

# Vývoj Aplikácií s Viacvrstvovou Architektúrou

## 04. Kolekcie



# Čo nás čaká a neminie...

## 1. časť

Úvod do Javy

Štruktúra platformy

Vývojové technológie

Kolekcie

Logovanie

Lokalizácia

## 2. časť

XML, IO

Regulárne výrazy

Modularita

JDBC

Bezpečnosť

Prehľad JEE a .NET

# Čo nás čaká a nemenie...

1. časť

Architektúra

2. časť

Java

Best practices | Faily | Fuckupy

Súbor Správa Vývojár Pomocník Povedzte, čo chcete urobiť

Odstrániť Archivovať Premiestniť Odpovedať všetkým Odpovedať Hľadať Označiť ako neprečítané Čítať nahlas Imerzná čítačka Lupa

Preposlat' Hľadať Čítať nahlas Imerzná čítačka Lupa

...

## Toolkit Tuesday: A Best Practice for Developing a MOSA-based Reference Architecture

TO

The Open Group <donotreply@opengroup.org>



Odpovedať

Odpovedať všetkým

Preposlat'

st 16. 2. 2022 21:26

Ak sa vyskytnú problémy so zobrazením tejto správy, kliknutím sem ju zobrazte vo webovom prehliadači.



Dear Miroslav,

In the next episode of the [Architect's Toolkit Tuesday Broadcast Series](#), we are talking about Modular Open Systems Approach (MOSA).

Join us on **Tuesday, February 22, 2022**, at 8:30 am PST, 4:30 pm GMT when Dr. Steven Davidson, Chief Scientist for Systems Architecture Electronic Systems Innovation Center, The MITRE Corporation will be talking about **A Best Practice for Developing a MOSA-based Reference Architecture**.



Súbor Správa Vývojár Pomocník Povedzte, čo chcete urobiť

Odstrániť Archivovať Premiestniť Odpovedať všetkým Odpovedať Preposlať Označiť ako neprečítané Hľadať Čítať nahlas Imerzná čítačka Lupa ...

## You are invited - Architecture & Migration Insights for new ISVs



Microsoft Slovakia <Microsoft@e-mail.microsoft.com>  
Komu riaditel@it-academy.sk

Like Odpovedať Odpovedať všetkým Preposlať ...

ut 8. 3. 2022 10:01



Hello,

If you are an Independent Software Vendor who considers starting to cooperate with Microsoft or a new Microsoft ISV Partner and you are interested in key aspects of designing a cloud solution versus on-premise, please join us to a short webinar, where we will give the overview off:

- Baseline and pain points of Non-Cloud solutions
- Azure Cloud benefits
- Biggest myths about Azure Cloud
- A way to cloud for a frequent On-Premise architecture
- Briefly SQL and Non-SQL data in Azure Cloud
- Mapping of key application layers to Cloud platform services
- How to extend the value of your solution
- Additional resources and next steps

By the end of the webinar, you should get an overview of useful hints which will help you with essential decisions regarding the core of your solution's architecture.

**Date:** March 10<sup>th</sup> 2022

**Time:** 10:00 -10:50 CET

**Prečo by mal každý programátor ovládať kolekcie?**

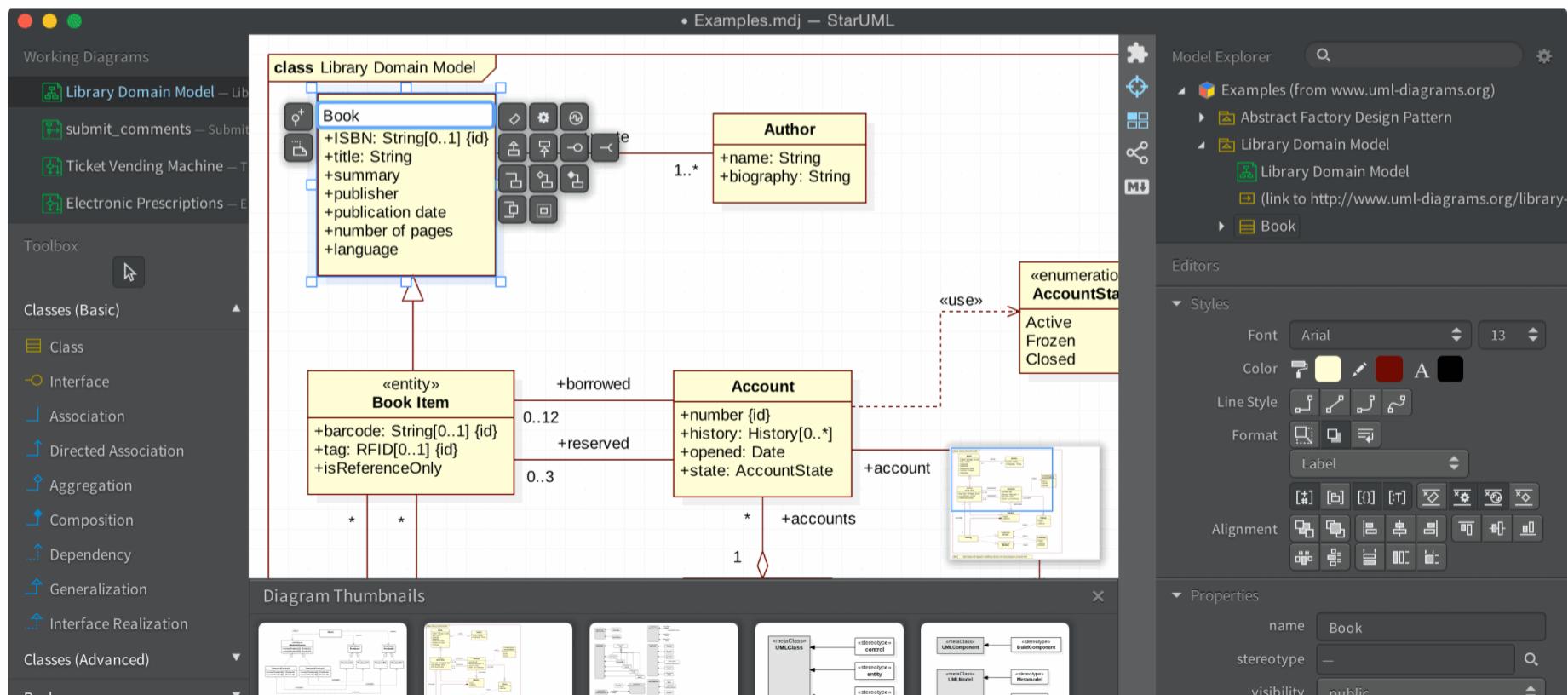


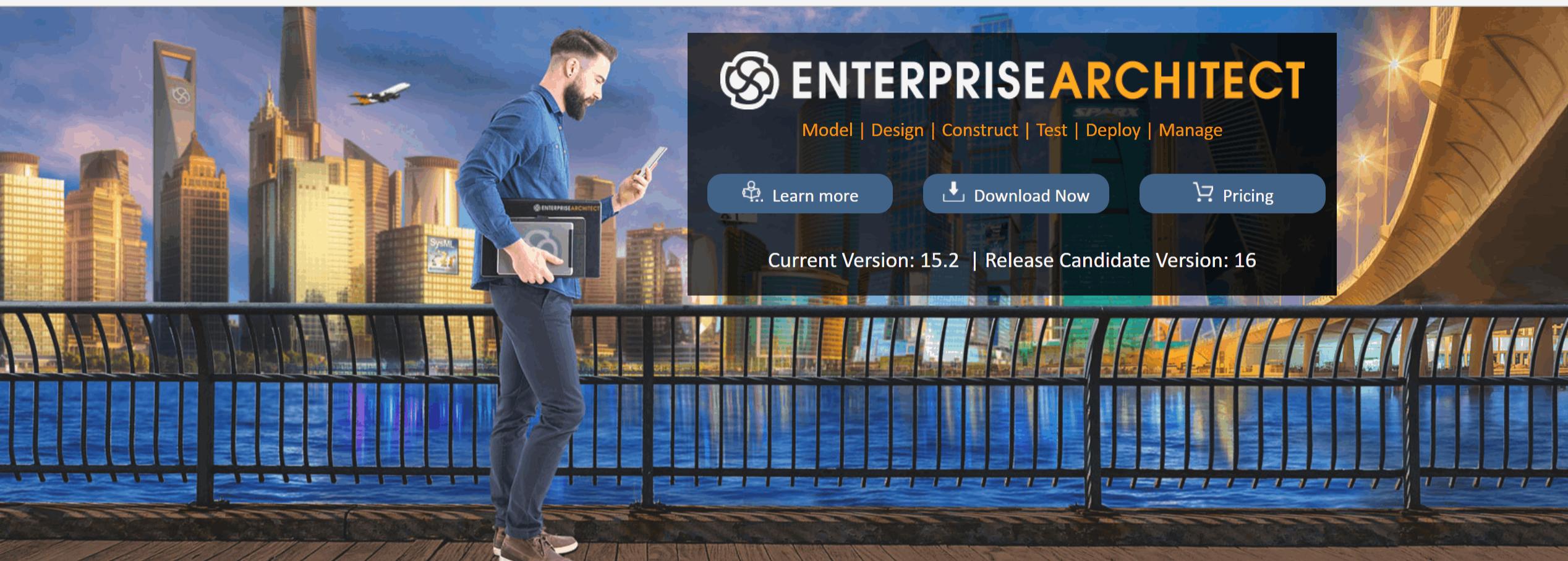
# StarUML

A sophisticated software modeler for agile and concise modeling

Download for Windows

Current version: 4.1.6





# ENTERPRISEARCHITECT

Model | Design | Construct | Test | Deploy | Manage

 Learn more

 Download Now

 Pricing

Current Version: 15.2 | Release Candidate Version: 16

# Make Your Vision a Reality



## Enterprise Architect

Fast Intuitive Modeling & Design

# Enterprise Architecture

## Introduction

## Enterprise Architecture Overview

## Meet the Enterprise Architecture Tools

## Uses of an Enterprise Architecture

## Architecture Program Setup

## Managing an Enterprise Architecture

## Developing an Enterprise Architecture

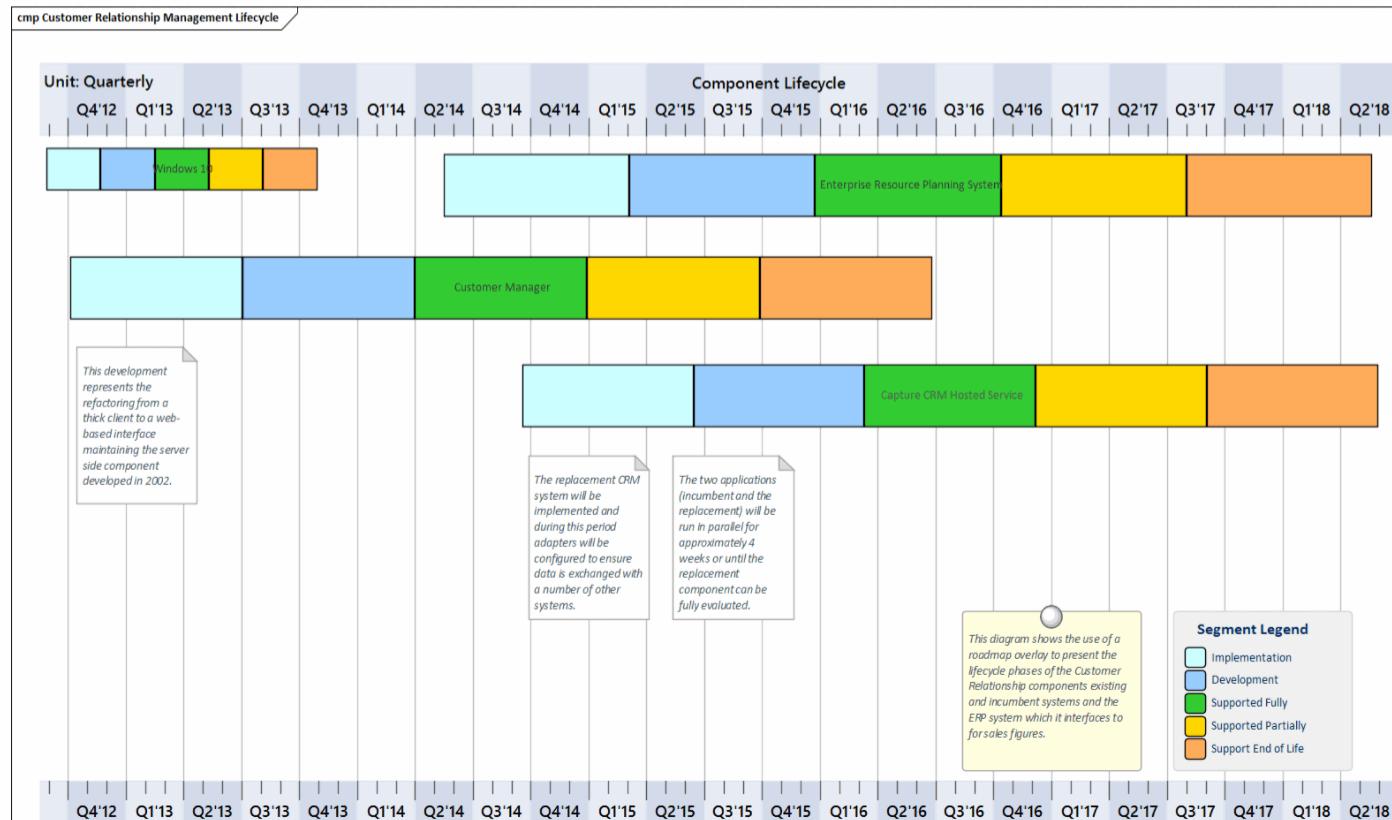
## Documenting an Enterprise Architecture

