Introduction to Iterators

Marcus Börger

international **PHP**2004 conference
– spring edition -

Introduction to Iterators

- ☑ What are Iterators
- ☑ The basic concepts

What are Iterators

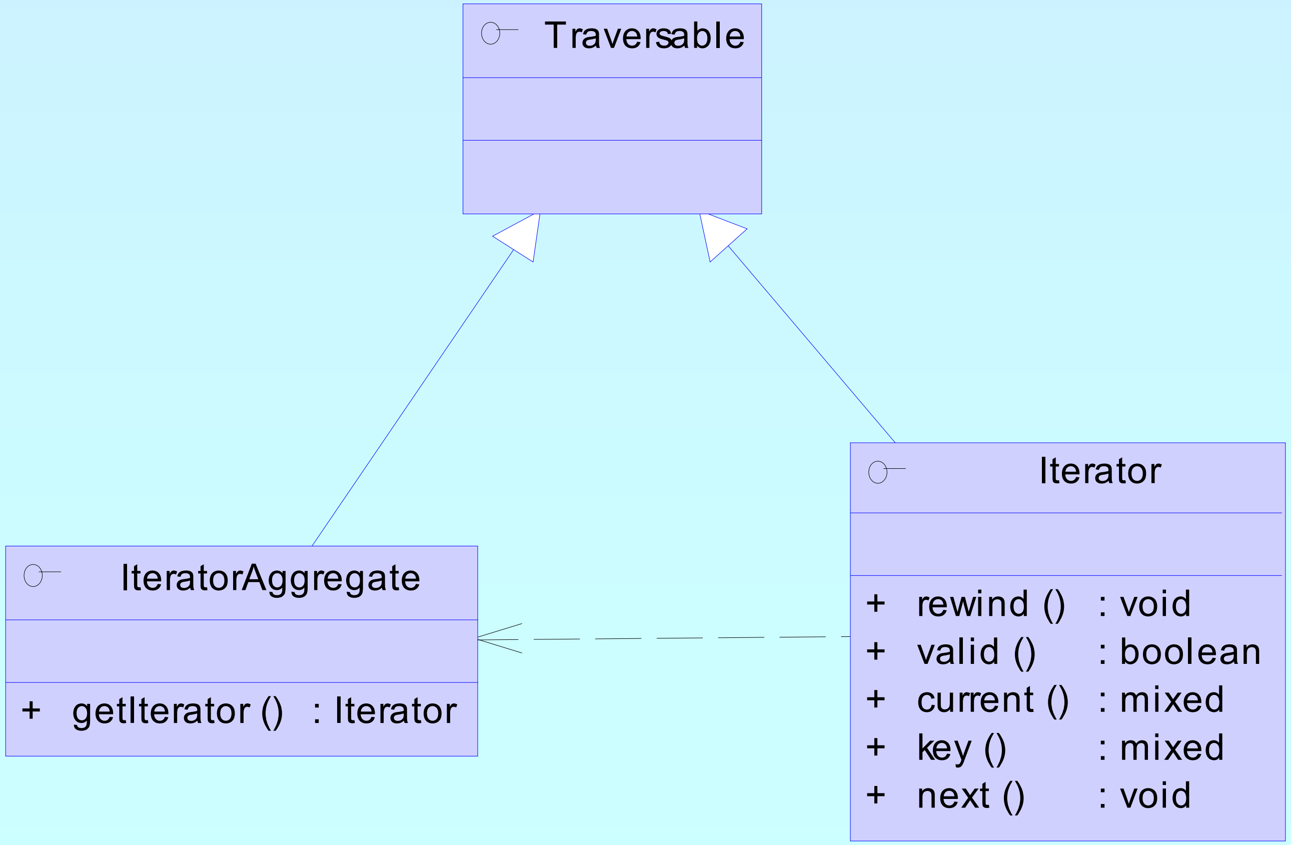
- ☑ Iterators are a concept to iterate anything that contains other things. Examples:
  - ☑ Values and Keys in an array
  - ☑ Textlines in a file
  - ☑ Database query results
  - ☑ Files in a directory
  - ☑ Elements or Attributes in XML
  - ☑ Bits in an image
  - ☑ Dates in a calendar range
- ☑ Iterators allow to encasulate algorythms

