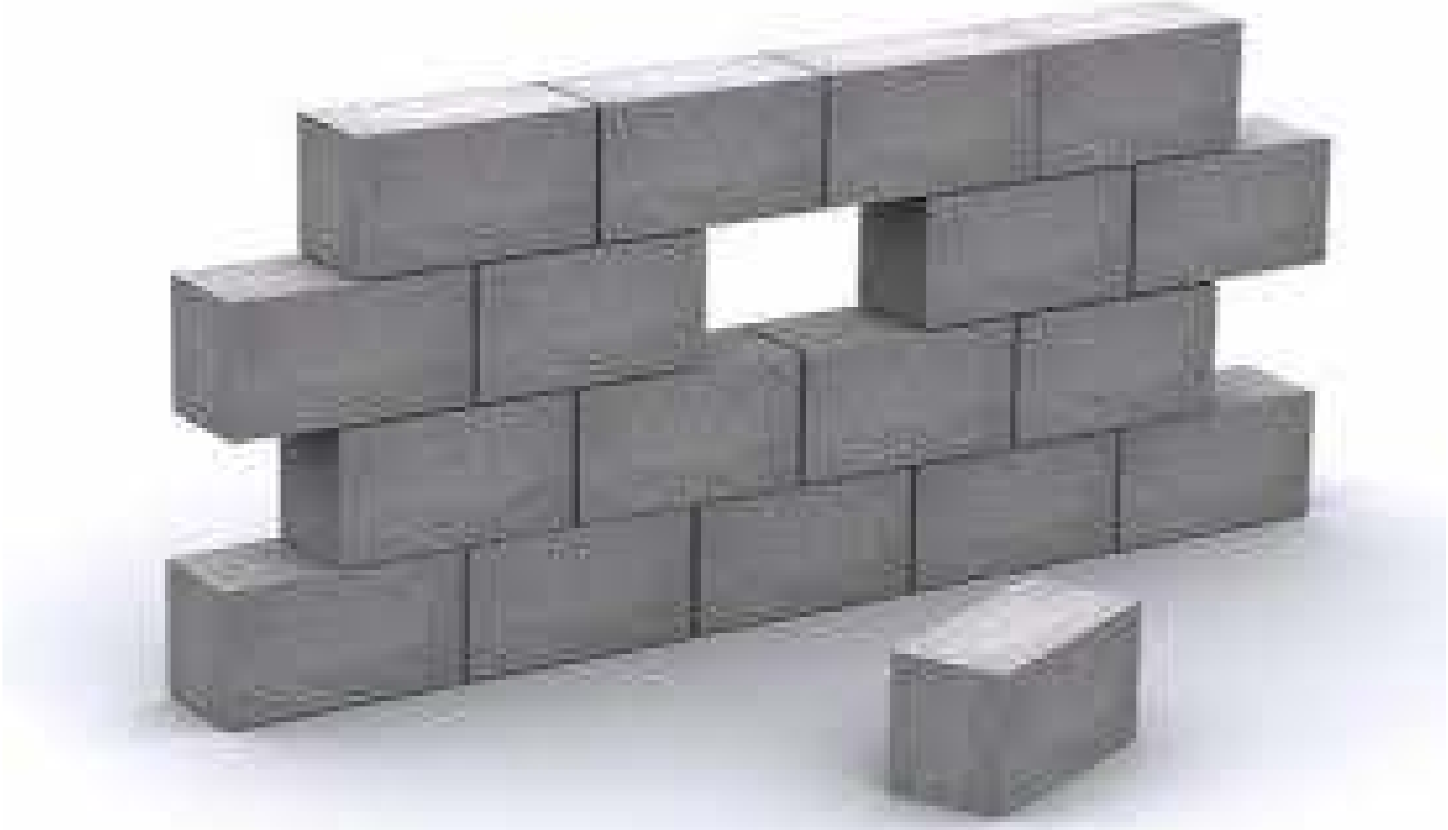


UNICORN Education

JAVA COLLECTIONS

leto 2017

Jaro Jakubík

AGENDA

- Code conventions
- Collections API
  - Example

CODE CONVENTIONS

CLASS DECLARATIONS

- class/interface documentation comment
  - /** …. */
- class/interface statement
- class/interface implementation comment
  - /* …. */
- class static variables
- instance variables
- constructors
- methods

LINES OF CODE

- Avoid lines longer than 80 characters
- When an expresion not fit on single line
  - Break after a comma.
  - Break before an operator.
  - Align new line with the begining of the expression at the same level.
  - Use TAB for code leveling.