## Enterprise Architecture Techniques

## Additional Enterprise Architecture Tools

Enterprise Architecture has emerged as a discipline that can help steer the 'ship' of the enterprise through both quiet and turbulent waters, charting a course from its current location to a future location in a safe and streamlined way. The discipline has become more prevalent in recent years, but the precepts go back almost to the beginning of what is often termed the information age. An enterprise is one of the most complex man-made systems and is composed of human, political, social, software, hardware and technology components. In an enterprise of any appreciable size, it is impossible for a single person to understand the way the parts all work together, let alone understand its position in relation to the system of other organizations that form its environment, or to determine how it can evolve.

Enterprise Architecture can be used to create visualizations of the enterprise at different levels of abstractions and to create Roadmaps that show how the enterprise can be transitioned from its baseline (current) state to a target (future) state.





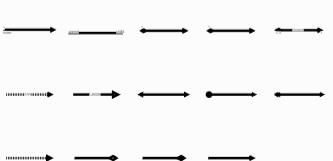
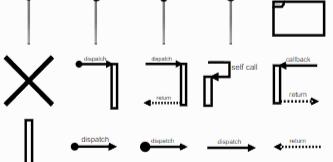
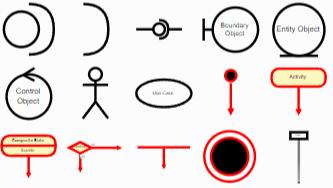
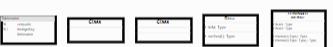
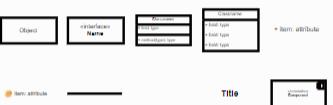
Drag elements here

## General

## Misc

## Advanced

## UML



+ More Shapes...

Page-1



Diagram

Style

## View

- Grid
  - Page View
  - Background
  - Shadow
- 10 pt
- Change

## Options

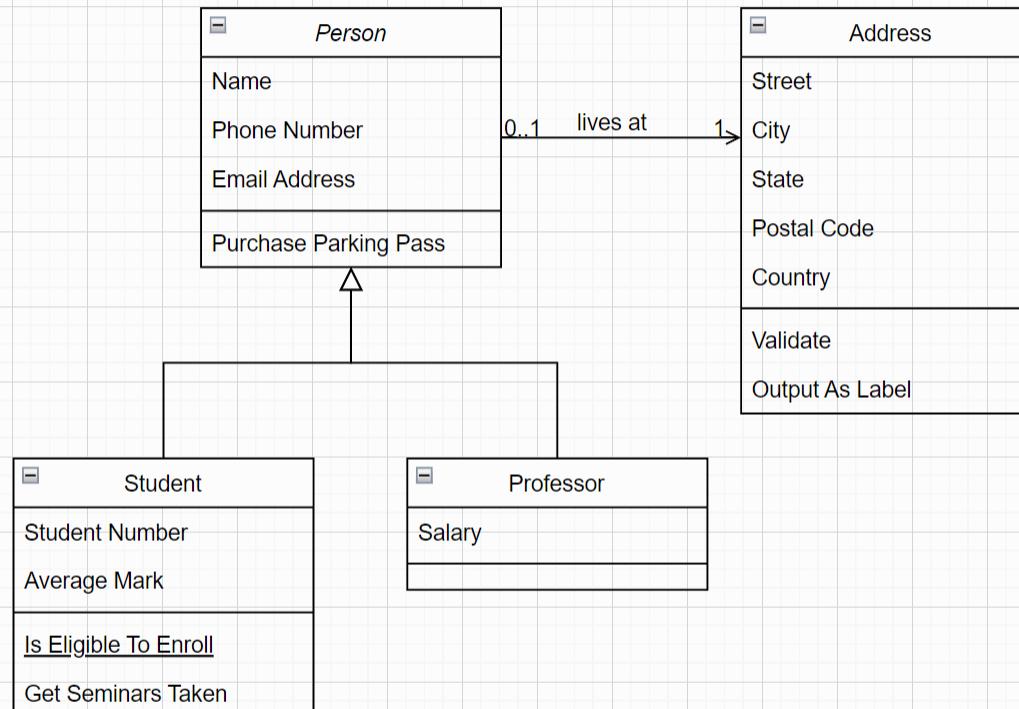
- Connection Arrows
- Connection Points
- Guides

## Paper Size

- A4 (210 mm x 297 mm)
- Portrait  Landscape

Edit Data...

Clear Default Style





Join at  
**slido.com**  
**#VAVA4**

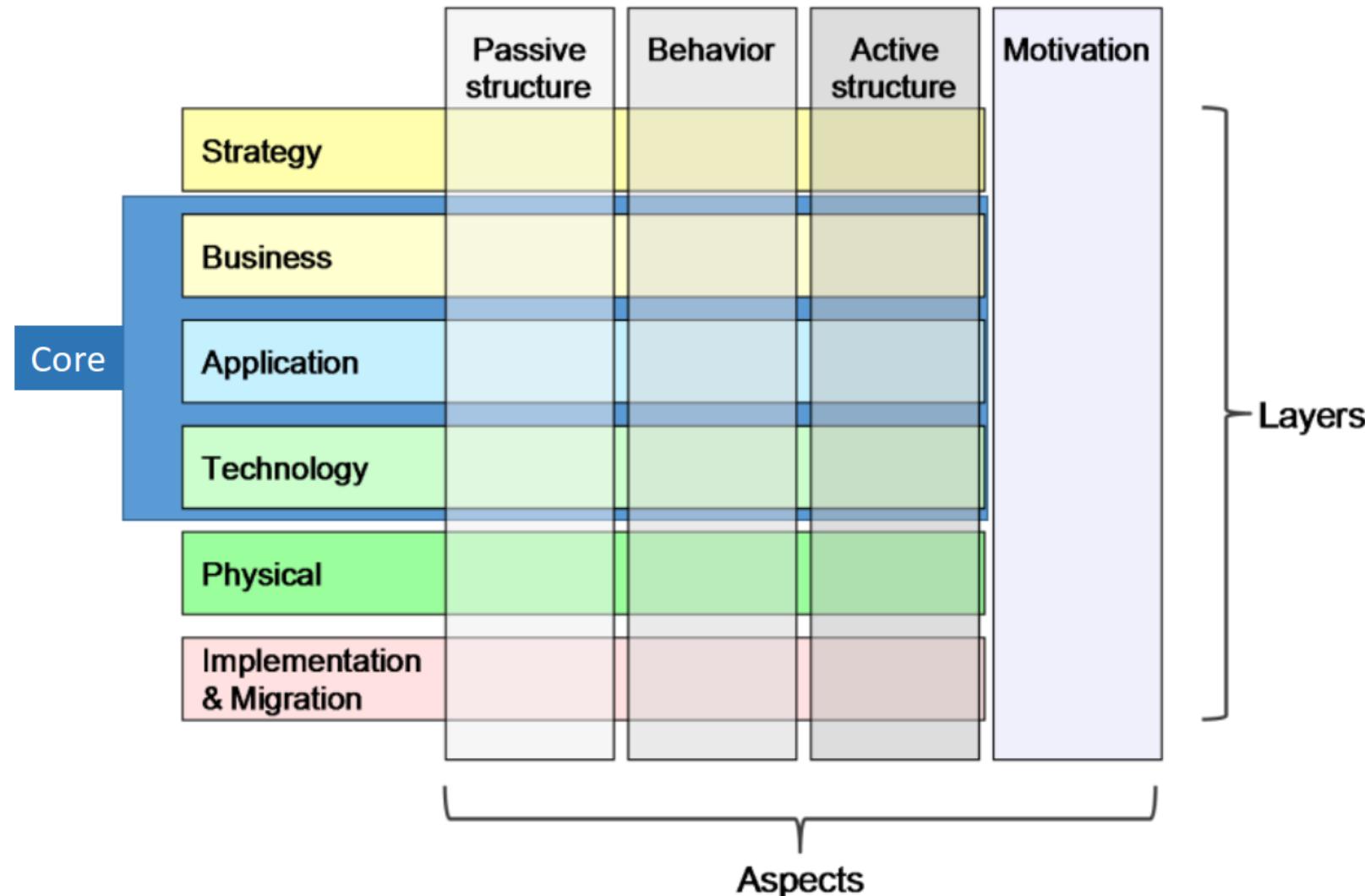


# SLIDO – VAVA4

1. Ktoré Java kolekcie sú najčastejšie používané vo vývoji aplikácií a prečo?
2. Z hľadiska Enterprise Architektúry a ArchiMate, je kolekcia aký typ elementu a prečo?
3. Čo je kolekcia z pohľadu Javy?