The basic concepts

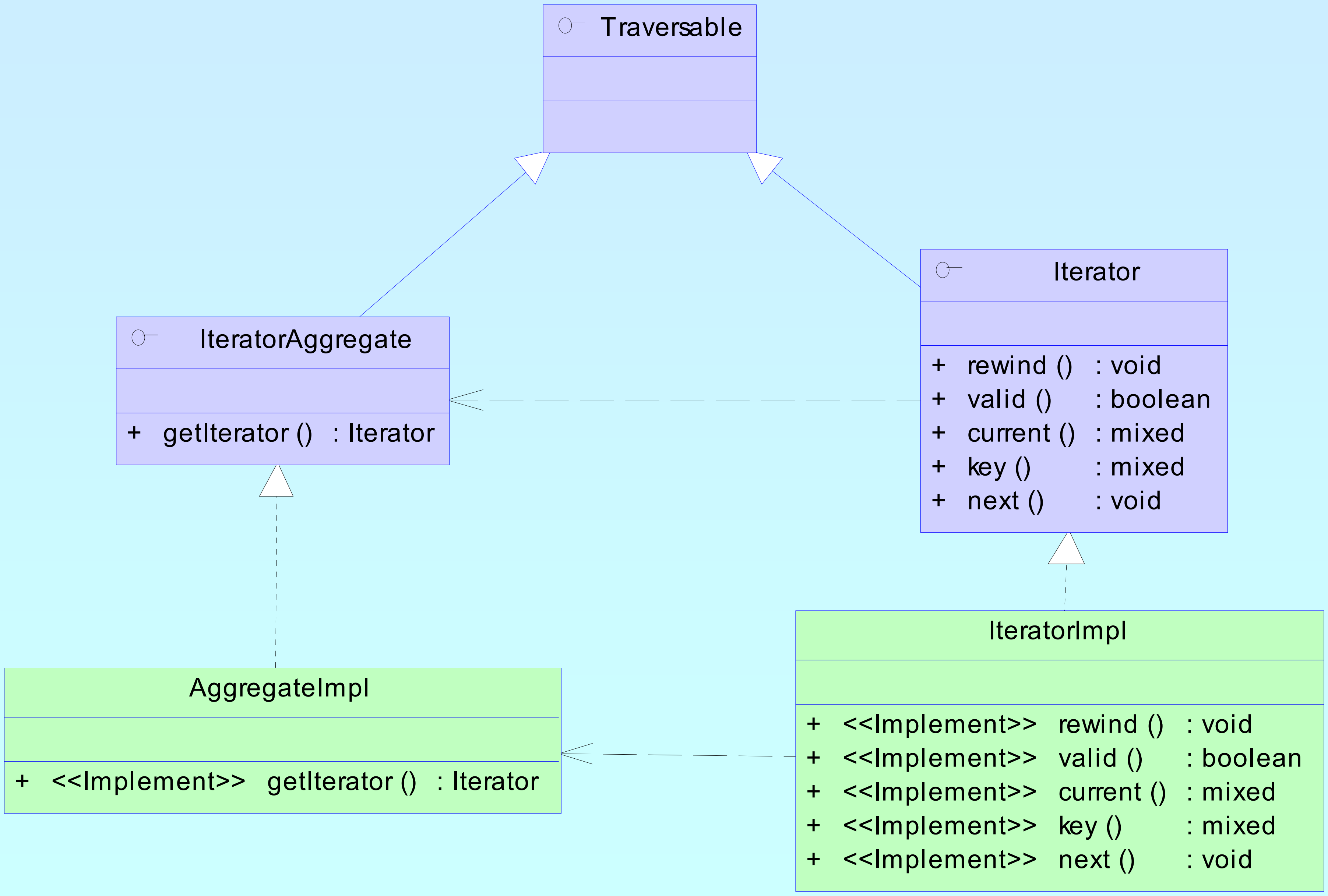
- ☑ Iterators can be internal or external also referred to as active or passive
- ☑ An internal iterator modifies the object itself
- ☑ An external iterator points to another object without modifying it
- ☑ PHP always uses external iterators at engine-level

PHP Iterators

- ☑ Anything that can be iterated implements **Traversable**
- ☑ User classes cannot implement **Traversable**
- ☑ **Aggregate** is used for objects that use external iterators
- ☑ **Iterator** is used for internal traversal or external iterators



Implementing Iterators



Traversable
IteratorAggregate
+ getIterator () : Iterator
Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
AggregateImpl
+ <<Implement>> getIterator () : Iterator
IteratorImpl
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void

How Iterators work

- ☑ Iterators can be used manually
- ☑ Iterators can be used implicitly with **foreach**

```
<?php
$o = new ArrayIterator(array(1, 2, 3));
$o->rewind();
while ($o->valid) {
    $key = $o->key();
    $val = $o->current();
    // some code
    $o->next();
}
?>
```

```
<?php
$o = new ArrayIterator(array(1, 2, 3));
foreach($o as $key => $val) {
    // some code
}
?>
```

Debug Session

```php
<?php
class ArrayIterator {
    protected $ar;
    function __construct(Array $ar) {
        $this->ar = $ar;
    }
    function rewind() {
        rewind($this->ar);
    }
    fucntion valid() {
        return !is_null(key($this->ar));
    }
    function key() {
        return key($this->ar);
    }
    fucntion current() {
        return current($this->ar);
    }
    function next() {
        next($this->ar);
    }
}
?>
```

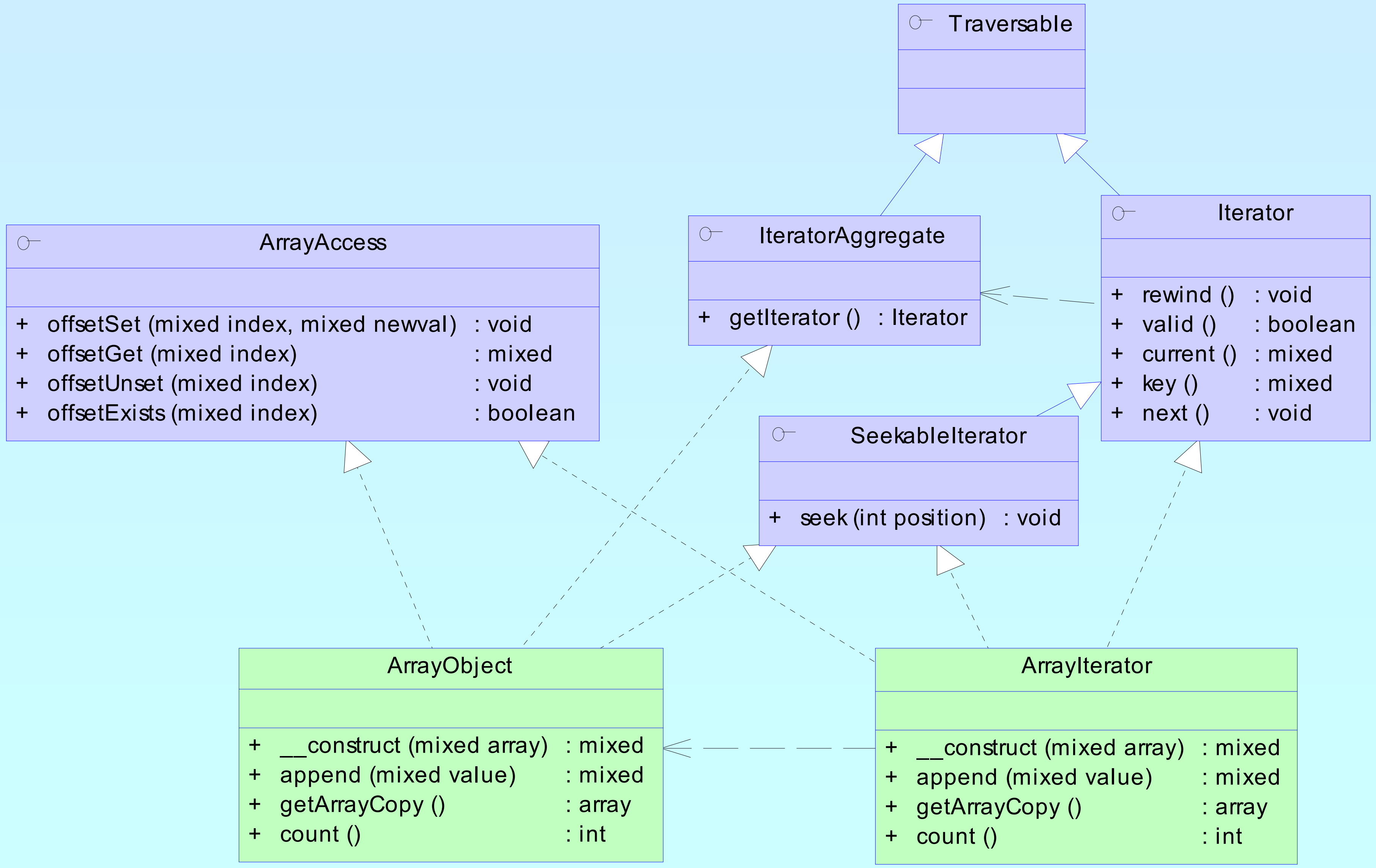
```php
<?php
$a = array(1, 2, 3);
$o = new ArrayIterator($a);
foreach($o as $key => $val) {
    echo "$key => $va\n";
}
?>
```

```
0 => 1
1 => 2
2 => 3
```