COMMENTS

- Implementation comments
  - //
  - /* …. */
- Documentation comments
  - /** …. */
- Frequency of comments sometimes reflects poor quality of code When you feel compelled to add a comment, consider rewriting the code to make it clearer.
- Comments should not be enclosed in large boxes drawn with asterisks or other characters.
- Comments should never include special characters such as form-feed and backspace.

DECLARATIONS

- One declaration per line is recommended.
- No space between a method name and the parenthesis "(" starting its parameter list.
- Open brace "{" appears at the end of the same line as the declaration statement.
- Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{".
- Methods are separated by a blank line.

STATEMENTS

- Each line should contain at most one statement.
- A return statement with a value should not use parentheses unless they make the return value more obvious in some way.
- Avoid the following error-prone IF form:

```
if(condition)
    statements
```

- for, while, do-while, switch, try-catch-finally

NAMING CONVENTIONS (1/2)

- Packages
  - The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names.
  - Subsequent components of the package name vary according to an organization's own internal naming conventions.
- Classes
  - Class names should be nouns, in mixed case with the first letter of each internal word capitalized.
  - Try to keep your class names simple and descriptive.
  - Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

NAMING CONVENTIONS (2/2)

- Interfaces
    - Interface names should be capitalized like class names.
- Methods
    - Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.
- Variables
    - In mixed case with a lowercase first letter. Internal words start with capital letters.
    - Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.
    - Variable names should be short yet meaningful.
- Constants
    - The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_").

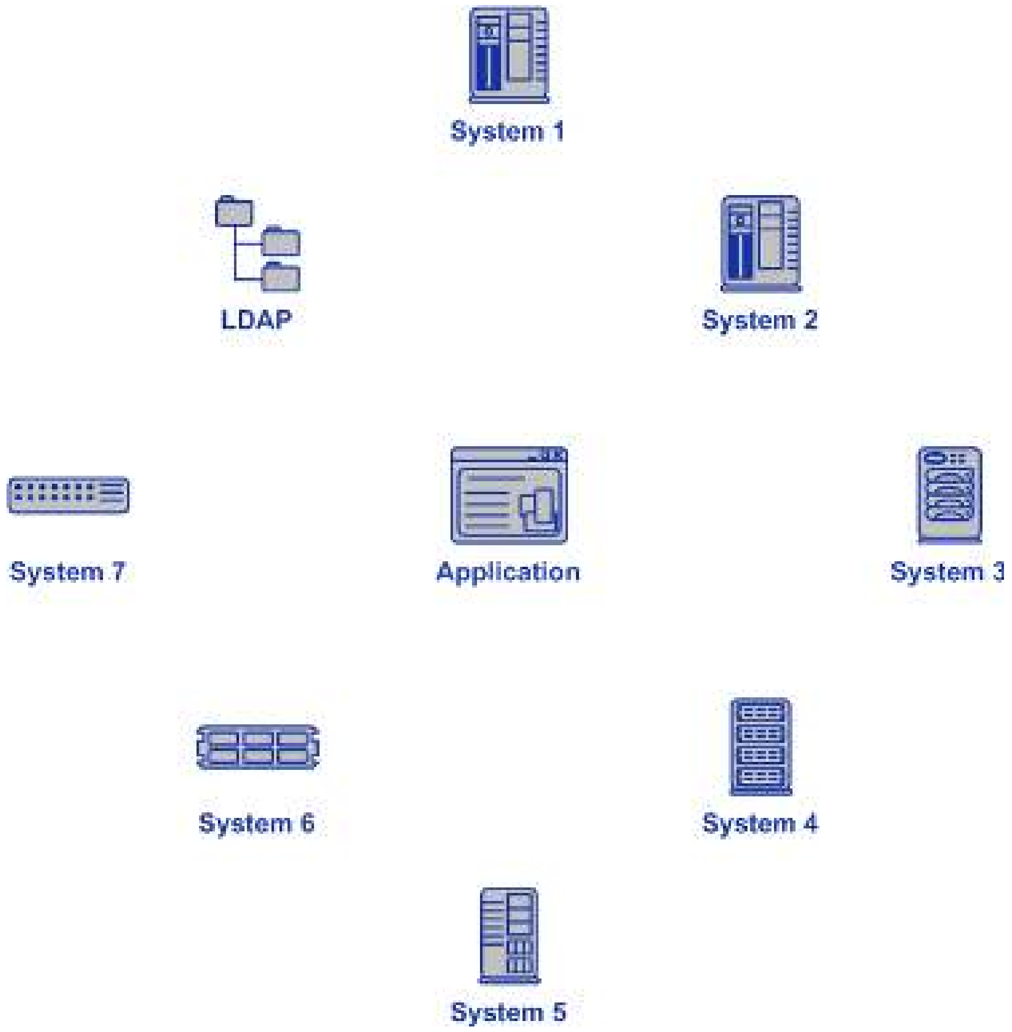
PROGRAMMING PRACTICES

- Don't make any instance or class variable public without good reason.
- Avoid using an object to access a class (static) variable or method.
- Avoid assigning several variables to the same value in a single statement. It is hard to read.