<https://wall.sli.do/event/sW3ieupQnVBySBmFiw67rk?section=d1135817-a1b7-4ba2-a55a-726aa59e59b2>

# Úplný rámec - vrstvy a aspekty



# Aspekty (Aspects)

ArchiMate striktne oddeluje 3 základné aspeky:

1. **Active Structure Aspect** – reprezentujú **štrukturálne elementy** (napr. aktérov, role, komponenty, hardvér, ktoré sú schopné určitého chovania. Tiež sú tieto elementy označované ako aktívne).
2. **Behavior Aspect** – Reprezentuje **chovanie** (napr. procesy, funkcie, služby) **vykonávané aktívnymi** (štrukturálnymi) **elementami**. Chovanie je priradené (assigned) aktívному elementu.
3. **Passive Structure Aspect** (pasívne elementy) – **reprezentujú objekty** (napr. biznis objekty, dátové objekty ale aj fyzické objekty), na ktorých môže byť aplikované chovanie aktívneho elementu (či sú takýmto chovaním ovplyvnený)

# Pochopenie typov aspektov elementu

- Aká je aktívna štruktúra, behaviorálna a pasívna štruktúra?
  - Kuchár varí jedlo
  - Mechanik zvára auto
- **Kuchár/Mechanik**
  - **Aktívny prvok** štruktúry schopný **vykonávať správanie, zobrazuje správanie – subject/podmet (podstatné meno)**
- **Varenie/zváranie**
  - **Behaviorálny** prvok je jednotka **aktivity vykonávaná aktívnym elementom** (štruktúrami) - typicky verb/**prísudok (sloveso)**
- **Potraviny/Auto**
  - **Prvok pasívnej** štruktúry, **na ktorom sa vykonáva správanie** – objekt/**predmet (podstatné meno)** (informácie, údaje alebo fyzické)

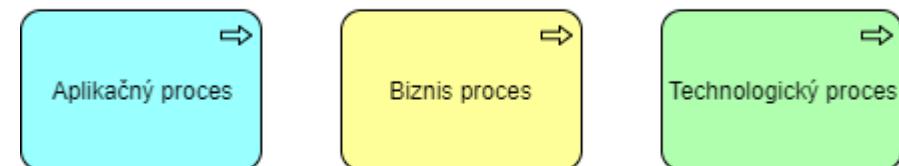
Inšpirácia prirodzeným jazykom

# Rozlúšenie elementov podľa rohov

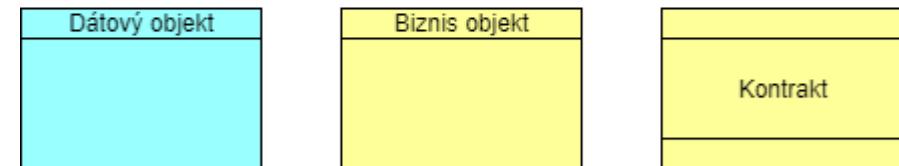
## 1. Ostré rohy – aktívne elementy



## 2. Zaoblené rohy – element chovania (behavior)

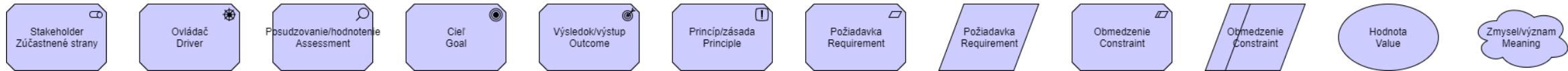
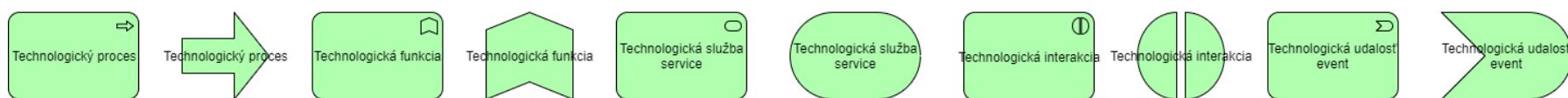
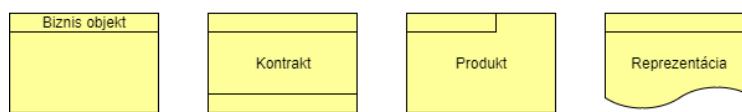
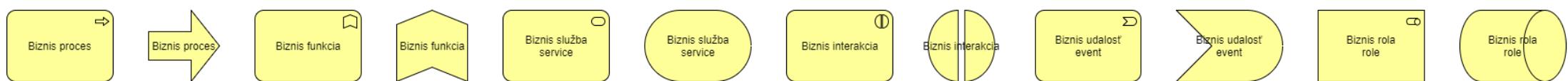
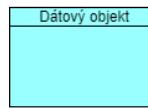
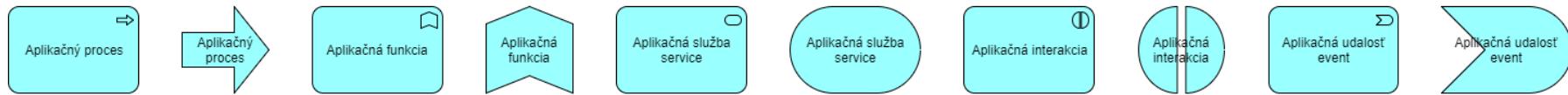
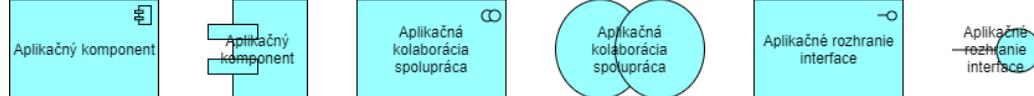


## 3. Ostrý roh s preškrtnutím – pasívne elementy



## 4. Skosené rohy – motivačný element





		Java Language											
		java	javac	javadoc	jar	javap	jdeps	Scripting					
JDK	Tools & Tool APIs	Security	Monitoring	JConsole	VisualVM	JMC	JFR						
		JPDA	JVM TI	IDL	RMI	Java DB	Deployment						
		Internationalization		Web Services		Troubleshooting							
JRE	Deployment	Java Web Start			Applet / Java Plug-in								
		JavaFX											
		Swing		Java 2D		AWT	Accessibility						
JRE	User Interface Toolkits	Drag and Drop		Input Methods		Image I/O	Print Service	Sound					
		IDL	JDBC	JNDI	RMI	RMI-IIOP		Scripting					
		Beans	Security		Serialization		Extension Mechanism						
JRE	Integration Libraries	JMX	XML JAXP		Networking		Override Mechanism						
		JNI	Date and Time		Input/Output		Internationalization						
		lang and util											
JRE	Base Libraries	Math		Collections		Ref Objects		Regular Expressions					
		Logging		Management		Instrumentation		Concurrency Utilities					
		Reflection	Versioning		Preferences API		JAR	Zip					
Java Virtual Machine		Java HotSpot Client and Server VM											

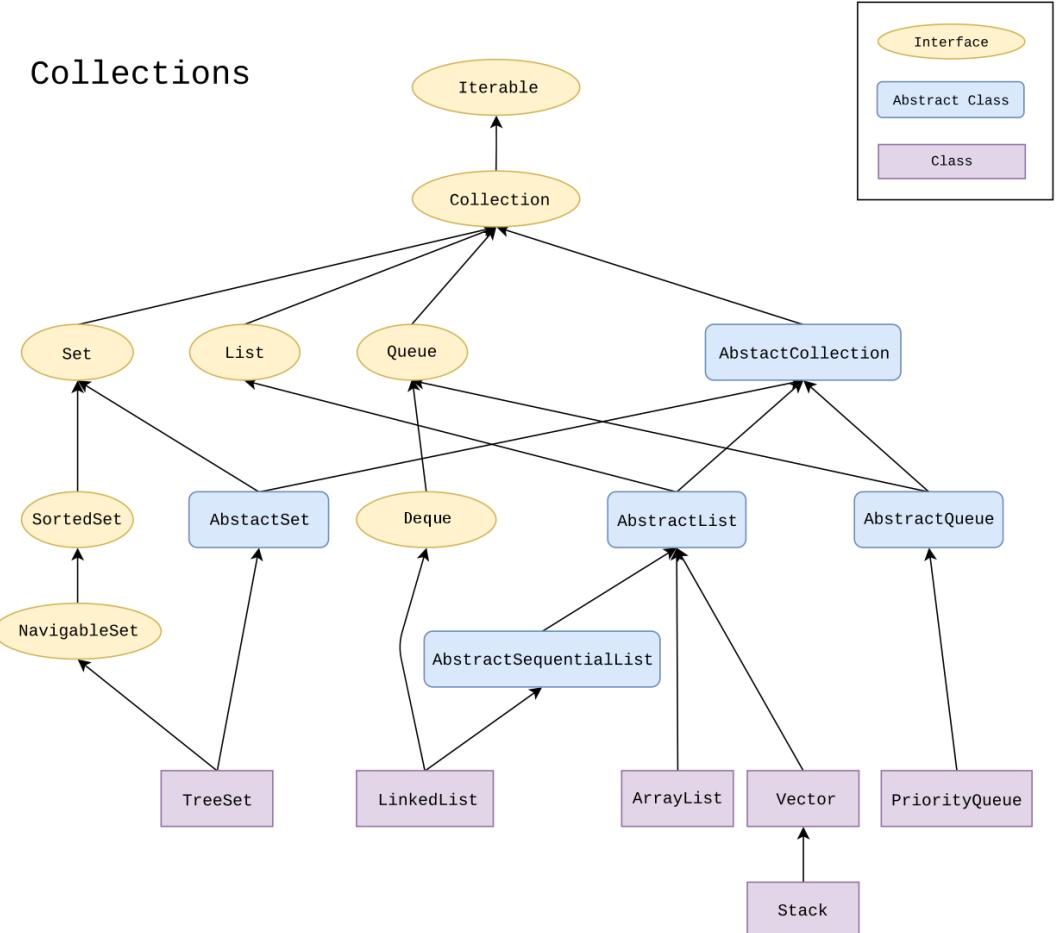
Java SE

API

Compact Profiles

# Java collections framework (JCF)

- Je **súbor tried a rozhraní**, ktoré implementujú bežne **znovu použiteľné kolekcie dátových štruktúr**
- Aj keď sa označuje ako rámc, **fnguje na spôsob knižnice**
- Rámc kolekcií poskytuje **rozhrania**, ktoré definujú rôzne kolekcie, aj **triedy**, ktoré **ich implementujú**



# Príklady kolekcií

1. Čakatelia na pozíciu profesora
2. Ašpiranti na nobelové ceny
3. Zoznam študentov s ID
4. Hra poker (kolekcia kariet)
5. Poštová schránka s listami (kolekcia listov)
6. Telefónny zoznam (priradenie mien k telefónnym číslam)
7. Zoznam tvarov vo vzorkovnici modelovacieho nástroja

# Architektúra kolekcií

- Takmer všetky kolekcie v Jave sú **odvodené z rozhrania java.util.Collection**
- Rozhranie **Collection** definuje **základné časti všetkých kolekcií**
- Rozhranie uvádza **metódy add()** a **remove()** na pridávanie a odstraňovanie z kolekcie
- Vyžaduje sa aj metóda **toArray()**, ktorá **konvertuje** kolekciu **na jednoduché pole** všetkých prvkov v kolekcii
- Metóda **contains()** skontroluje, či je **zadaný prvok v kolekcií**
- Rozhranie Collection je podrozhraním **java.lang.Iterable**, takže každá kolekcia môže používať **for-each**
- Rozhranie Iterable poskytuje metódu **iterator()** používanú príkazmi **for-each**
- Všetky kolekcie majú **iterator**, ktorý **prechádza všetkými prvkami v kolekcií**
- Kolekcia je **generická**
- Akákoľvek kolekcia môže byť napísaná na uloženie akejkoľvek triedy

Ordered lists  
(Zoznamy)

Maps  
(Dictionaries)

Sets  
(Množiny)

## The Java™ Tutorials

*The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.*

See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

The Java Tutorials are practical guides for programmers who want to use the Java programming language to create applications. They include hundreds of complete, working examples, and dozens of lessons. Groups of related lessons are organized into "trails".

### Trails Covering the Basics

These trails are available in book form as *The Java Tutorial, Sixth Edition*. To buy this book, refer to the box to the right.

- » [Getting Started](#) — An introduction to Java technology and lessons on installing Java development software and using it to create a simple program.
- » [Learning the Java Language](#) — Lessons describing the essential concepts and features of the Java Programming Language.
- » [Essential Java Classes](#) — Lessons on exceptions, basic input/output, concurrency, regular expressions, and the platform environment.
- » [Collections](#) — Lessons on using and extending the Java Collections Framework.
- » [Date-Time APIs](#) — How to use the `java.time` package to write date and time code.
- » [Deployment](#) — How to package applications and applets using JAR files, and deploy them using Java Web Start and Java Plug-in.
- » [Preparation for Java Programming Language Certification](#) — List of available training and tutorial resources.

### Creating Graphical User Interfaces

- » [Creating a GUI with Swing](#) — A comprehensive introduction to GUI creation on the Java platform.
- » [Creating a JavaFX GUI](#) — A collection of JavaFX tutorials.

### Specialized Trails and Lessons

These trails and lessons are only available as web pages.

- » [Custom Networking](#) — An introduction to the Java platform's powerful networking features.
- » [The Extension Mechanism](#) — How to make custom APIs available to all applications running on the Java platform.
- » [Full-Screen Exclusive Mode API](#) — How to write applications that more fully utilize the user's graphics hardware.
- » [Generics](#) — An enhancement to the type system that supports operations on objects of various types while providing compile-time type safety. Note that this lesson is for advanced users. The [Java Language](#) trail contains a [Generics](#) lesson that is suitable for beginners.
- » [Internationalization](#) — An introduction to designing software so that it can be easily adapted (localized) to various languages and regions.
- » [JavaBeans](#) — The Java platform's component technology.
- » [JAXB](#) — Introduces the Java architecture for XML Binding (JAXB) technology.
- » [JAXP](#) — Introduces the Java API for XML Processing (JAXP) technology.