Array and property traversal

- ☑ **ArrayObject** allows external traversal of arrays and object properties
- ☑ **ArrayObject** creates **ArrayIteraor** instances for iteration
- ☑ Multiple **ArrayIterator** instances can reference the same target with different states

Array and property traversal

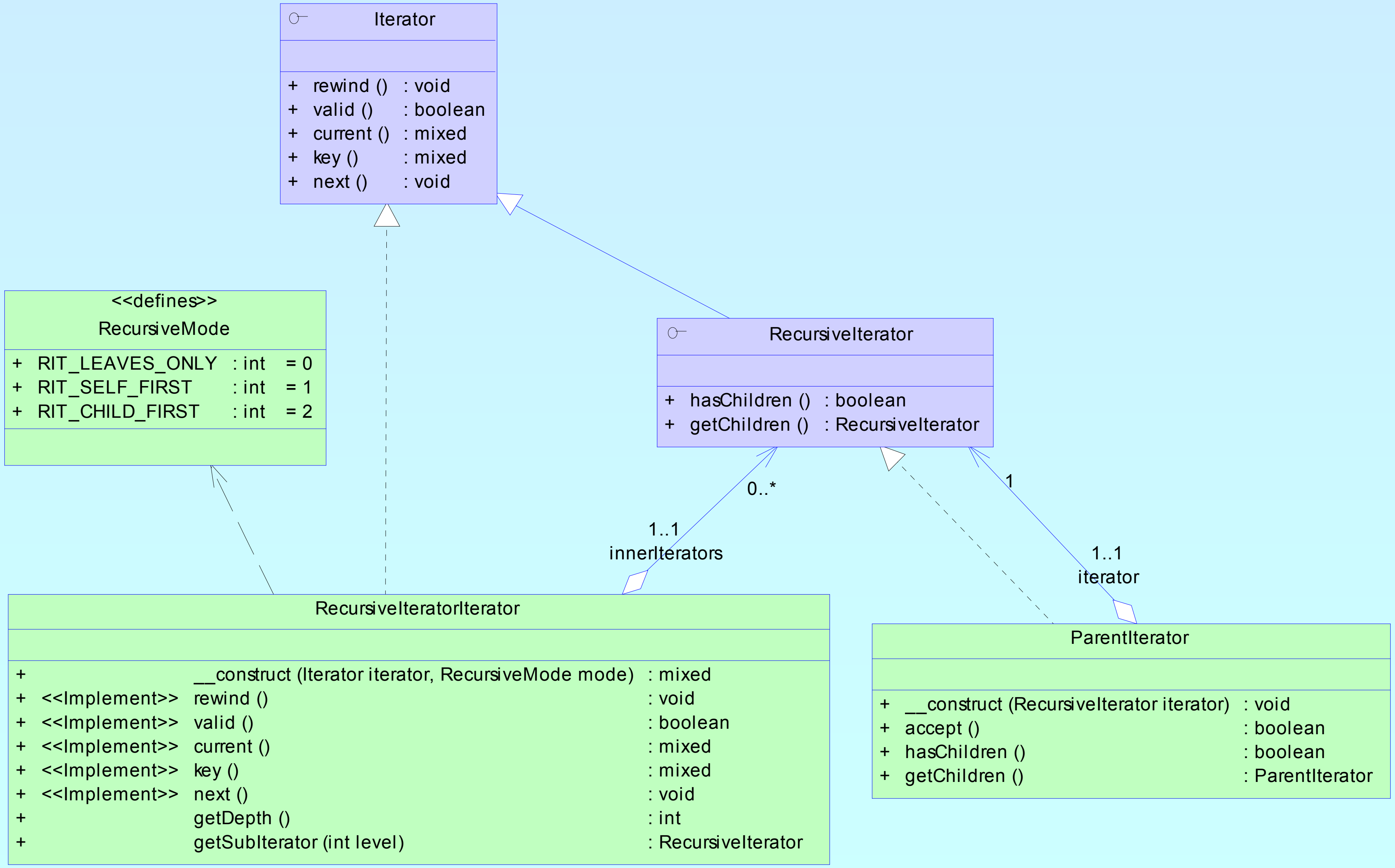

Traversable
IteratorAggregate
+ getIterator () : Iterator
Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
ArrayAccess
+ offsetSet (mixed index, mixed newval) : void
+ offsetGet (mixed index) : mixed
+ offsetUnset (mixed index) : void
+ offsetExists (mixed index) : boolean
SeekableIterator
+ seek (int position) : void
ArrayObject
+ __construct (mixed array) : mixed
+ append (mixed value) : mixed
+ getArrayCopy () : array
+ count () : int
ArrayIterator
+ __construct (mixed array) : mixed
+ append (mixed value) : mixed
+ getArrayCopy () : array
+ count () : int

