JAVA is the new BLACK

GENERYKI | LAMBDY | STREAMY | VAVR
KOLEKCJE | WZORCE PROJEKTOWE | TESTY
GUAVA | COMPLETABLE FUTURE

JAVA FAKTURA

JUG dla początkujących:
[praktykanci, testerzy, juniorzy, początkujący „regularzy”]

Poruszamy najtrudniejsze aspekty fundamentalnych zagadnień

Spotkania co dwa tygodnie w sezonach

Wspieramy i wskazujemy dalsze kierunki samodzielnego rozwoju

Eksperymentujemy – wszystko może się zmienić, liczymy na Wasz feedback!

Starszych programistów zapraszamy do dołączenia!

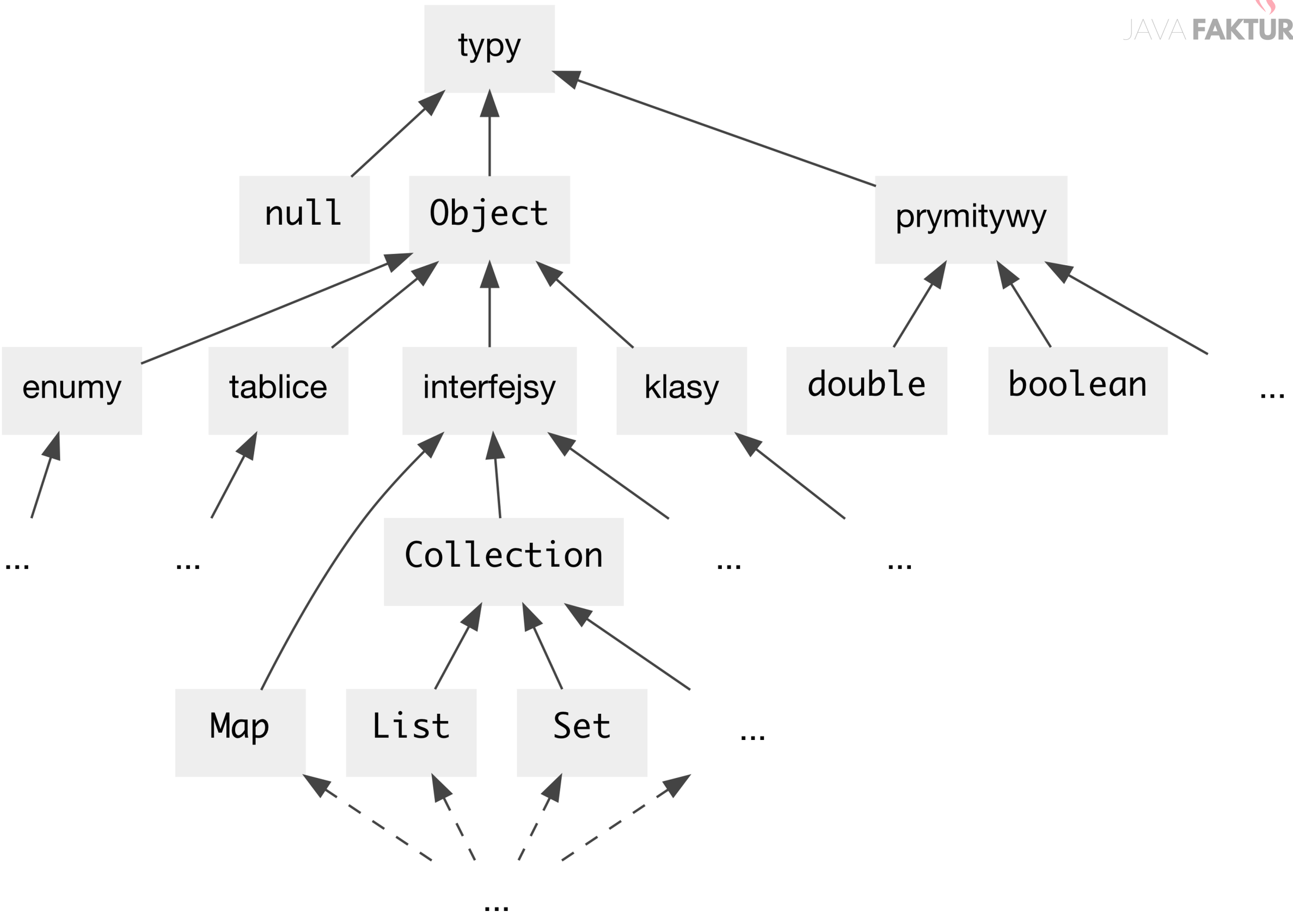
s01e01: Kolekcje, typy generyczne, lambdy