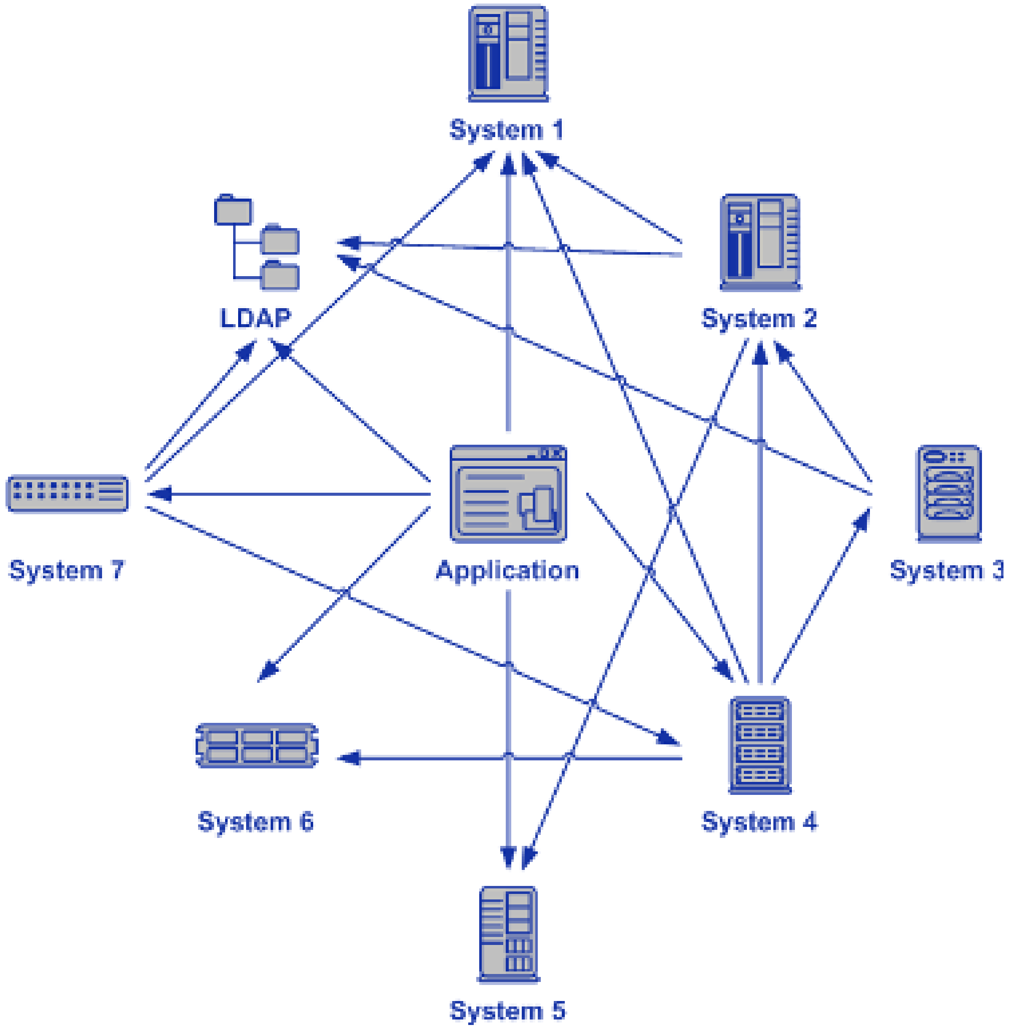
- April 20, 1999
- http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