Recursive traversal

- ☑ Some data may be recursively traversable
- ☑ Interface **RecursiveIterator** tells when
- ☑ Class **RecursiveIteratorIterator** use it

- ☑ Examples:
  - ☑ Arrays
  - ☑ XML data
  - ☑ Directories

Recursive traversal



Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
<<defines>>
RecursiveMode
+ RIT_LEAVES_ONLY : int = 0
+ RIT_SELF_FIRST : int = 1
+ RIT_CHILD_FIRST : int = 2
RecursiveIterator
+ hasChildren () : boolean
+ getChildren () : RecursiveIterator
0..*
1..1
innerIterators
1
1..1
iterator
RecursiveIteratorIterator
+ __construct (Iterator iterator, RecursiveMode mode) : mixed
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void
+ getDepth () : int
+ getSubIterator (int level) : RecursiveIterator
ParentIterator
+ __construct (RecursiveIterator iterator) : void
+ accept () : boolean
+ hasChildren () : boolean
+ getChildren () : ParentIterator

Filtering values

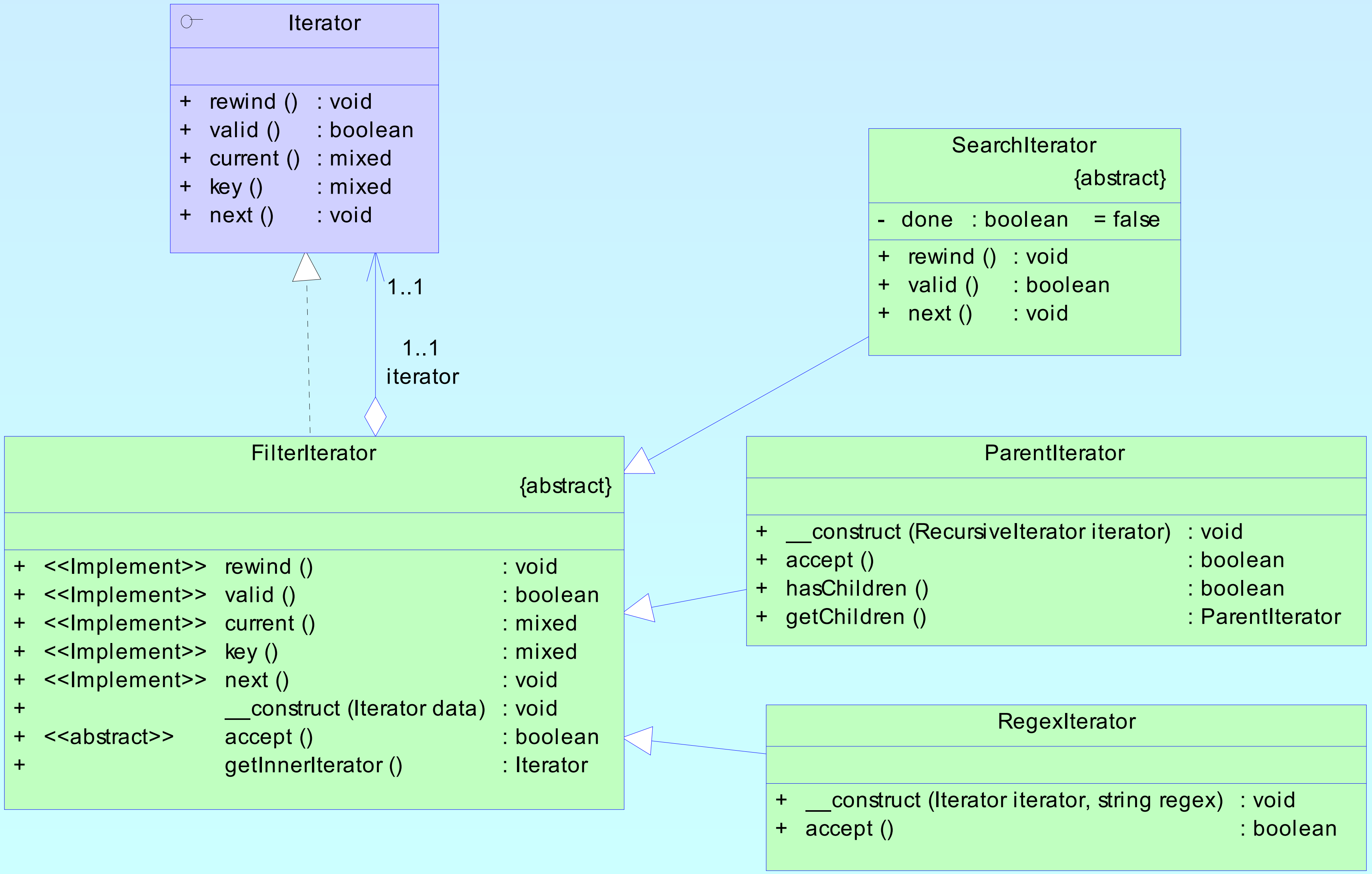
- **FilterIterator** allows to filter data

  Compareable to SQL **WHERE** clauses

  - **FilterIterator::__construct** takes any **Iterator**
  - **FilterIterator**::**accept** needs to be implemented

- Specializations:
  - **SearchIterator** stops at the first accepted value
  - **ParentIterator** only accepts values which have childs