Not sure where to start?

See [Learning Paths](#)

### Tutorial Contents

[Really Big Index](#)

### Tutorial Resources

- ▶ Last Updated 2022-03-04
- ▶ The Java Tutorials' Blog has news and updates about the Java SE tutorials.
- ▶ Download the latest Java Tutorials bundle.

### In Book Form

- ▶ *The Java Tutorial, Sixth Edition*.  
[Amazon.com](#).

### Other Resources

- ▶ [Java SE 8 Developer Guides](#)
- ▶ [JDK 8 API Documentation](#)

### Oracle Training and

## The Java™ Tutorials

« Previous • Trail • Next »

[Home Page](#)

*The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.*

*See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.*

*See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.*

### Trail: Collections

This section describes the Java Collections Framework. Here, you will learn what collections are and how they can make your job easier and programs better. You'll learn about the core elements — interfaces, implementations, aggregate operations, and algorithms — that comprise the Java Collections Framework.

 **Introduction** tells you what collections are, and how they'll make your job easier and your programs better. You'll learn about the core elements that comprise the Collections Framework: *interfaces*, *implementations* and *algorithms*.

 **Interfaces** describes the core collection *interfaces*, which are the heart and soul of the Java Collections Framework. You'll learn general guidelines for effective use of these interfaces, including when to use which interface. You'll also learn idioms for each interface that will help you get the most out of the interfaces.

 **Aggregate Operations** iterate over collections on your behalf, which enable you to write more concise and efficient code that process elements stored in collections.

 **Implementations** describes the JDK's *general-purpose collection implementations* and tells you when to use which implementation. You'll also learn about the *wrapper implementations*, which add functionality to general-purpose implementations.

 **Algorithms** describes the *polymorphic algorithms* provided by the JDK to operate on collections. With any luck you'll never have to write your own sort routine again!

 **Custom Implementations** tells you why you might want to write your own collection implementation (instead of using one of the general-purpose implementations provided by the JDK), and how you'd go about it. It's easy with the JDK's *abstract collection implementations*!

 **Interoperability** tells you how the Collections Framework interoperates with older APIs that predate the addition of Collections to Java. Also, it tells you how to design new APIs so that they'll interoperate seamlessly with other new APIs.

# Lesson: Introduction to Collections

A *collection* — sometimes called a *container* — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. Typically, they represent data items that form a natural group, such as a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers). If you have used the Java programming language — or just about any other programming language — you are already familiar with collections.

## What Is a Collections Framework?

A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

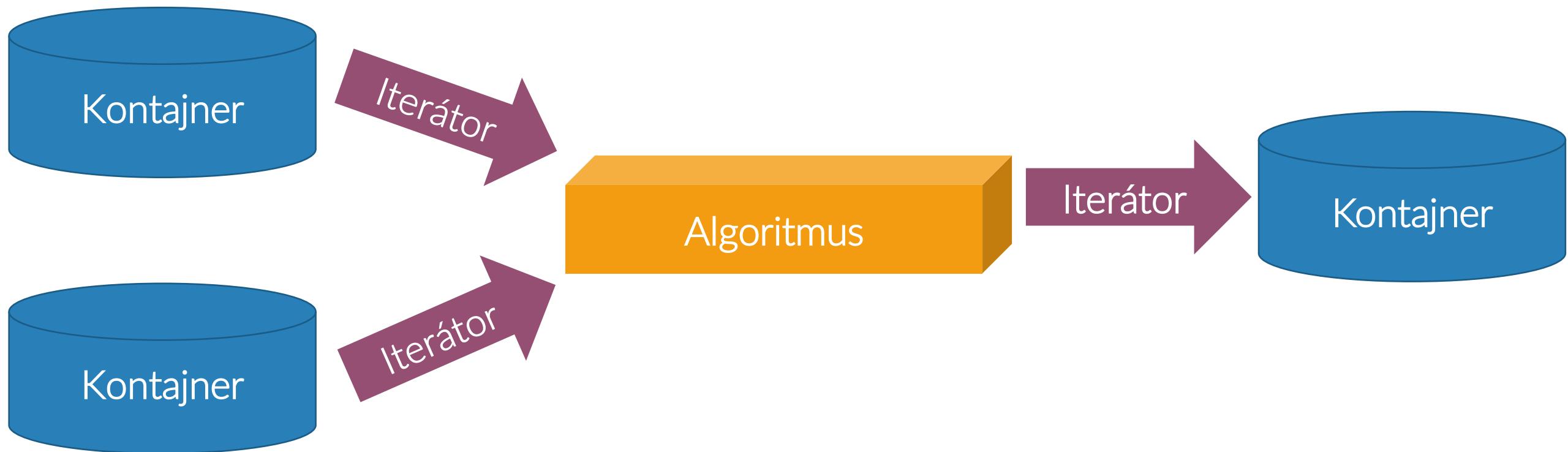
- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

Apart from the Java Collections Framework, the best-known examples of collections frameworks are the C++ Standard Template Library (STL) and Smalltalk's collection hierarchy. Historically, collections frameworks have been quite complex, which gave them a reputation for having a steep learning curve. We believe that the Java Collections Framework breaks with this tradition, as you will learn for yourself in this chapter.

## Benefits of the Java Collections Framework

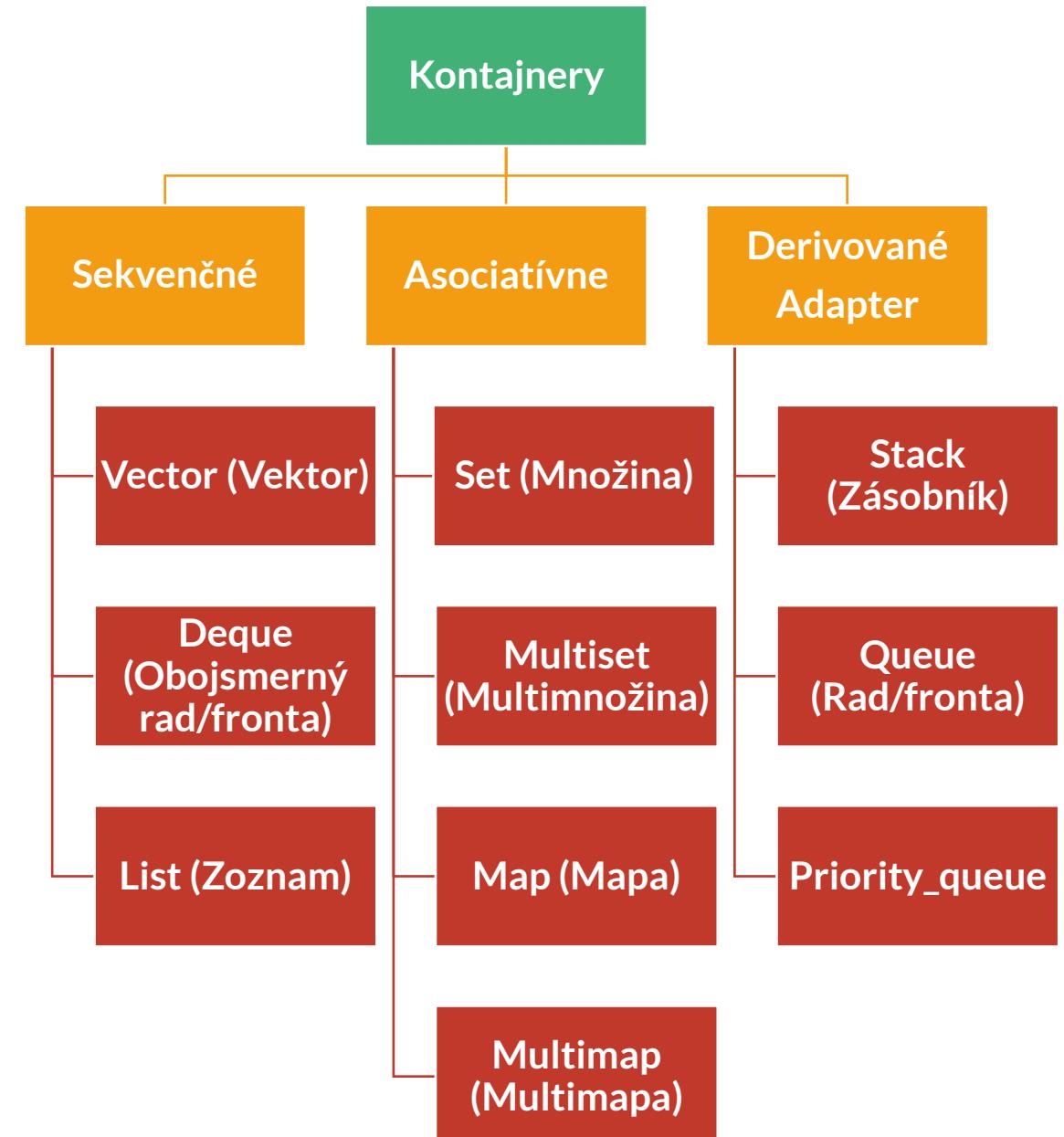
The Java Collections Framework provides the following benefits:

- **Reduces programming effort:** By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work. By facilitating interoperability among unrelated APIs, the Java Collections Framework frees you from writing adapter objects or conversion code to connect APIs.
- **Increases program speed and quality:** This Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you're freed from the drudgery of writing your own data structures, you'll have more time to devote to improving programs' quality and performance.
- **Allows interoperability among unrelated APIs:** The collection interfaces are the vernacular by which APIs pass collections back and forth. If my network administration API furnishes a collection of node names and if your GUI toolkit expects a collection of column headings, our APIs will interoperate seamlessly, even though they were written independently.
- **Reduces effort to learn and to use new APIs:** Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.
- **Reduces effort to design new APIs:** This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.
- **Fosters software reuse:** New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.



# Kontajnery

- Sekvenčné kontajnery umožňujú pracovať s prvkami, ktoré sú uložené v určitom poradí (sekvencií)
  - Pristupujeme k nim **cez indexy**, prípadne pomocou **iterátorov**.
  - Veľkou výhodou sekvenčných kontajnerov je **možnosť pracovať s údajmi dynamicky**, t.j. zvyšovať alebo znížovať ich počet
- Okrem sekvenčných kontajnerov existuje aj ďalšia skupina kontajnerov, tzv. **asociatívne** a **derivované kontajnery**



# Príklady štandardných knižníc

C	C++	Python	Go	Java
<ul style="list-style-type: none"><li>• assert</li><li>• float</li><li>• math</li><li>• stdio</li><li>• stdlib</li><li>• string</li><li>• threads</li></ul>	<ul style="list-style-type: none"><li>• list</li><li>• map</li><li>• deque</li><li>• vector</li><li>• algorithm</li><li>• iterator</li><li>• string</li><li>• regex</li><li>• fstream</li><li>• iostream</li><li>• exception</li></ul>	<ul style="list-style-type: none"><li>• list</li><li>• dict</li><li>• tuple</li><li>• range</li><li>• open</li><li>• enum</li><li>• collections</li><li>• math</li><li>• pickle</li><li>• zlib</li><li>• threading</li><li>• queue</li><li>• email</li><li>• json</li></ul>	<ul style="list-style-type: none"><li>• tar</li><li>• gzip</li><li>• heap</li><li>• crypto</li><li>• hash</li><li>• image</li><li>• io</li><li>• mime</li><li>• net</li><li>• os</li><li>• path</li><li>• strings</li><li>• time</li><li>• unicode</li></ul>	<ul style="list-style-type: none"><li>• java.lang</li><li>• java.io</li><li>• java.net</li><li>• java.math</li><li>• java.text</li><li>• java.awt.image</li><li>• java.security</li><li>• java.beans</li></ul>

ALL CLASSES

SEARCH:



SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

**Module** java.base**Package** java.util

## Class Collections

java.lang.Object  
java.util.Collections

```
public class Collections
extends Object
```

This class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.