Programovanie voči rozhraniam

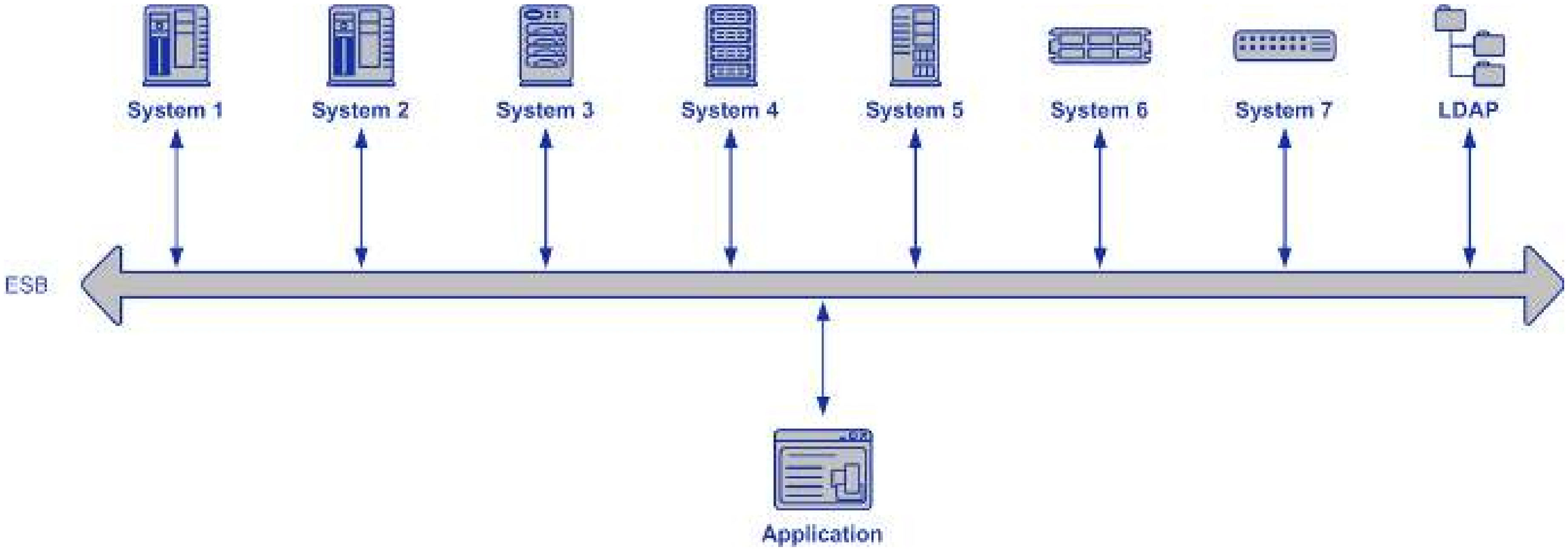
Integrovaný systém - 1



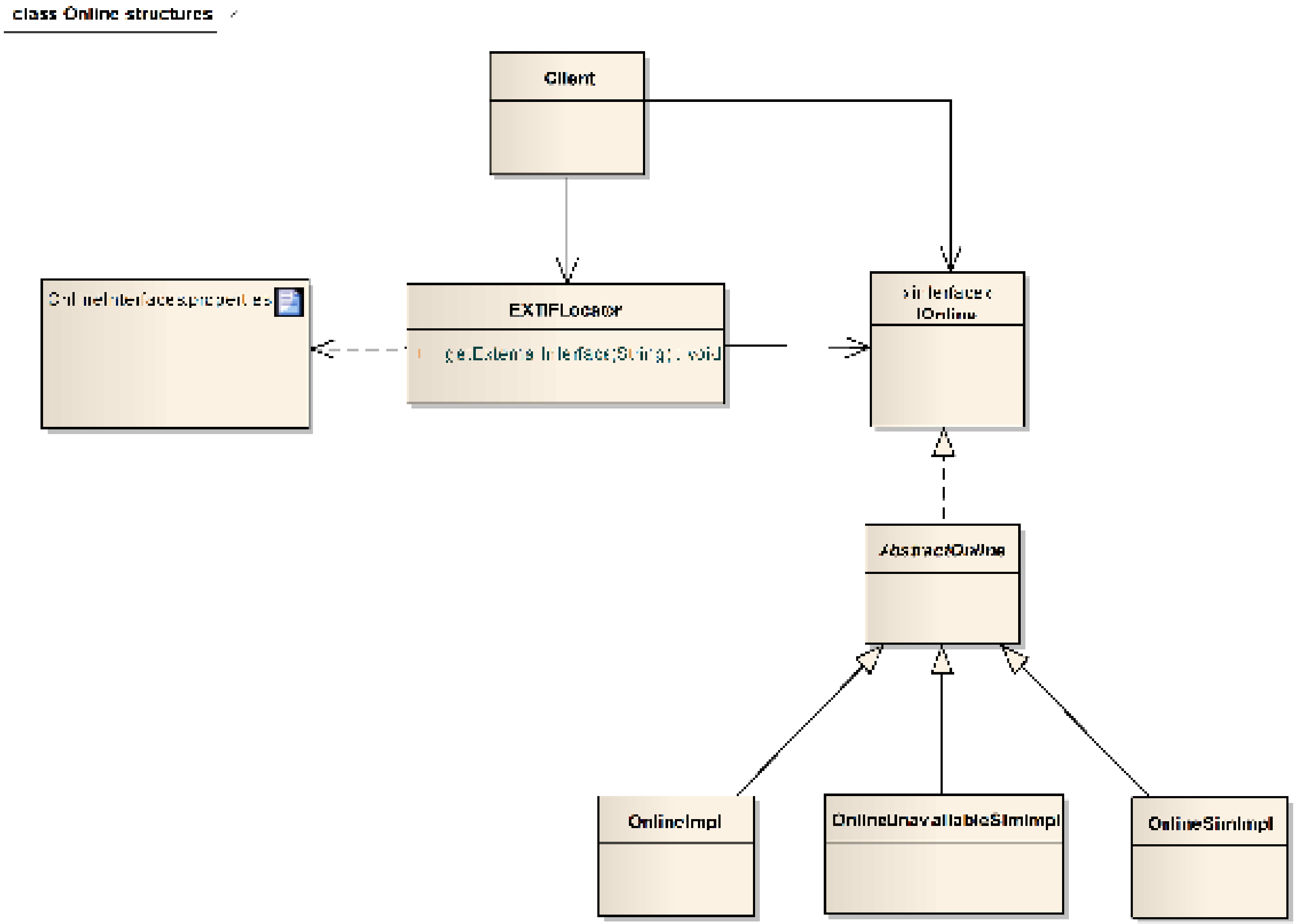
Integrovaný systém - 2



Integrovaný systém - 3



Factory method



COLLECTIONS API

INTRODUCTION

- Collection is simply an object that groups multiple elements into a single unit.