Filtering values


Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
1..1
1..1
iterator
SearchIterator
{abstract}
- done : boolean = false
+ rewind () : void
+ valid () : boolean
+ next () : void
FilterIterator
{abstract}
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void
+ __construct (Iterator data) : void
+ <<abstract>> accept () : boolean
+ getInnerIterator () : Iterator
ParentIterator
+ __construct (RecursiveIterator iterator) : void
+ accept () : boolean
+ hasChildren () : boolean
+ getChildren () : ParentIterator
RegexIterator
+ __construct (Iterator iterator, string regex) : void
+ accept () : boolean

Limiting values

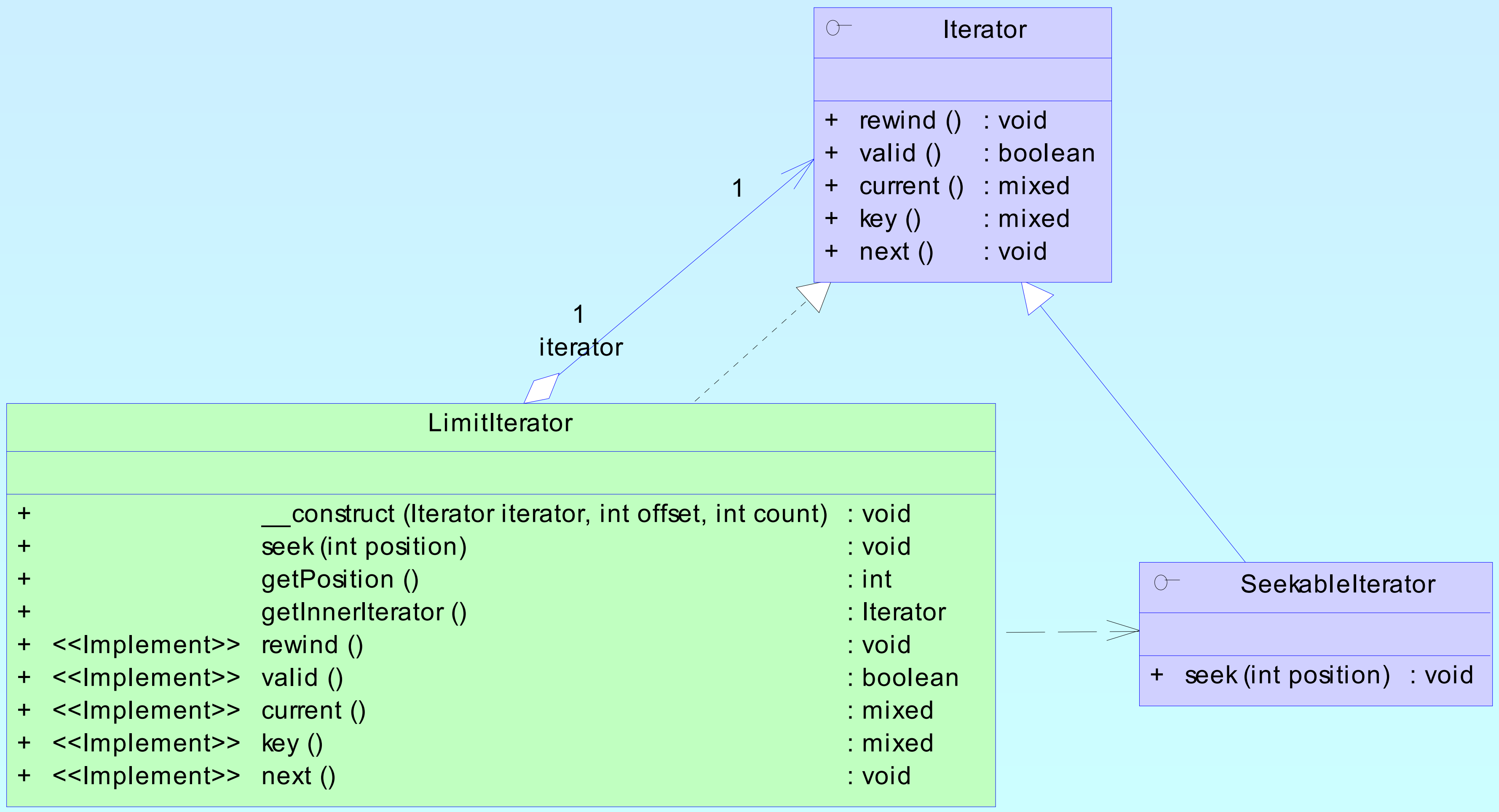
☑ **LimitIterator** allows to limit the returned values

Compareable to **LIMIT** of some SQL dialects

- ☑ You can specify the start offset
- ☑ You can specify the number of returned values

- ☑ When the inner Iterator is a **SeekableIterator** then method seek will be used. Otherwise seek operation will be manually.

Limiting values


Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
1
1
iterator
LimitIterator
+ __construct (Iterator iterator, int offset, int count) : void
+ seek (int position) : void
+ getPosition () : int
+ getInnerIterator () : Iterator
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void
SeekableIterator
+ seek (int position) : void