JAVA FAKTURA

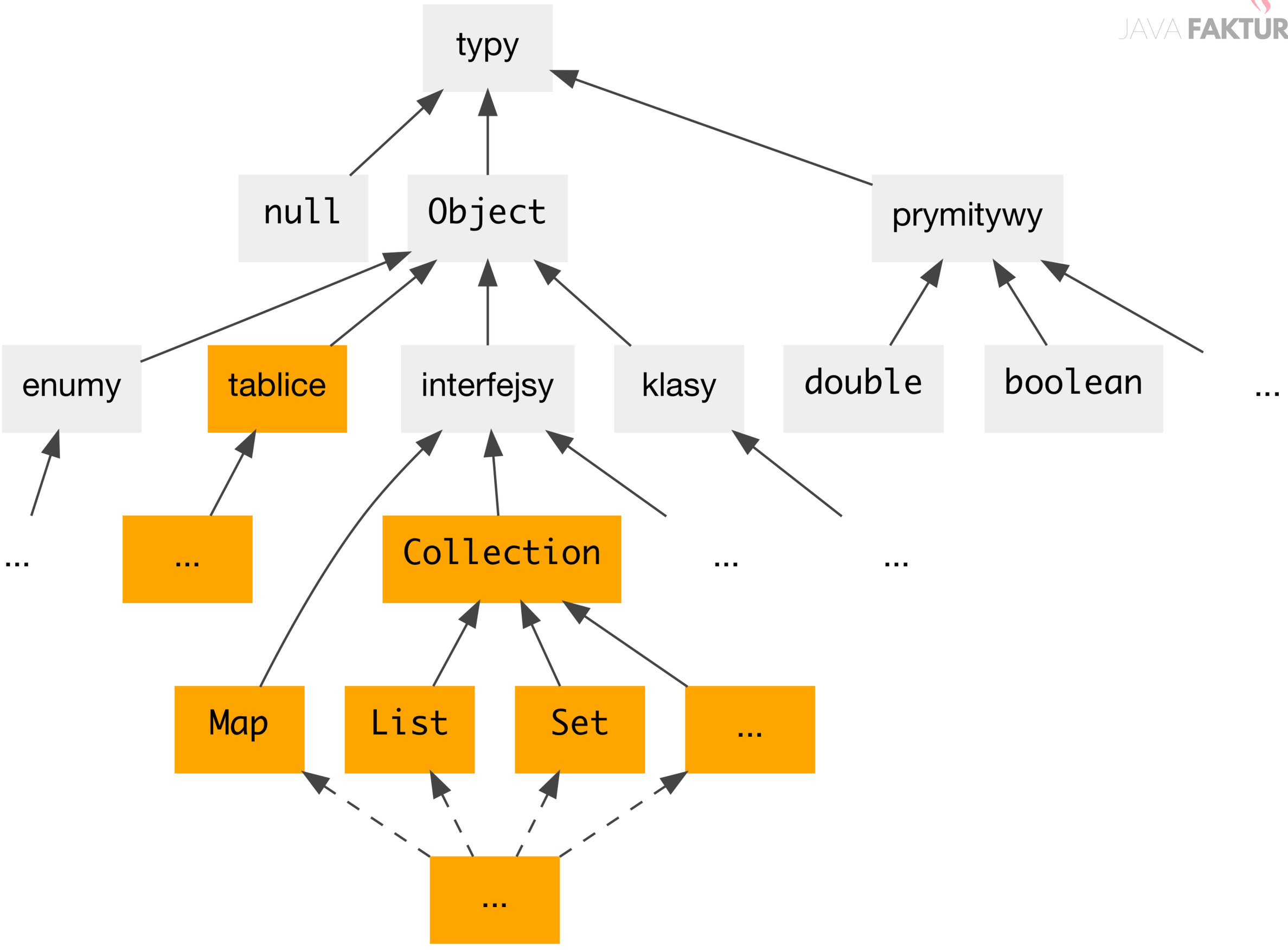
Typy generyczne, `T extends R`, `T super R` **etc.**
– kowariancja i kontrawariancja w Javie