- Collections are used to store, retrieve, manipulate, and communicate aggregate data.

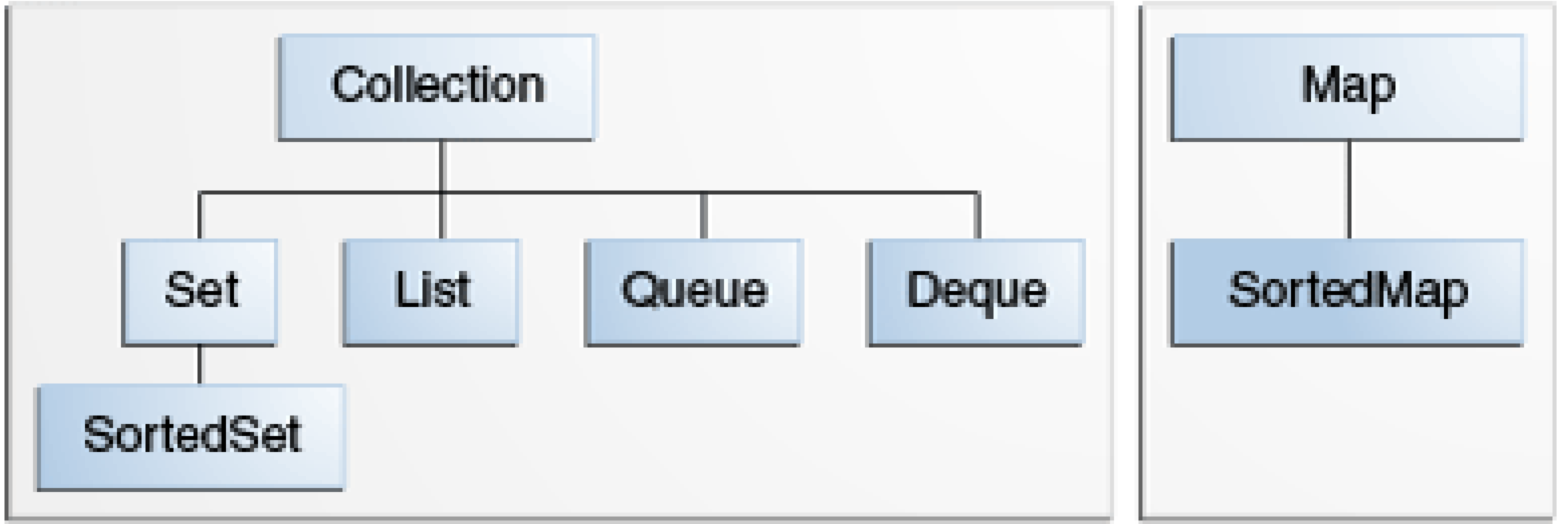
- A collections framework is a unified architecture for representing and manipulating collections.
  - Interface
  - Implementations
  - Algorithms