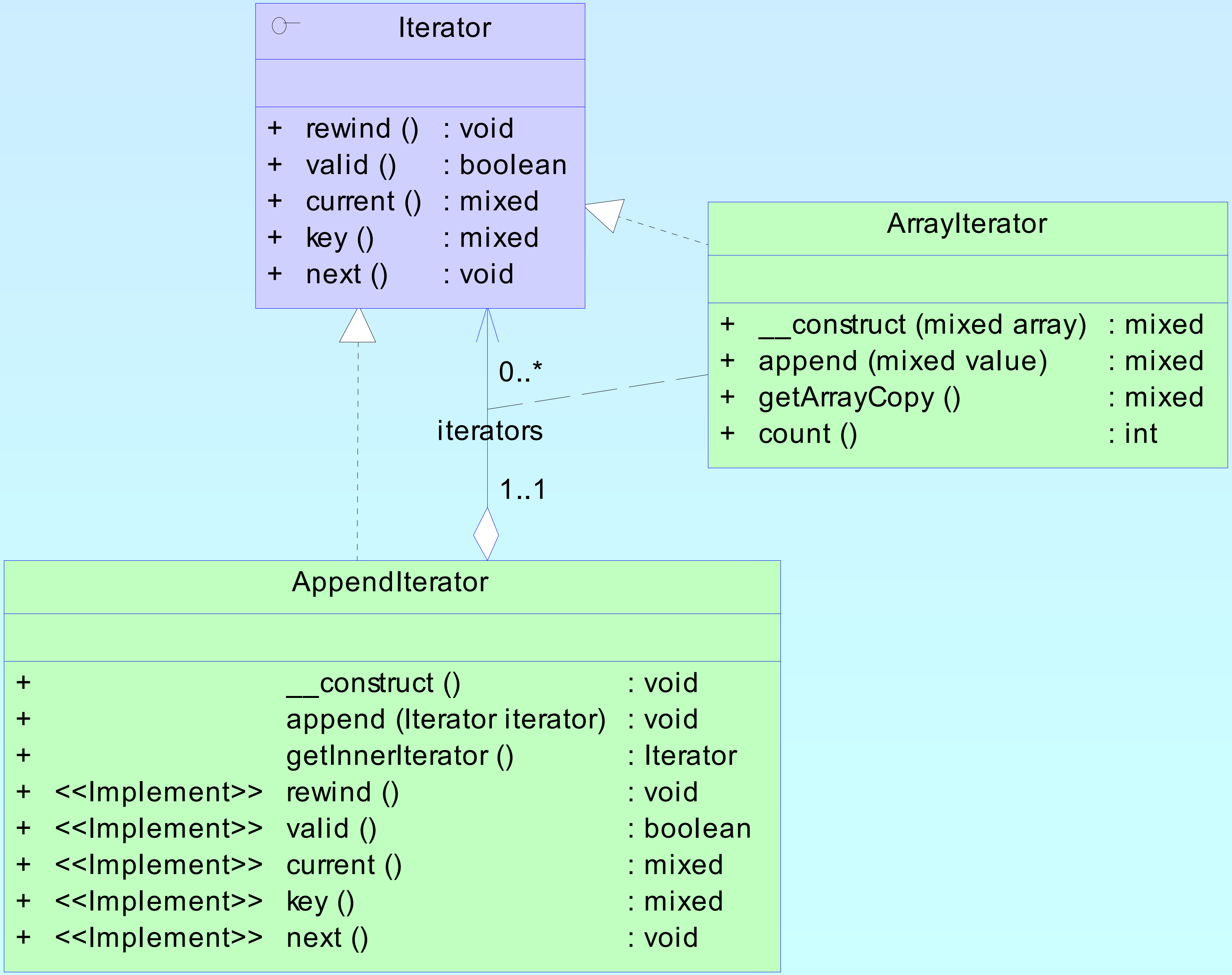
Appending Iterators

- ☑ **AppendIterator** allows to concatenate Iterators

  Compareable to SQL clause **UNION**

  - ☑ Uses a private **ArrayIterator** to store Iterators
  - ☑ **AppendIterator::append()**
    - ☑ allows to append iterators
    - ☑ does not call the rewind()
    - ☑ if $this is invalid $this will move to the appended iterator

Appending Iterators



Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
ArrayIterator
+ __construct (mixed array) : mixed
+ append (mixed value) : mixed
+ getArrayCopy () : mixed
+ count () : int
0..*
iterators
1..1
AppendIterator
+ __construct () : void
+ append (Iterator iterator) : void
+ getInnerIterator () : Iterator
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void

