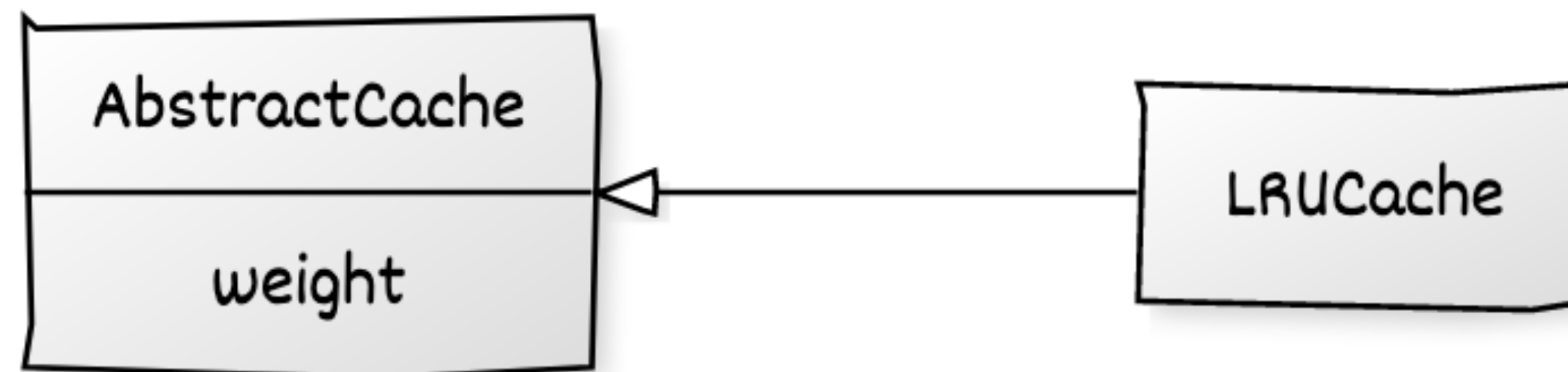
factory:

Pharo things:
atRandom
primes
classes

AbstractCache

FOCUS: API - weight

- AbstractCache defines weight
- *AbstractCache comment* says what this is about



```
? Comment x AbstractCache x UML-Class x + Inst. side methc x
Class: AbstractCache
-----
I am Cache.
I am an abstract class.

I am a limited cache holding onto key/value pairs.

My primary interface is #at:ifAbsentPut: which takes two arguments: a key and a block. Either the key
```

BACKLOG

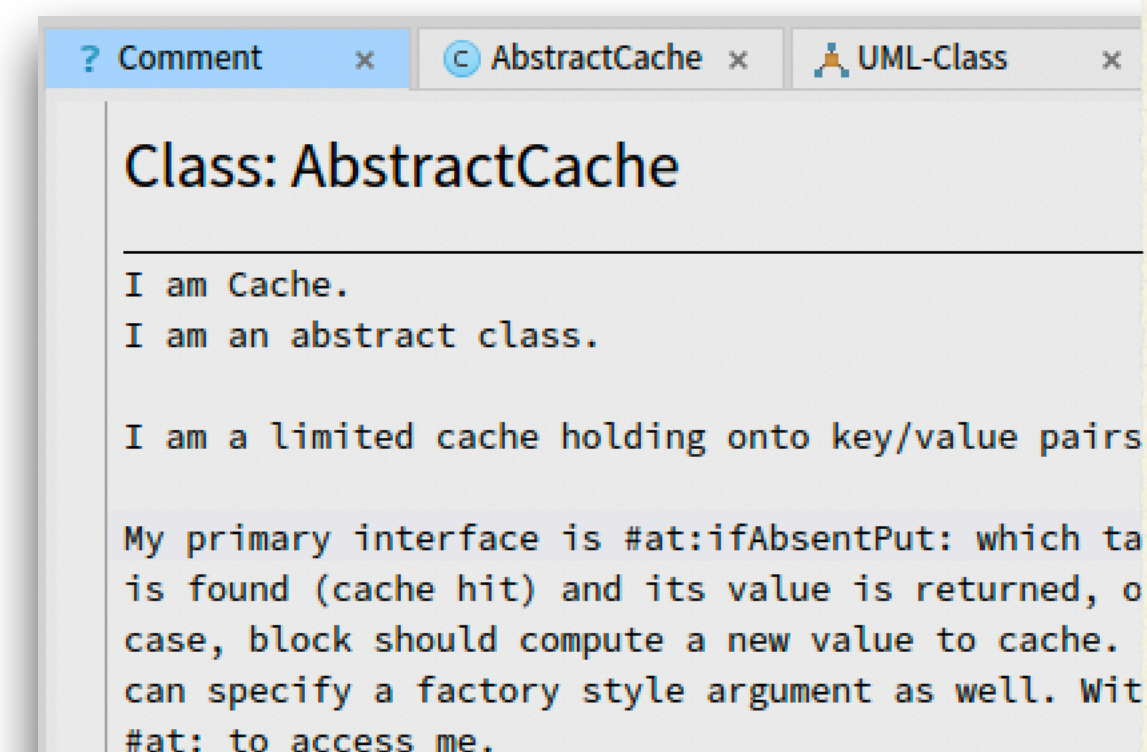
TTLCache
Statistics
Extensions?
factory:
weight impl!