- C++ Standard Template Library (STL)
- Smalltaks collection hierarchy

FRAMEWORK BENEFITS

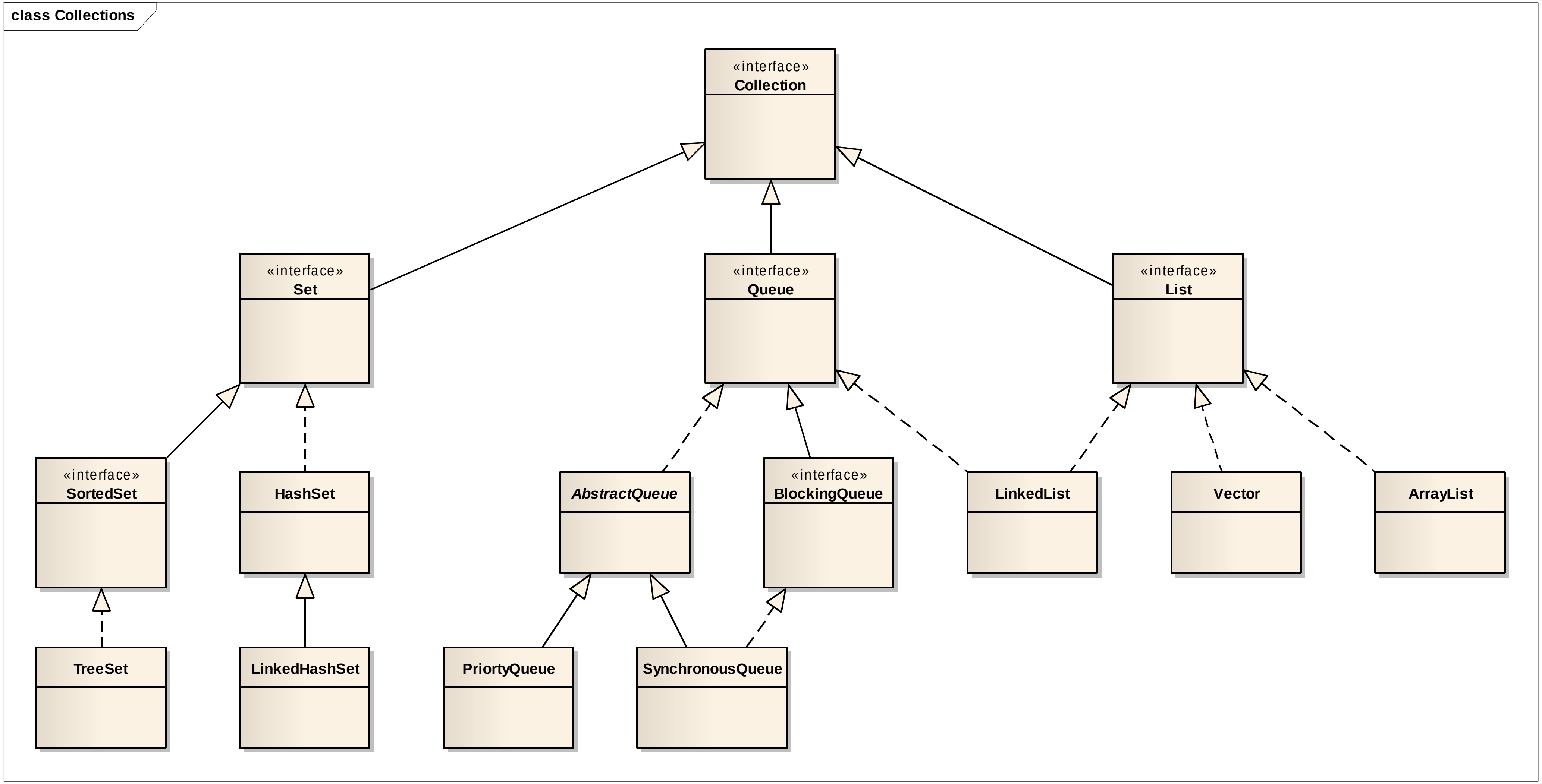
- Reduces programming effort.
- Increases program speed and quality.
- Reduces effort to learn and to use new APIs.
- Reduces effort to design new APIs.
- Fosters software reuse.