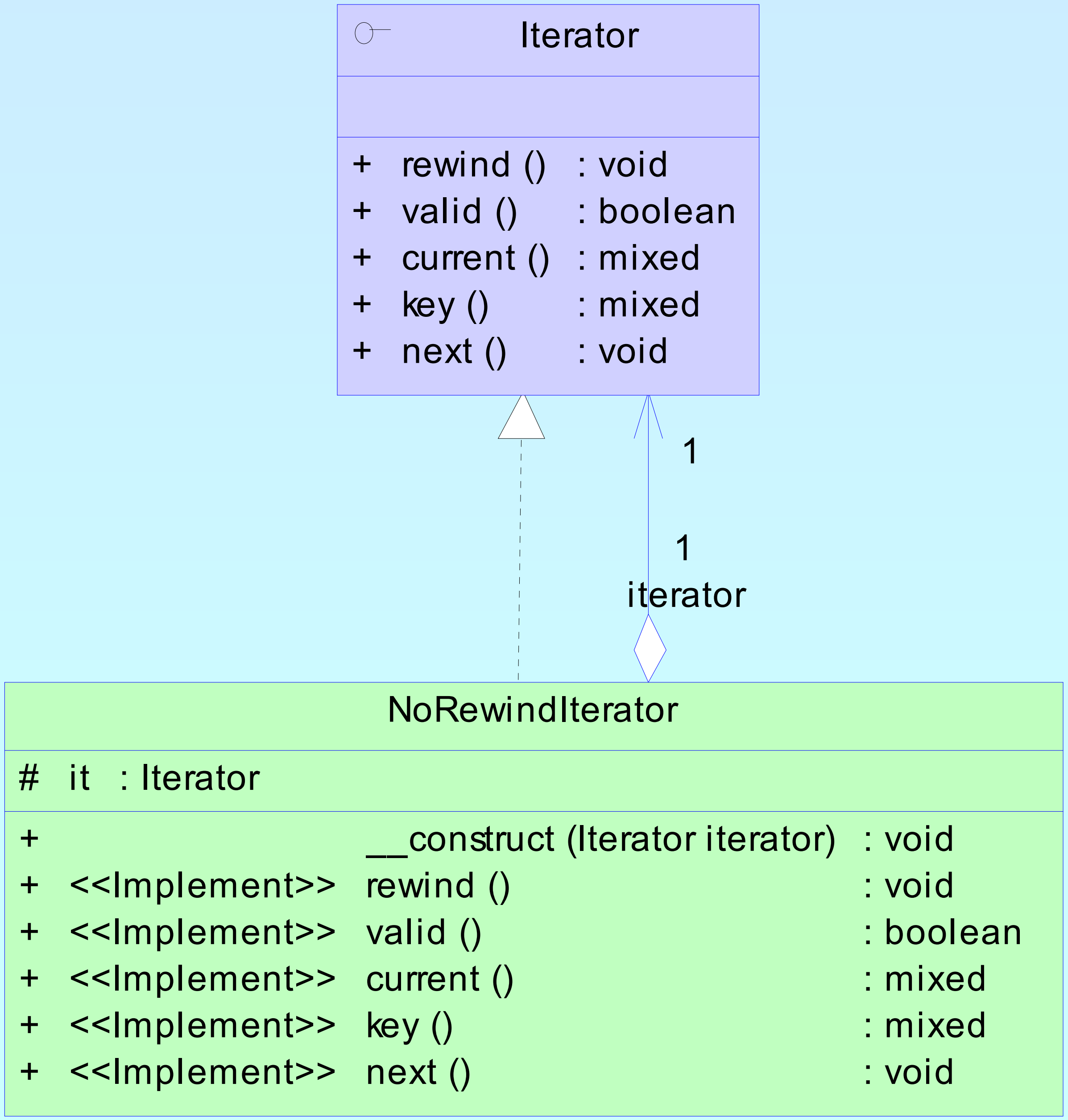
Getting rid of rewind

- ☑ **NoRewindIterator** allows to omit rewind calls

This is especially helpful when appending with

- ☑ **ArrayObject::append**()
- ☑ **ArrayIterator::append**()
- ☑ **Appenditerator::append**()

Getting rid of rewind()

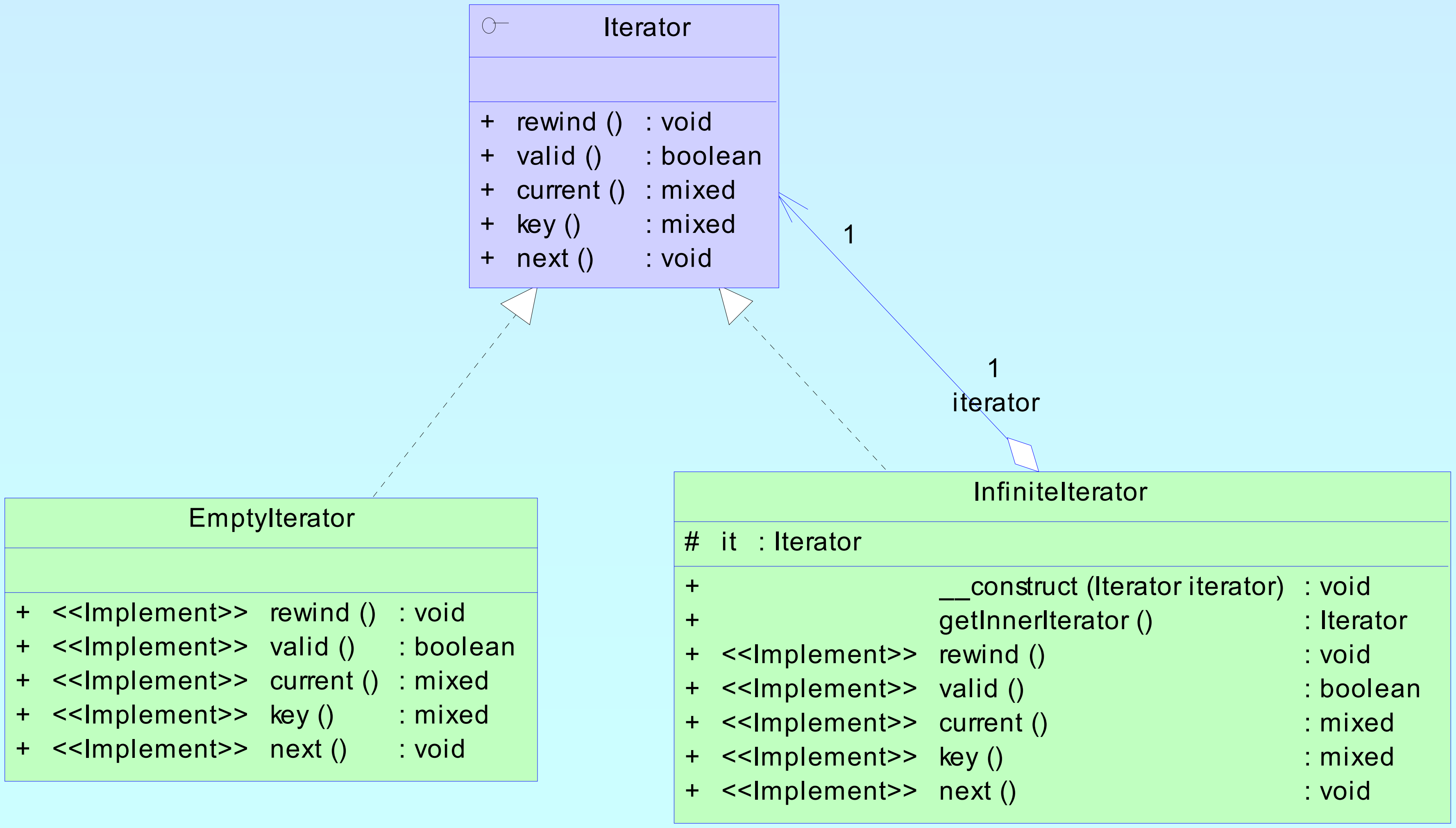

Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
1
1
iterator
NoRewindIterator
# it : Iterator
+ __construct (Iterator iterator) : void
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void

Vacuity & Infinity

Sometimes it is helpful to have

- ☑ **EmptyIterator** as a placeholder for no data
- ☑ **InfiniteIterator** to endlessly repeat data in an iterator