Pharo things:
atRandom
primes
classes

Information Confirmed by the Comment

FOCUS: API - weight

- `at:ifAbsentPut:` is the **primary API method** (interface) !
- It contains (key,value) pairs
- New information
 - a cache has a weight (capacity)
 - a cache has a max weight (max capacity)



```
? Comment x AbstractCache x UML-Class x
Class: AbstractCache
I am Cache.
I am an abstract class.
I am a limited cache holding onto key/value pairs
My primary interface is #at:ifAbsentPut: which ta
is found (cache hit) and its value is returned, o
case, block should compute a new value to cache.
can specify a factory style argument as well. Wit
#at: to access me.
```

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!

Pharo things:
atRandom
primes
classes

Implementor's Hat: How Insertions Work

FOCUS: Implementation insert + evict

- Hypothesis
 - A *hit* means we find an element in the cache
 - A *miss* means we did not find it, we should add it
 - If we reach the capacity, we should *evict* something (LRU policy)



BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!

Pharo things:
atRandom
primes
classes

Insertion Implementation

FOCUS: Implementation insert + evict

```
LRUCache >> at: key ifAbsentPut: block
[...]  
association := keyIndex  
associationAt: key  
ifAbsent: [ | value |  
           value := block cull: key.  
           [...]  
           association := self newAssociationKey: key value: value.  
           [...]  
           ^ self handleMiss: association ].  
^ self handleHit: association
```

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!

Pharo things:
atRandom
primes
classes

Ignoring Complex Details at First

FOCUS: Implementation insert + evict

```
LRUCache >> at: key ifAbsentPut: block  
[...]
```

Semaphores

```
association := keyIndex  
associationAt: key  
ifAbsent: [ | value |  
value := block cull: key.  
[...]
```

Double checks

```
association := self newAssociationKey: key value: value.  
[...]  
^ self handleMiss: association ].  
^ self handleHit: association
```

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!

Pharo things:
atRandom
primes
classes

Focus on the Important

FOCUS: Implementation insert + evict

```
LRUCache >> at: key ifAbsentPut: block
[...]
```

```
association := keyIndex
associationAt: key
ifAbsent: [ | value |
    value := block cull: key.
    [...]
```

```
association := self newAssociationKey: key value: value.
[...]
```

```
^ self handleMiss: association ].
```

```
^ self handleHit: association
```

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks

Pharo things:
atRandom
primes
classes
concurrency

Where is the Insertion?

FOCUS: Implementation insert + evict

```
LRUCache >> at: key ifAbsentPut: block
[...]
```

*If we "miss",
we should insert
the value*

```
    association := keyIndex
    associationAt: key
    ifAbsent: [ | value |
        value := block cull: key.
        [...]
```

handleMiss

```
        association := self newAssociationKey: key value: value.
        [...]
```

```
    ^ self handleMiss: association ].
^ self handleHit: association
```

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks
hits

Pharo things:
atRandom
primes
classes
concurrency

Confirming the Hypothesis

FOCUS: Implementation insert + evict

```
handleMiss: association
  | link |
  statistics addMiss.
  self addWeight: association value.
  link := lruList addLast: association.
  keyIndex at: association key put: link.
  ^ association value
```

Bingo

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks
hits

Pharo things:
atRandom
primes
classes
concurrency

Where is the eviction?

FOCUS: Implementation insert + evict

```
handleMiss: association  
  | link |  
  statistics addMiss.  
  self addWeight: association value.  
  link := linkList addLast: association.  
  keyIndex at: association key put: link.  
  ^ association value
```

Candidates

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks
hits

Pharo things:
atRandom
primes
classes
concurrency

Looking in the Statistics

FOCUS: Implementation insert + evict

- Implementors of addMiss

Nope, just an increment

```
CacheStatistics >> addMiss  
    misses := misses + 1
```

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks
hits

★ addWeight:

Pharo things:
atRandom
primes
classes
concurrency

Stepping Back

FOCUS: Implementation insert + evict

- Implementors of addMiss

```
handleMiss: association
  | link |
  statistics addMiss.
  self addWeight: association value.
  link := lruList addLast: association.
  keyIndex at: association key put: link.
  ^ association value
```

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks
hits

Pharo things:
atRandom
primes
classes
concurrency

Backtracking and Trying Other Path

FOCUS: Implementation insert + evict

- Implementors of addWeight:

```
addWeight: value
weight add: value
[ weight isBelowMaximum ]
  whileFalse: [
    self isEmpty
    ifTrue: [ self error: '...' ]
    ifFalse: [ self evict ] ]
```

Bingo again!

BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks
hits

LRU?
why loop?

Pharo things:
atRandom
primes
classes
concurrency

What did we learn?

- LRU Cache is a cache with a *Least Recently Used* policy
- Works as a (key, value) pair
- Main API: **at:ifAbsentPut:**
- Has a max capacity, *evicts* elements to not surpass it

What did we *NOT* learn?

- How the LRU policy is implemented
- How is the weight/eviction implemented?
- Is it thread-safe? How? How does Pharo concurrency work?

- Many classes: Statistics, TTLCache...
- How do Pharo random generators work?
- ...

We ignored more than what we learned



BACKLOG

TTLCache
Statistics
Extensions?
factory:
weight impl!
Semaphores
Two checks
hits

LRU?
why loop?

Pharo things:
atRandom
primes
classes
concurrency

How did we learn?

- We **focused on the target**
- Flow: High-level View => Usage => Implementation
- We ignored things *not in focus*, and kept a log for later

- Comments had important info: the **why** of the design
- Senders show **examples of users!**
- Methods were too detailed: learn *what lines to ignore*