INTERFACES

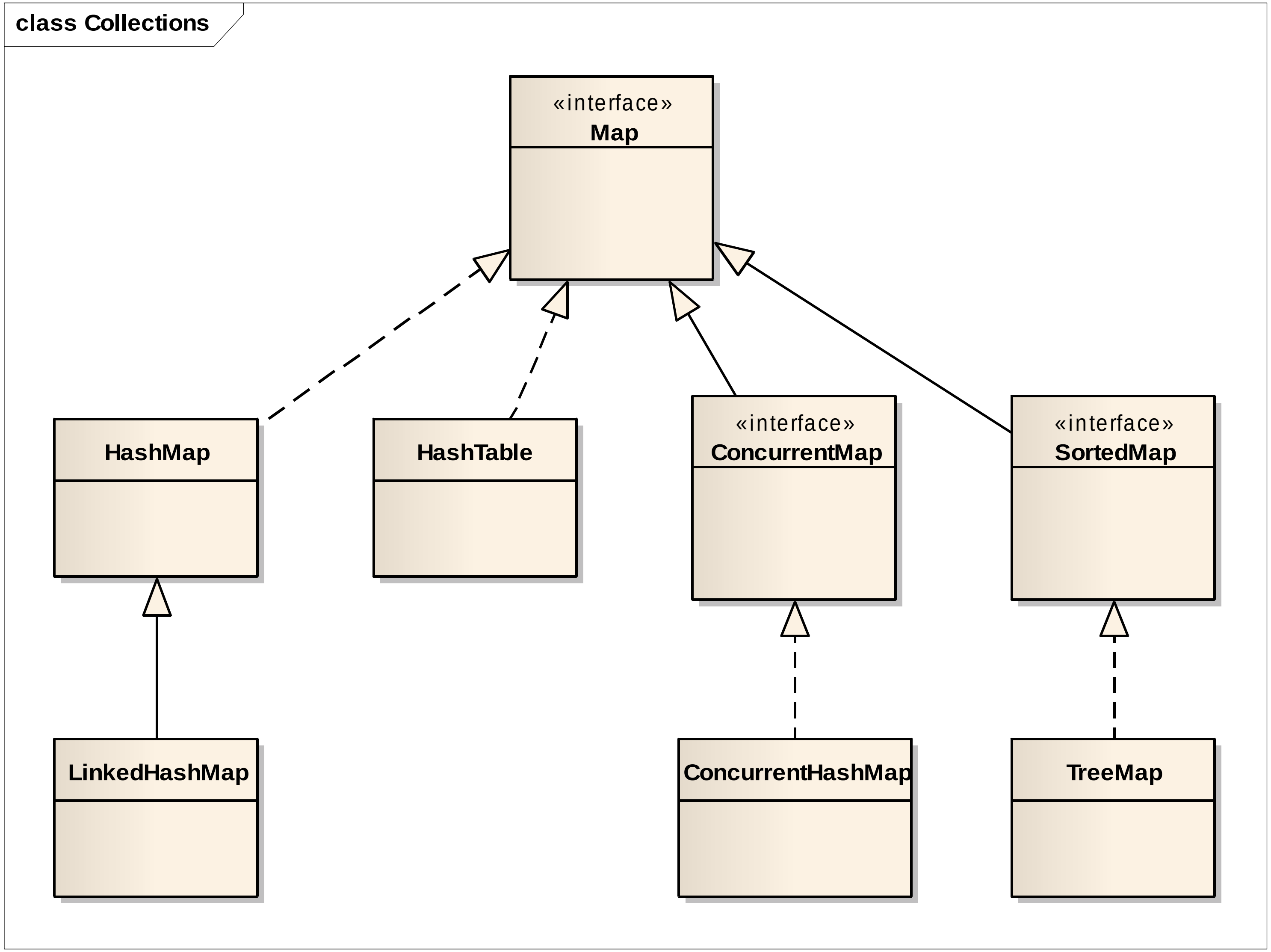


- These interfaces (Collection and all extended) allow collections to be manipulated independently of the details of their representation.

COLLECTIONS API (1/2)



COLLECTIONS API (2/2)



COLLECTION INTERFACE

- The root of the collection hierarchy.
- A collection represents a group of objects known as its elements.
- See javadoc
  - http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

LIST INTERFACE

- An ordered collection.
- Lists can contain duplicate elements.

LIST IMPLEMENTATIONS

- LinkedList
  - Doubly linked list implementation
  - Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.
- ArrayList
  - Array
  - Resizeable

QUEUE

- Queues typically, but not necessarily, order elements in a FIFO manner. Among the exceptions are priority queues, which order elements according to their values.
- Queue methods
  - add – offer
  - remove – poll
  - element – peek
- Queue implementations generally do not allow insertion of null elements. The LinkedList implementation, which was retrofitted to implement Queue, is an exception.
- LinkedList, PriorityQueue

DEQUE INTERFACE

- Usually pronounced as deck, a deque is a double-ended-queue.
- A double-ended-queue is a linear collection of elements that supports the insertion and removal of elements at both end points.
- The Deque interface is a richer abstract data type than both Stack and Queue because it implements both stacks and queues at the same time.