Vacuity & Infinity



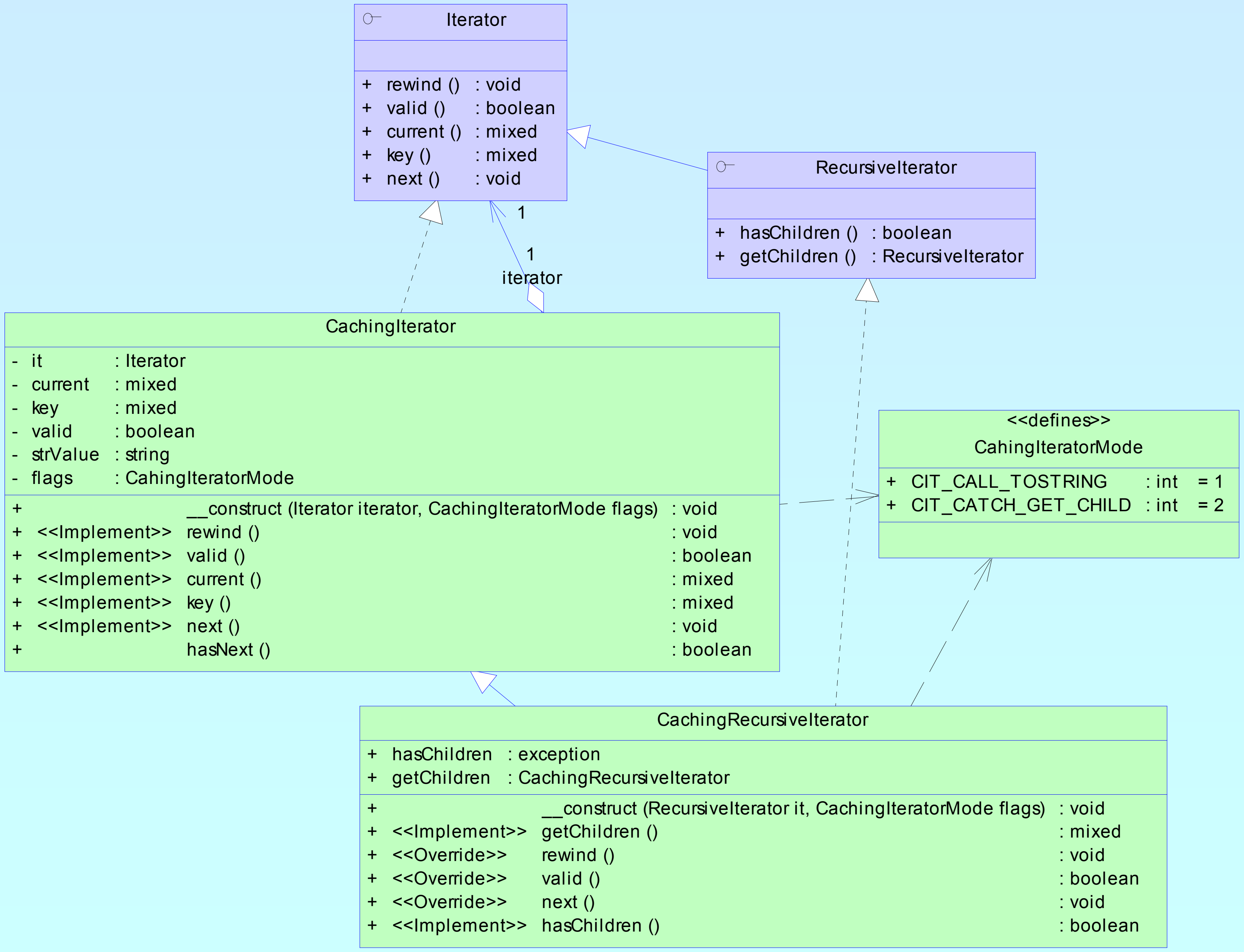
Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
1
1
iterator
EmptyIterator
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void
InfiniteIterator
# it : Iterator
+ __construct (Iterator iterator) : void
+ getInnerIterator () : Iterator
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void

hasNext ?

- ☑ **CachingIterator** caches the current element
  - ☑ This allows to know whether one more value exists
- ☑ **CachingRecursiveIterator** does this recursively
  - ☑ This allows to draw tree graphics

```
marcus@frodo /usr/src/php-cvs $ php ext/spl/examples/tree.php ext/spl
ext/spl
|-CVS
|-examples
| |-CVS
| \-tests
|   \-CVS
\-tests
  \-CVS
```

hasNext ?


Iterator
+ rewind () : void
+ valid () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
RecursiveIterator
+ hasChildren () : boolean
+ getChildren () : RecursiveIterator
1
1
iterator
CachingIterator
- it : Iterator
- current : mixed
- key : mixed
- valid : boolean
- strValue : string
- flags : CahingIteratorMode
+ __construct (Iterator iterator, CachingIteratorMode flags) : void
+ <<Implement>> rewind () : void
+ <<Implement>> valid () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void
+ hasNext () : boolean
<<defines>>
CahingIteratorMode
+ CIT_CALL_TOSTRING : int = 1
+ CIT_CATCH_GET_CHILD : int = 2
CachingRecursiveIterator
+ hasChildren : exception
+ getChildren : CachingRecursiveIterator
+ __construct (RecursiveIterator it, CachingIteratorMode flags) : void
+ <<Implement>> getChildren () : mixed
+ <<Override>> rewind () : void
+ <<Override>> valid () : boolean
+ <<Override>> next () : void
+ <<Implement>> hasChildren () : boolean

References

☑ Documentation and Sources to PHP5
  http://php.net

☑ Documentation to ext/spl
  http://cvs.php.net/co.php/php-src/ext/spl/spl.php?r=HEAD
  http://somabo.de/php/ext/spl/html/

☑ Sourcecode for examples
  ext/spl/examples

☑ These slides
  http://somabo.de/talks/