The methods of this class all throw a `NullPointerException` if the collections or class objects provided to them are null.

The documentation for the polymorphic algorithms contained in this class generally includes a brief description of the *implementation*. Such descriptions should be regarded as *implementation notes*, rather than parts of the *specification*. Implementors should feel free to substitute other algorithms, so long as the specification itself is adhered to. (For example, the algorithm used by `sort` does not have to be a mergesort, but it does have to be *stable*.)

The "destructive" algorithms contained in this class, that is, the algorithms that modify the collection on which they operate, are specified to throw `UnsupportedOperationException` if the collection does not support the appropriate mutation primitive(s), such as the `set` method. These algorithms may, but are not required to, throw this exception if an invocation would have no effect on the collection. For example, invoking the `sort` method on an unmodifiable list that is already sorted may or may not throw `UnsupportedOperationException`.

This class is a member of the Java Collections Framework.

**Since:**

1.2

**See Also:**

`Collection`, `Set`, `List`, `Map`

### Field Summary

ALL CLASSES

SEARCH:  Search X

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

**Module** java.base**Package** java.util

## Interface Collection<E>

**Type Parameters:**

E - the type of elements in this collection

**All Superinterfaces:**

Iterable&lt;E&gt;

**All Known Subinterfaces:**

BeanContext, BeanContextServices, BlockingDeque&lt;E&gt;, BlockingQueue&lt;E&gt;, Deque&lt;E&gt;, EventSet, List&lt;E&gt;, NavigableSet&lt;E&gt;, Queue&lt;E&gt;, Set&lt;E&gt;, SortedSet&lt;E&gt;, TransferQueue&lt;E&gt;

**All Known Implementing Classes:**

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

---

```
public interface Collection<E>
extends Iterable<E>
```

The root interface in the *collection hierarchy*. A collection represents a group of objects, known as its *elements*. Some collections allow duplicate elements and others do not. Some are ordered and others unordered. The JDK does not provide any *direct* implementations of this interface: it provides implementations of more specific subinterfaces like *Set* and *List*. This interface is typically used to pass collections around and manipulate them where maximum generality is desired.

*Bags* or *multisets* (unordered collections that may contain duplicate elements) should implement this interface directly.

All general-purpose *Collection* implementation classes (which typically implement *Collection* indirectly through one of its subinterfaces) should provide two "standard" constructors: a void (no arguments) constructor, which creates an empty collection, and a constructor with a single argument of type *Collection*, which creates a new collection with the same elements as its argument. In effect, the latter constructor allows the user to copy any collection, producing an equivalent collection of the desired implementation type. There is no way to enforce this convention (as interfaces cannot contain constructors) but all of the general-purpose *Collection* implementations in the Java platform libraries

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

SEARCH:

**Module** java.base**Package** java.util

## Interface Iterator<E>

**Type Parameters:**

E - the type of elements returned by this iterator

**All Known Subinterfaces:**

EventIterator, ListIterator<E>, PrimitiveIterator<T, T\_CONS>, PrimitiveIterator.OfDouble, PrimitiveIterator.OfInt, PrimitiveIterator.OfLong, XMLEventReader

**All Known Implementing Classes:**

BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

---

```
public interface Iterator<E>
```

An iterator over a collection. `Iterator` takes the place of `Enumeration` in the Java Collections Framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

**API Note:**

An `Enumeration` can be converted into an `Iterator` by using the `Enumeration.asIterator()` method.

**Since:**

1.2

**See Also:**

`Collection`, `ListIterator`, `Iterable`

### Method Summary

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

## Lesson: Implementations

Implementations are the data objects used to store collections, which implement the interfaces described in the [Interfaces](#) section. This lesson describes the following kinds of implementations:

- **General-purpose implementations** are the most commonly used implementations, designed for everyday use. They are summarized in the table titled General-purpose-implementations.
- **Special-purpose implementations** are designed for use in special situations and display nonstandard performance characteristics, usage restrictions, or behavior.
- **Concurrent implementations** are designed to support high concurrency, typically at the expense of single-threaded performance. These implementations are part of the `java.util.concurrent` package.
- **Wrapper implementations** are used in combination with other types of implementations, often the general-purpose ones, to provide added or restricted functionality.
- **Convenience implementations** are mini-implementations, typically made available via static factory methods, that provide convenient, efficient alternatives to general-purpose implementations for special collections (for example, singleton sets).
- **Abstract implementations** are skeletal implementations that facilitate the construction of custom implementations — described later in the [Custom Collection Implementations](#) section. An advanced topic, it's not particularly difficult, but relatively few people will need to do it.

The general-purpose implementations are summarized in the following table.

General-purpose Implementations

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

As you can see from the table, the Java Collections Framework provides several general-purpose implementations of the [Set](#), [List](#), and [Map](#) interfaces. In each case, one implementation — [HashSet](#), [ArrayList](#), and [HashMap](#) — is clearly the one to use for most applications, all other things being equal. Note that the [SortedSet](#) and the [SortedMap](#) interfaces do not have rows in the table. Each of those interfaces has one implementation ([TreeSet](#) and [TreeMap](#)) and is listed in the [Set](#) and the [Map](#) rows. There are two general-purpose [Queue](#) implementations — [LinkedList](#), which is also a [List](#) implementation, and [PriorityQueue](#), which is omitted from the table. These two implementations provide very different semantics: [LinkedList](#) provides

# JDK Release Notes

## Oracle Java SE Publicly Available Release Families

[JDK 17 Release notes](#)[JDK 16 Release notes](#)[JDK 15 Release notes](#)[JDK 14 Release notes](#)[JDK 13 Release notes](#)[JDK 12 Release notes](#)[JDK 11 Release notes](#)[JDK 10 Release notes](#)[JDK 9 Release Notes](#)[JDK 8 Release Notes](#)[JDK 7 Release Notes for update releases](#)[JDK 7 Release Notes for revision builds](#)[JDK 6 Release Notes post 6u45, and revision builds of earlier releases](#)[JDK 6 Release Notes through 6u45](#)

## Oracle Java SE Commercial Offering Releases

JDK 7 and JDK 8 release families that have reached end-of-public-updates but are still supported for Oracle Customers. To access the latest releases of these families you will need to license Java SE Advanced.

### Resources for

### Why Oracle

### Learn

### What's New

### Contact Us

[Careers](#)[Analyst Reports](#)[What is cloud computing?](#)[Try Oracle Cloud Free Tier](#)[US Sales: +1.800.633.0738](#)[Developers](#)[Gartner MQ for ERP Cloud](#)[What is CRM?](#)[Oracle Product Navigator](#)[How can we help?](#)

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

## Set Implementations

The `Set` implementations are grouped into general-purpose and special-purpose implementations.

### General-Purpose Set Implementations

There are three general-purpose `Set` implementations — `HashSet`, `TreeSet`, and `LinkedHashSet`. Which of these three to use is generally straightforward. `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`; otherwise, use `HashSet`. 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`. The `LinkedHashSet` implementation spares its clients from the unspecified, generally chaotic ordering provided by `HashSet` without incurring the increased cost associated with `TreeSet`.

One thing worth keeping in mind about `HashSet` is that iteration is linear in the sum of the number of entries and the number of buckets (the *capacity*). Thus, choosing an initial capacity that's too high can waste both space and time. On the other hand, choosing an initial capacity that's too low wastes time by copying the data structure each time it's forced to increase its capacity. If you don't specify an initial capacity, the default is 16. In the past, there was some advantage to choosing a prime number as the initial capacity. This is no longer true. Internally, the capacity is always rounded up to a power of two. The initial capacity is specified by using the `int` constructor. The following line of code allocates a `HashSet` whose initial capacity is 64.

```
Set<String> s = new HashSet<String>(64);
```

The `HashSet` class has one other tuning parameter called the *load factor*. If you care a lot about the space consumption of your `HashSet`, read the `HashSet` documentation for more information. Otherwise, just accept the default; it's almost always the right thing to do.

If you accept the default load factor but want to specify an initial capacity, pick a number that's about twice the size to which you expect the set to grow. If your guess is way off, you may waste a bit of space, time, or both, but it's unlikely to be a big problem.

`LinkedHashSet` has the same tuning parameters as `HashSet`, but iteration time is not affected by capacity. `TreeSet` has no tuning parameters.

### Special-Purpose Set Implementations

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

## List Implementations

List implementations are grouped into general-purpose and special-purpose implementations.

### General-Purpose List Implementations

There are two general-purpose [List](#) implementations — [ArrayList](#) and [LinkedList](#). Most of the time, you'll probably use [ArrayList](#), which offers constant-time positional access and is just plain fast. It does not have to allocate a node object for each element in the [List](#), and it can take advantage of [System.arraycopy](#) when it has to move multiple elements at the same time. Think of [ArrayList](#) as [Vector](#) without the synchronization overhead.

If you frequently add elements to the beginning of the [List](#) or iterate over the [List](#) to delete elements from its interior, you should consider using [LinkedList](#). These operations require constant-time in a [LinkedList](#) and linear-time in an [ArrayList](#). But you pay a big price in performance. Positional access requires linear-time in a [LinkedList](#) and constant-time in an [ArrayList](#). Furthermore, the constant factor for [LinkedList](#) is much worse. If you think you want to use a [LinkedList](#), measure the performance of your application with both [LinkedList](#) and [ArrayList](#) before making your choice; [ArrayList](#) is usually faster.

[ArrayList](#) has one tuning parameter — the *initial capacity*, which refers to the number of elements the [ArrayList](#) can hold before it has to grow. [LinkedList](#) has no tuning parameters and seven optional operations, one of which is [clone](#). The other six are [addFirst](#), [getFirst](#), [removeFirst](#), [addLast](#), [getLast](#), and [removeLast](#). [LinkedList](#) also implements the [Queue](#) interface.

### Special-Purpose List Implementations

[CopyOnWriteArrayList](#) is a [List](#) implementation backed up by a copy-on-write array. This implementation is similar in nature to [CopyOnWriteArraySet](#). No synchronization is necessary, even during iteration, and iterators are guaranteed never to throw [ConcurrentModificationException](#). This implementation is well suited to maintaining event-handler lists, in which change is infrequent, and traversal is frequent and potentially time-consuming.

If you need synchronization, a [Vector](#) will be slightly faster than an [ArrayList](#) synchronized with [Collections.synchronizedList](#). But [Vector](#) has loads of legacy operations, so be careful to always manipulate the [Vector](#) with the [List](#) interface or else you won't be able to replace the implementation at a later time.

If your [List](#) is fixed in size — that is, you'll never use [remove](#), [add](#), or any of the bulk operations other than [containsAll](#) — you have a third option that's definitely worth considering. See [Arrays.asList](#) in the [Convenience Implementations](#) section for more information.

*The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.*

*See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.*

*See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.*

## Map Implementations

Map implementations are grouped into general-purpose, special-purpose, and concurrent implementations.

### General-Purpose Map Implementations

The three general-purpose Map implementations are [HashMap](#), [TreeMap](#) and [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](#). In this respect, the situation for Map is analogous to Set. Likewise, everything else in the [Set Implementations](#) section also applies to Map implementations.

[LinkedHashMap](#) provides two capabilities that are not available with [LinkedHashSet](#). When you create a [LinkedHashMap](#), you can order it based on key access rather than insertion. In other words, merely looking up the value associated with a key brings that key to the end of the map. Also, [LinkedHashMap](#) provides the `removeEldestEntry` method, which may be overridden to impose a policy for removing stale mappings automatically when new mappings are added to the map. This makes it very easy to implement a custom cache.