DEQUE IMPLEMENTATIONS

- General purpose Deque implementations
  - LinkedList
  - ArrayDequeu
- The ArrayDeque class is the resizable array implementation of the Deque interface, whereas the LinkedList class is the list implementation.
- The LinkedList implementation is more flexible than the ArrayDeque implementation. LinkedList implements all optional list operations.
- Null elements are allowed in the LinkedList implementation but not in the ArrayDeque implementation.

SET INTERFACE

- A Set is a Collection that cannot contain duplicate elements.
- It models the mathematical set abstraction.
- The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
- Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.
  - Two Set instances are equal if they contain the same elements.

SET IMPLEMENTATIONS

- There are three general-purpose Set implementations
  - HashSet,
  - TreeSet,
  - LinkedHashSet.
- HashSet is much faster than TreeSet (constant-time versus log-time for most operations) but offers no ordering guarantees.
- If you need to use the operations in the SortedSet interface, or if value-ordered iteration is required, use TreeSet.
- It's a fair bet that you'll end up using HashSet most of the time.
- LinkedHashSet is in some sense intermediate between HashSet and TreeSet. Implemented as a hash table with a linked list running through it, it provides insertion-ordered iteration (least recently inserted to most recently) and runs nearly as fast as HashSet.

MAP INTERFACE

- Map as a collection
  - keySet — the Set of keys contained in the Map
  - values — The Collection of values contained in the Map.
    - This Collection is not a Set, because multiple keys can map to the same value.
  - entrySet — the Set of key-value pairs contained in the Map.
    - The Map interface provides a small nested interface called Map.Entry, the type of the elements in this Set.

MAP IMPLEMENTATIONS

- General purpose Map implementations
  - HashMap
  - TreeMap
  - LinkedHashMap
- If you need SortedMap operations or key-ordered Collection-view iteration, use TreeMap.
- If you want maximum speed and don't care about iteration order, use HashMap.
- If you want near-HashMap performance and insertion-order iteration, use LinkedHashMap.

CONVERSION CONSTRUCTOR

- All general-purpose collection implementations have a constructor that takes a Collection argument.
- This constructor initializes the new collection to contain all of the elements in the specified collection.

- Suppose, for example, that you have a Collection<String> c, which may be a List, a Set, or another kind of Collection.

```
List<String> list = new ArrayList<String>(c);
```

TRAVERSING COLLECTIONS (1/2)

- For-each construct
  - The for-each construct allows you to concisely traverse a collection or array using a for loop.

```
for (Object o : collection) {
	System.out.println(o);
}
```

TRAVERSING COLLECTIONS (2/2)

- Iterators
  - An Iterator is an object that enables you to traverse through a collection.

```
Iterator<Object> i = collection.iterator();
while(i.hasNext()) {
        Object o  = i.next();
        System.out.println(o);
}
```

ORDERING

```
List<?> l;
Collections.sort(l);
```

- If the List consists of String elements, it will be sorted into alphabetical order. If it consists of Date elements, it will be sorted into chronological order.
- String and Date both implement the Comparable interface. Comparable implementations provide a natural ordering for a class, which allows objects of that class to be sorted automatically.

COMPARABLE INTERFACE

- The compareTo method compares the receiving object with the specified object and returns:
  - a negative integer - the receiving object is less than the specified object,
  - 0 - the receiving object is equal to the specified object,
  - a positive integer - the receiving object is greater than the specified object.
- If the specified object cannot be compared to the receiving object, the method throws a ClassCastException.

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

COMPARATOR INTERFACE

- Ordering in order other than natural ordering.
- Ordering objects that don't implement Comparable interface.
- The compare method compares its two arguments, returning:
  - a negative integer - first argument is less than the second,
  - 0 - first argument is equal to the second,
  - a positive integer - first argument is greater than the second.
- If either of the arguments has an inappropriate type for the Comparator, the compare method throws a ClassCastException.

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

EXAMPLE

- Ukázať
  - Príklad vytvorenia inštancie listu

- Performance test pre pracu s vela prvkami v ArrayList a v LinkedList

- Príklad usporiadania zoznamu Objektov cez
  - Natural ordering pre Double (ako tam je implementované Comparable?)
  - implementovaním Comparable pre štruktúru
  - Implementovaním Comparator pre štruktúru

UNICORN Education

education