For example, this override will allow the map to grow up to as many as 100 entries and then it will delete the eldest entry each time a new entry is added, maintaining a steady state of 100 entries.

```
private static final int MAX_ENTRIES = 100;

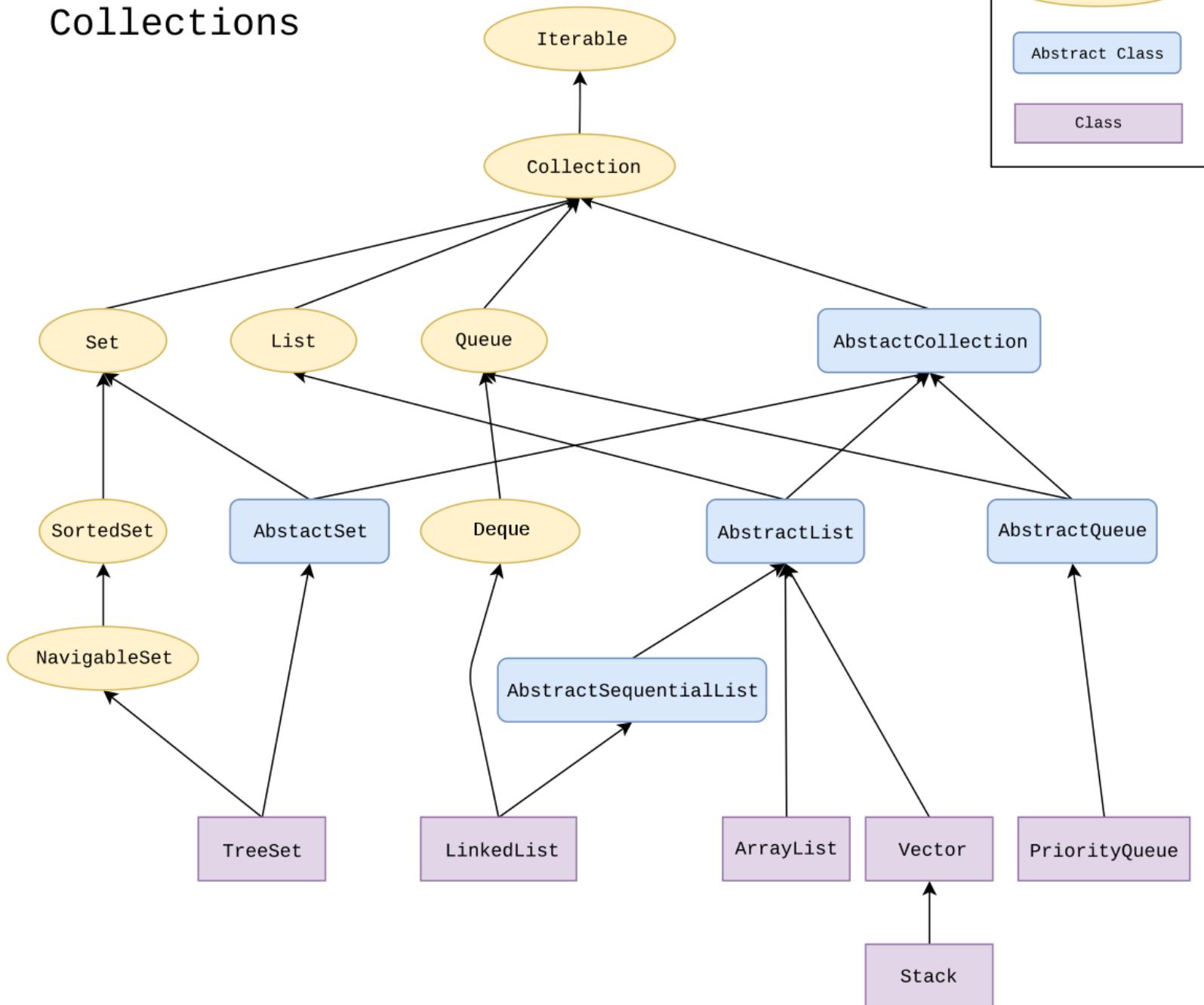
protected boolean removeEldestEntry(Map.Entry eldest) {
    return size() > MAX_ENTRIES;
}
```

### Special-Purpose Map Implementations

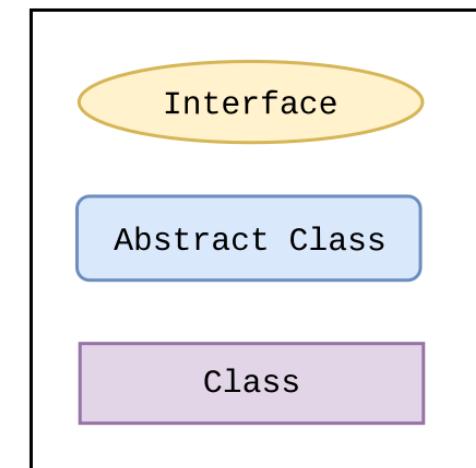
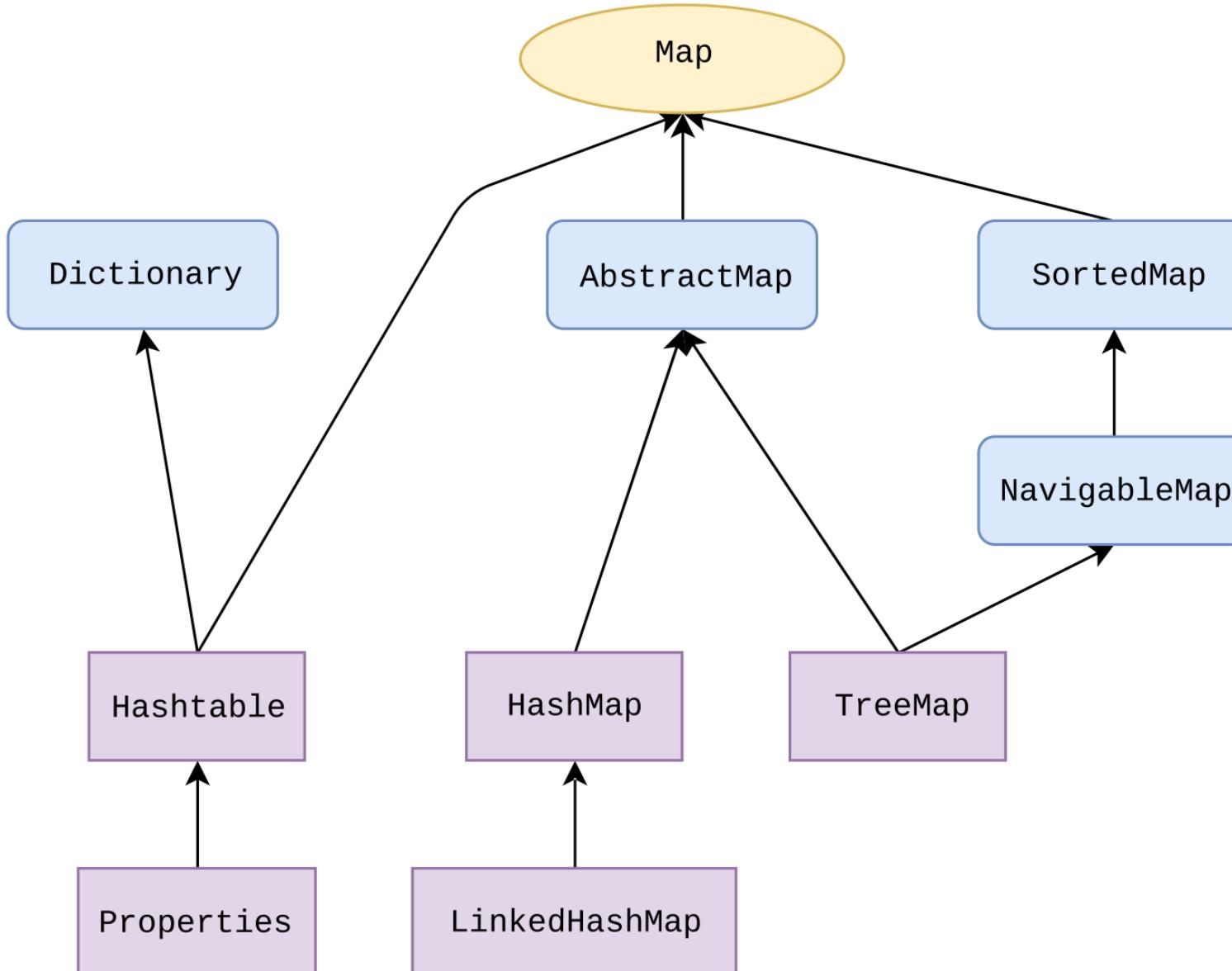
There are three special-purpose Map implementations — [EnumMap](#), [WeakHashMap](#) and [IdentityHashMap](#). [EnumMap](#), which is internally implemented as an array, is a high-performance Map implementation for use with enum keys. This implementation combines the richness and safety of the Map interface with a speed approaching that of an array. If you want to map an enum to a value, you should always use an [EnumMap](#) in preference to an array.

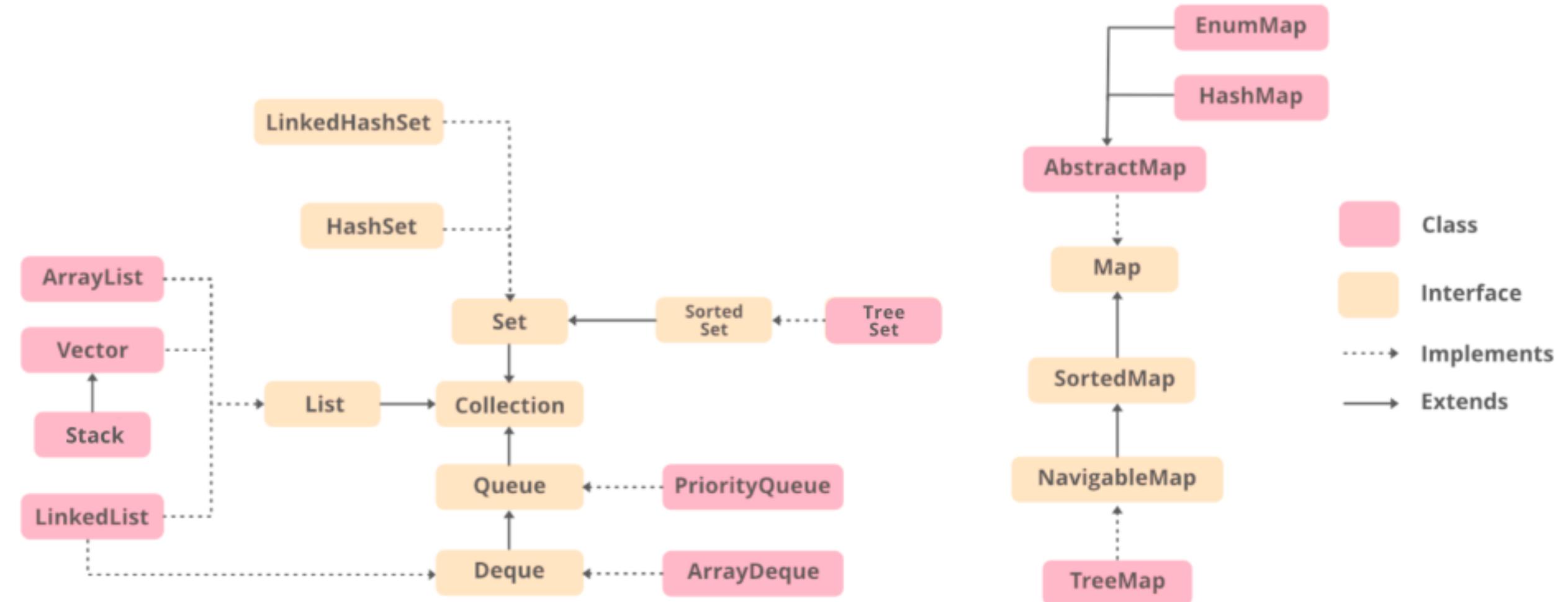
[WeakHashMap](#) is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbage-collected when its key is no longer referenced outside of the [WeakHashMap](#). This class provides the easiest way to harness the power of weak references. It is useful for implementing "registry-like" data structures, where the utility of an entry vanishes when its key is no longer reachable by any thread.

# Collections



# Map







Become a Patreon  
Get Image & PDF Reward

www.falkhausen.de

ENHANCED BY Google

Home

About Diagrams

Java 7

Java 8

Java 10

JavaFX 8

JavaFX 10

Patreon

Paypal

Twitter

Contact

Legal

## Content

This website presents [enhanced Java class diagrams](#). The diagrams are interactive and allow access to the complete API documentation. Please see the video for more information.

Interactive Class Diagrams

Pozriet' nes... Zdieľať

# Java 9

5,000 classes

90,000 pages documentation

Prehrat' na YouTube

- [About Diagrams](#) Diagram motivation, explanations and FAQ
- [Java 7](#) Java 7 SE class diagrams
- [Java 8](#) Java 8 SE class diagrams
- [Java 9](#) Java 10 SE class diagrams
- [JavaFX 8](#) JavaFX 8 class diagrams
- [JavaFX 10](#) JavaFX 10 class diagrams

- [Patreon](#) Become a Patreon
- [PayPal](#) [Donations with PayPal](#)
- [Twitter](#) For feedback and information about updates
- [Contact](#) Personal information
- [Legal](#) Legal notes and Copyright

## News

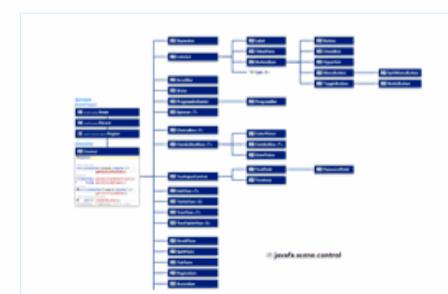
\* 22. Apr 2018



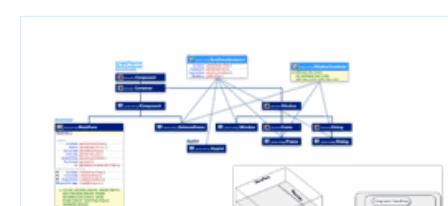
Java FX 9  
[javafx.beans.property.ObjectProperty](#)



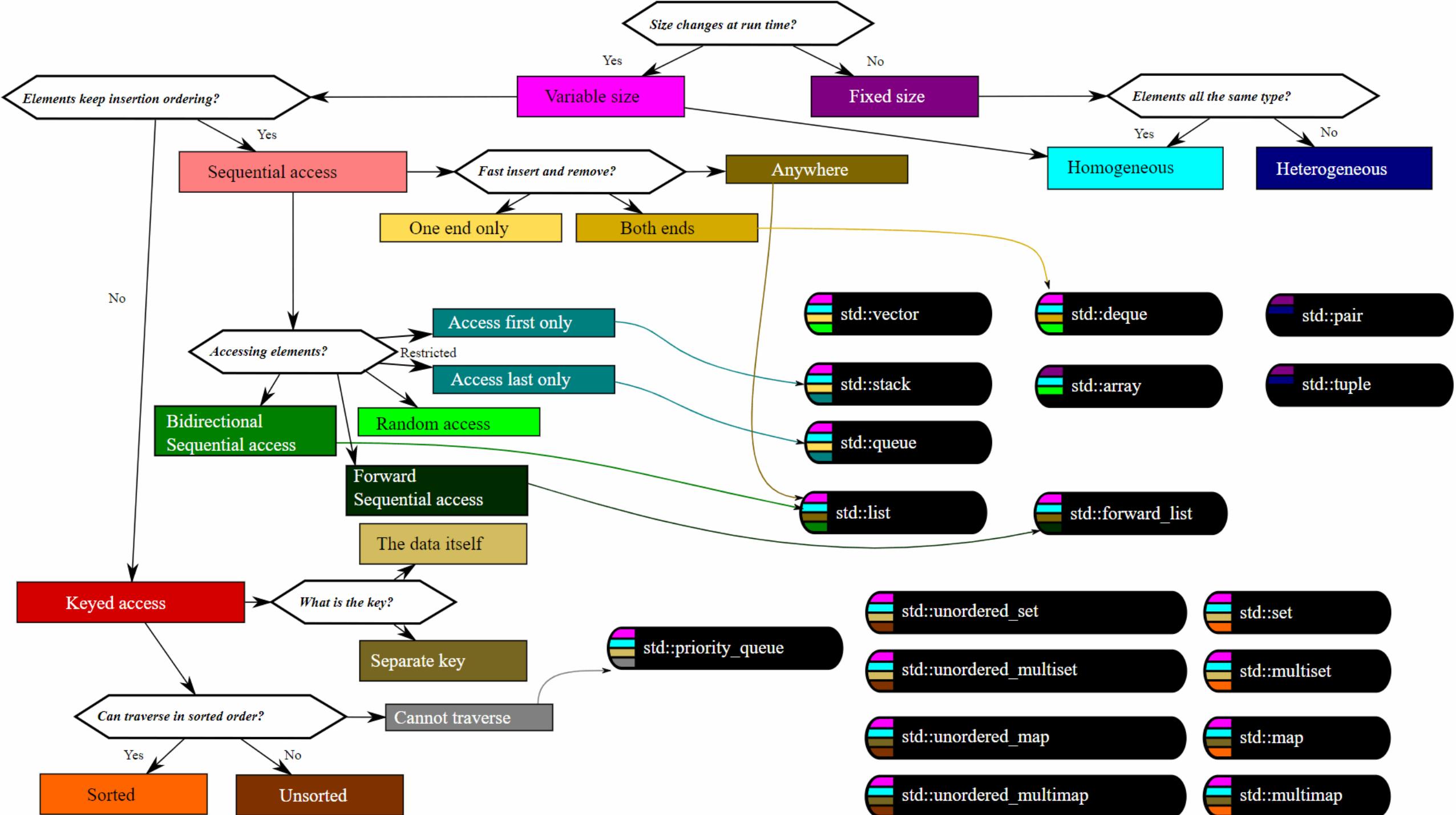
Java 9  
[javax.crypto.Key](#)



Java FX 9  
[Control Hierarchy](#)

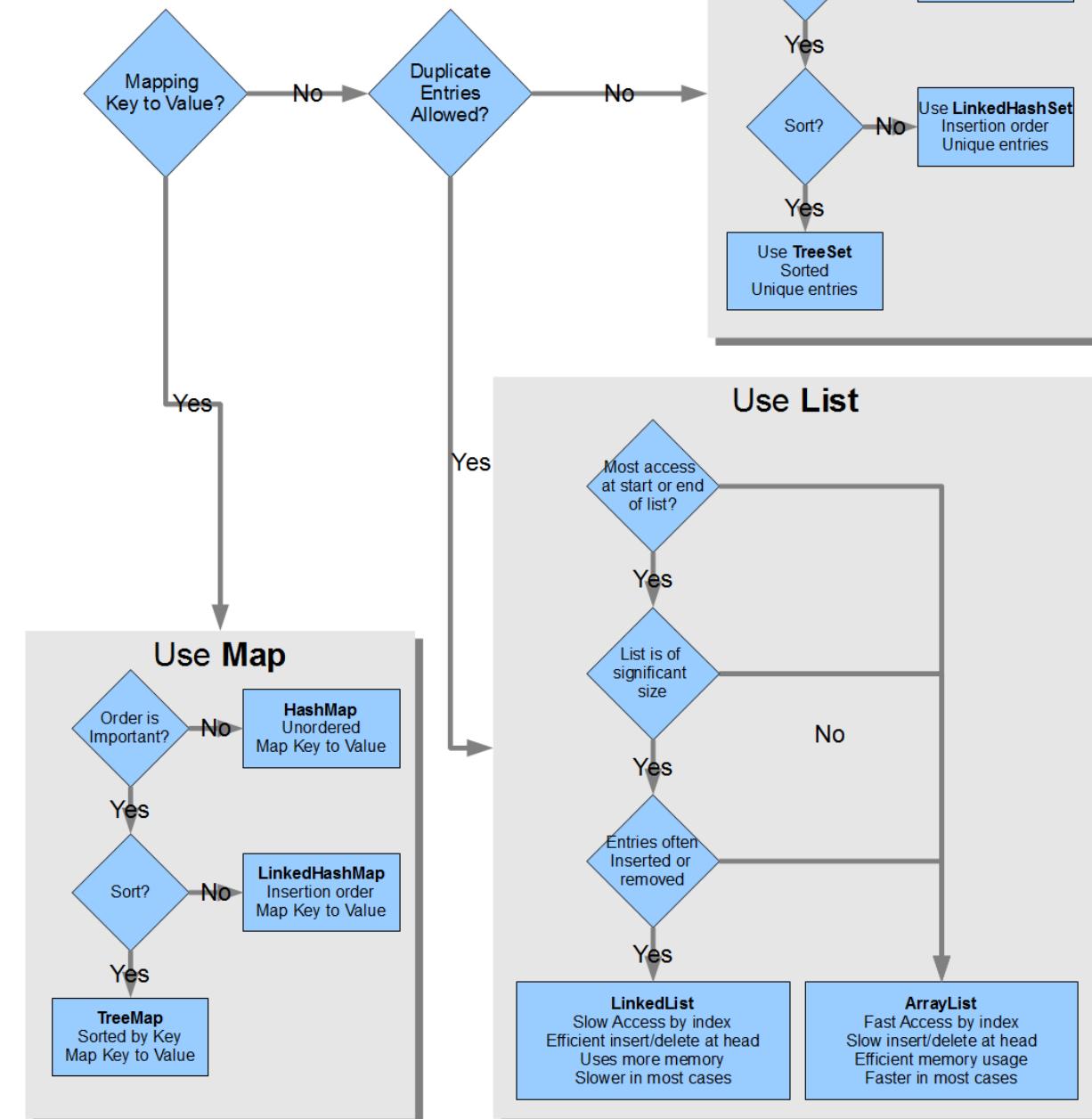


Java 9  
[javax.swing.JRootPane](#)

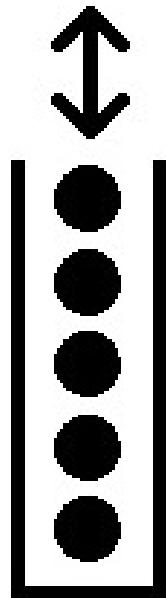


# Ktorú kolekciu mám použiť?

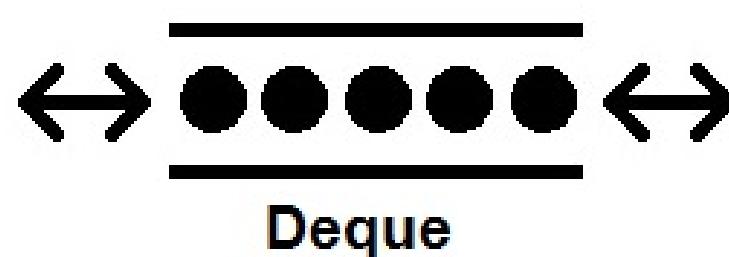
What java.util.collection should I use?



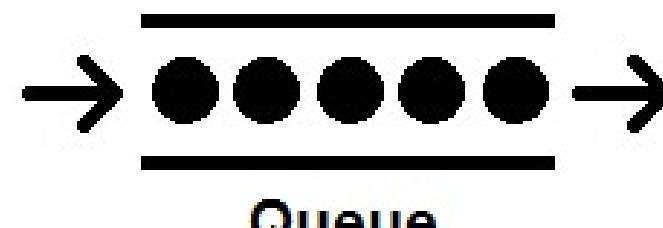
# Základné algoritmy za kolekciami



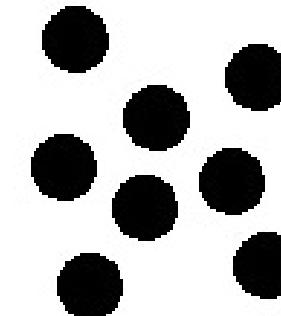
Stack



Deque



Queue



Collection

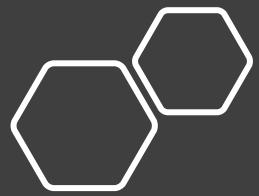


ArrayList



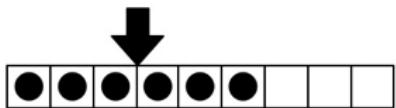
LinkedList

# Vkladanie nového elementu

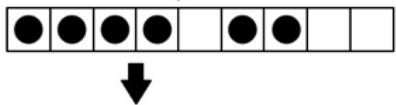
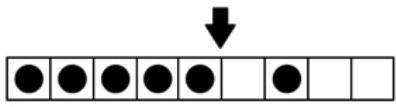


## Insert new element at index three

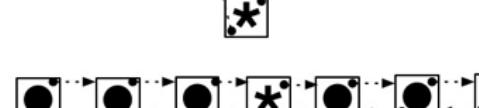
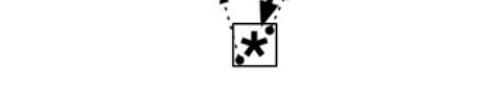
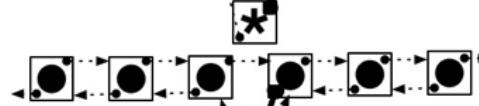
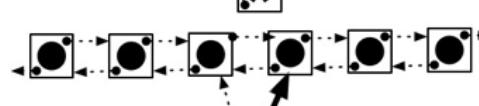
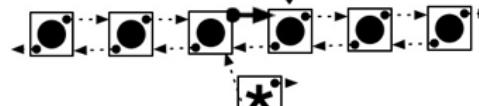
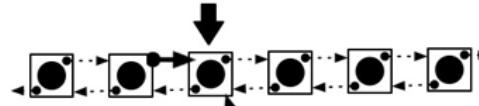
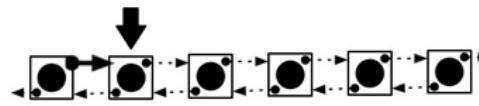
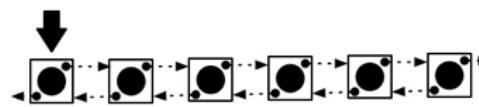
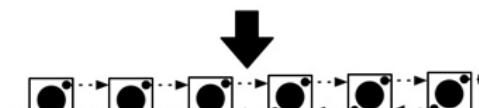
### ArrayList



```
if(size == capacity): Recreate list  
Go to address of last element(index[length - 1]):  
([start address] + (size - 1) * [size of each element])  
while(index <= 3)  
    Move element to right
```



### LinkedList



# Čo sa oplatí prečítať?

## Slovensko a česko

- Albatrosmedia
- Kopp
- Grada
- Wolters Kluwer
- BEN
- Veda

## Zahraničie

- O'Reilly
- Manning
- Packt
- Apress
- Wiley
- No Starch Press

## YouTube tutoriály

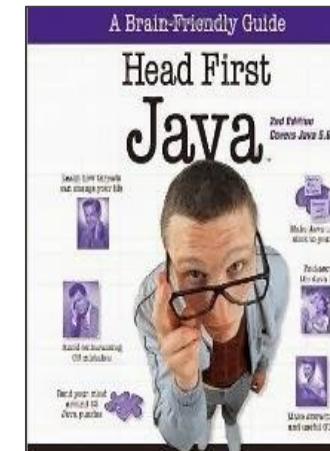
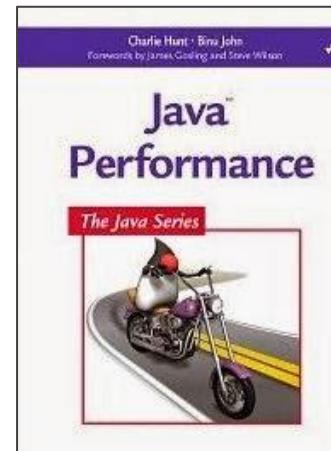
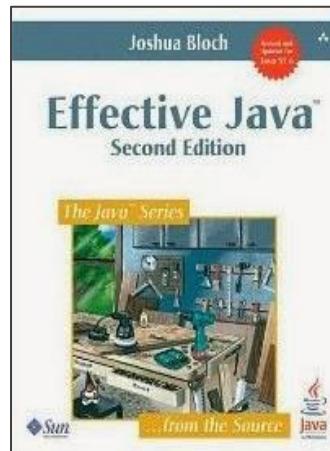
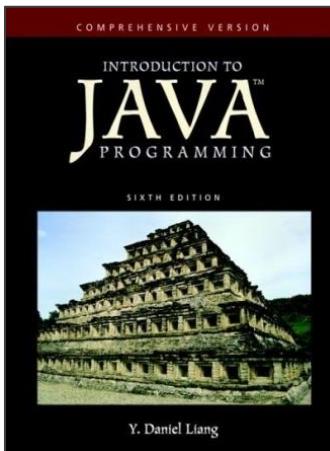
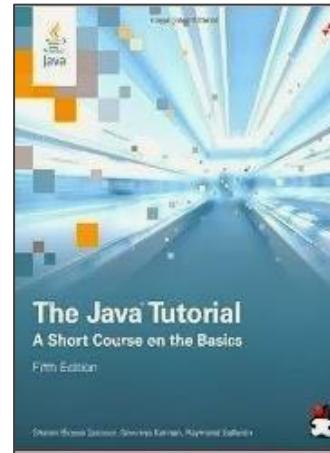
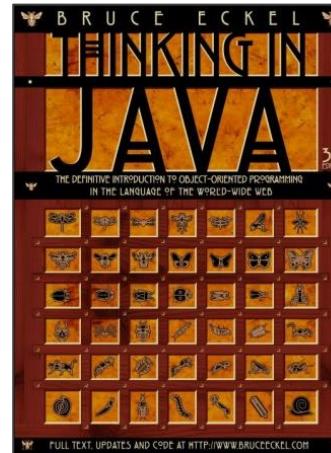
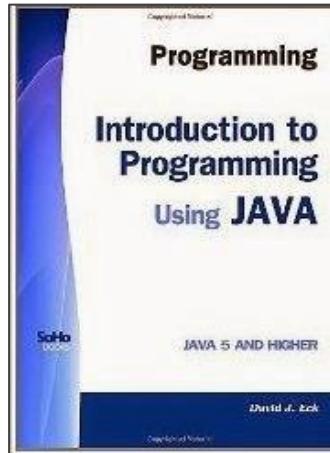
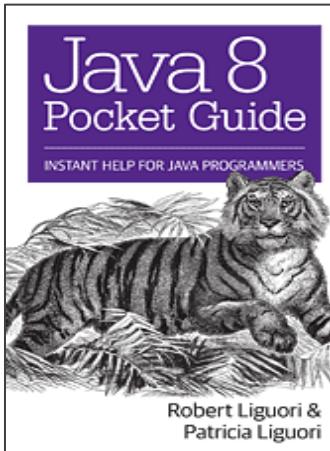
Oracle a O'Reilly

# Čo sa oplatí/neoplatí prečítať SK/CZ



Mistrovství a Kuchárka

# Čo sa oplatí/neoplatí prečítať EN



Head First

## Vývojári

Verejná skupina

Informácie

## Diskusia

Oznámenia

Členovia

Podujatia

Videá

Fotky

Súbory

Hľadať v tejto skupine



Ste člen

Upozornenia

Zdieľať

... Viac



Napísat' príspěvok...

Pridať fotku/vi...

Živé video

Viac



Napište niečo...

Fotka/video

Divácka páry

Označiť priat...

...

## Skratky

Podnikanie na Slove...

2

UK Manazment Externe...

Testovacia firma

VITA - Virtual It Academy

Startupisti

2

Rubyslava

2

Vývojári

▼ Zobrazit viac

## NOVÁ AKTIVITA

 Roland Mondek  
10 h

## POZVAŤ ČLENOV

+ Zadajte meno alebo e-mailovú adresu...



## ČLENOVIA

5 505 členov



## POPIS

Skupina softvérových vývojárov. Táto skupina by mala byť miestom... Zobraziť viac

## TYP SKUPINY

Všeobecné

## VAŠE STRÁNKY



IT Academy



VITA - Virtual It Academy

## KONTAKTY



Evka Rybárska



Jarmila Palenčárová



Stefan Orosi



Ivana Ivka Jasaňová



Hrá Word Blitz



Ivana Pavlíková



Martin Vanko



Lucia Kovačičová

4 h



Lošák Filip



Andrej Nejedlik



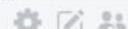
Gabika Zubrikova

## SKUPINOVÉ KONVERZÁCIE



Vytvoriť novú skupinu

Hľadať



...

...

```
52 </div>
53 </body>
54 <script type="text/javascript">
55 <!--
56 var currentImage = "bigImage1";
57 var pages = Math.ceil(photos.length / 9);
58 updatePages();
59 updateAllImages();
60 // document.getElementById('bigImage0').src = 'images/wieksza/' + photos[page * 9];
61 // document.getElementById('bigImage0').style.display = '';
62 changePhotoDescription('1');
63
64
65
66
67
68
69
70
71
72
73
```

# Programátori

• Verejná skupina · 7,2 tis. členov

Člen ▾

+ Pozvať

Informácie

Diskusia

Vybrané

Témy

Ludia

Podujatia

Médiá

Súbory



...



Napíšte niečo...

Živé video

Fotka/video

Anketa

Vybrané



## Informácie

Táto skupina slúži na dohadzovanie si kŕeftíkov a pre hľadačov programátorov / vývojárov.

• Verejná

Členov skupiny a ich príspevky bude vidieť ktokoľvek.





Miroslav



## JavaGroup

Verejná skupina · 450 členov

Člen ▾

+ Pozvť



Informácie

Diskusia

Vybrané

Témy

Ludia

Podujatia

Médiá

Súbory



Napíšte niečo...

Živé video

Fotka/video

Anketa

Vybrané



### Informácie

Ziju este nejaki Javisti? Dajte o sebe vediet

Skupina nie je určená pre zadávanie inzerátov a HR / recruiting. Dakujeme za pochopenie.

Verejná

Členov skupiny a ich príspevky bude vidieť



Home

PUBLIC

Questions

**Tags**

Users

COLLECTIVES

Explore Collectives

FIND A JOB

Jobs

Companies

TEAMS

Create free Team

# Tags

A tag is a keyword or label that categorizes your question with other, similar questions. Using the right tags makes it easier for others to find and answer your question.

[Show all tag synonyms](#)

java

java

Java is a high-level object oriented programming language. Use this tag when you're having problems using or understanding the language itself. Thi...

1827413 questions    419 asked today, 2408 this week

javascript

For questions regarding programming in ECMAScript (JavaScript/JS) and its various dialects/implementations (excluding ActionScript). Note...

2335556 questions    779 asked today, 4877 this week

javafx

The JavaFX platform enables developers to create and deploy Graphical User Interface (GUI) applications that behave consistently...

36355 questions    6 asked today, 50 this week

java-8

for questions specific to Java 8 which is version 8 (internal number 1.8) of the Java platform, released on 18 March 2014. In most cases, you should also...

22076 questions    9 asked today, 40 this week

java-stream

for questions related to the use of the Stream API. It was introduced in Java 8 and supports functional-style operations on streams of values, such...

10293 questions    5 asked today, 26 this week

java-native-interface

The Java Native Interface (JNI) gives both the ability for JVM implementations to run system native code and the ability for native code t...

9404 questions    12 asked this week, 38 this month

rx-java

RxJava – Reactive Extensions for the JVM – a library for composing asynchronous and event-based programs using observable sequence...

6796 questions    6 asked this week, 27 this month

javascript-objects

for questions related to JavaScript objects.

6151 questions    20 asked this week, 118 this month

java.util.scanner

A simple text scanner in the JDK which can parse primitive types and strings using regular expressions.

javafx-8

JavaFX 8 (previously named JavaFX 3) introduces a new API for JavaFX technology. JavaFX 8 supports 3D and brings up a Retina-Display Support. It ...

java-me

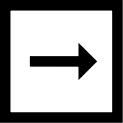
Java Platform, Micro Edition, or Java ME, is a Java platform designed for embedded systems.

facebook-javascript-sdk

Facebook's JavaScript SDK provides a rich set of client-side functionality for accessing Facebook's server-side API calls. It can collaborate with any SDK...

Priestor pre vaše otázky.

Mrkni na náš YouTube kanál a daj odber

 [WWW.YOUTUBE.COM/C/IT-ACADEMYSK](https://www.youtube.com/c/IT-ACADEMYSK) 