Dostępne typy kolekcji w Javie
– budowa i zastosowania

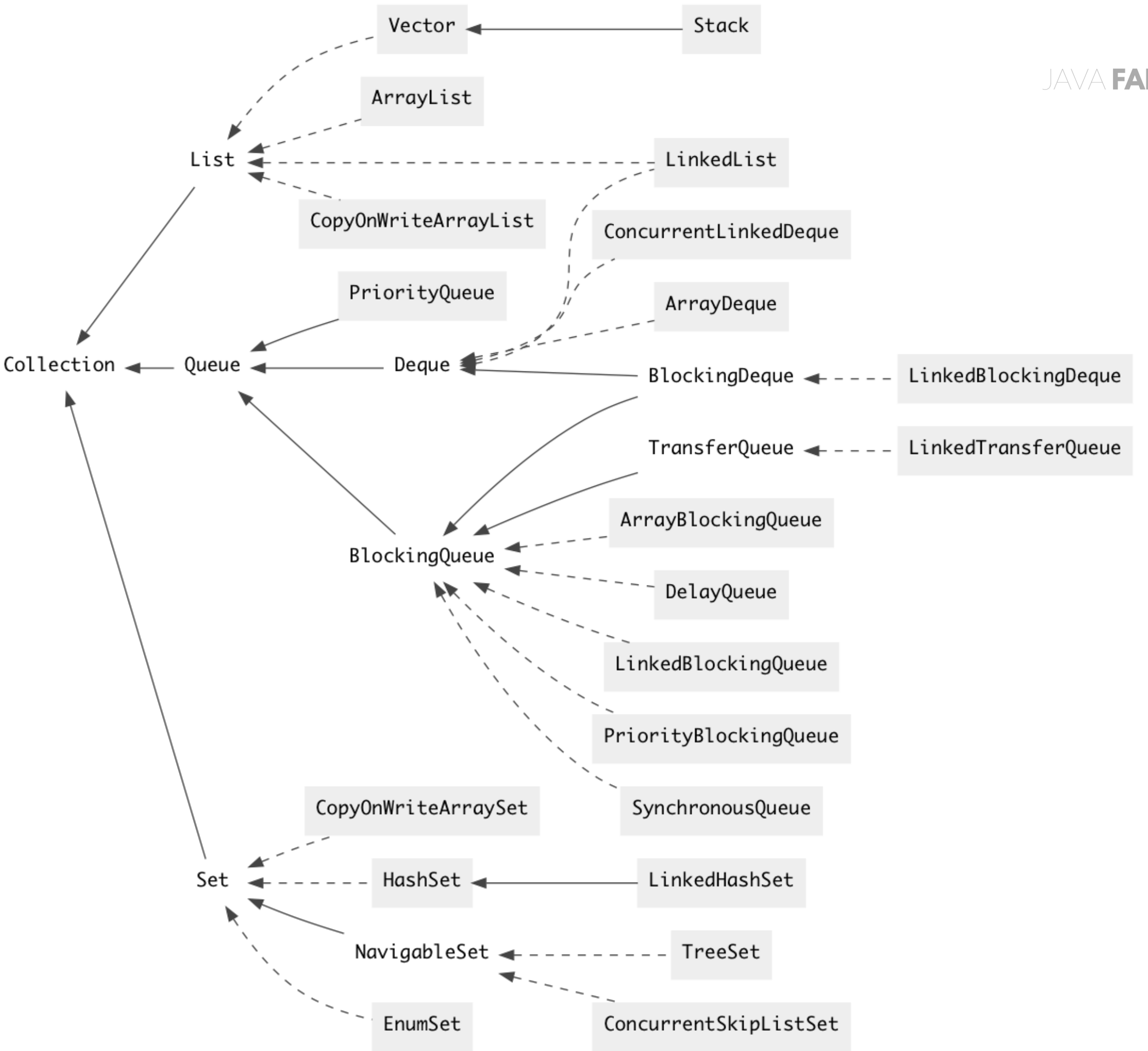
Lambdy
– wprowadzenie przed następnym spotkaniem


typy
null
Object
prymitywy
enumy
tablice
interfejsy
klasy
double
boolean
...
...
...
...
...
Collection
Map
List
Set
...
...

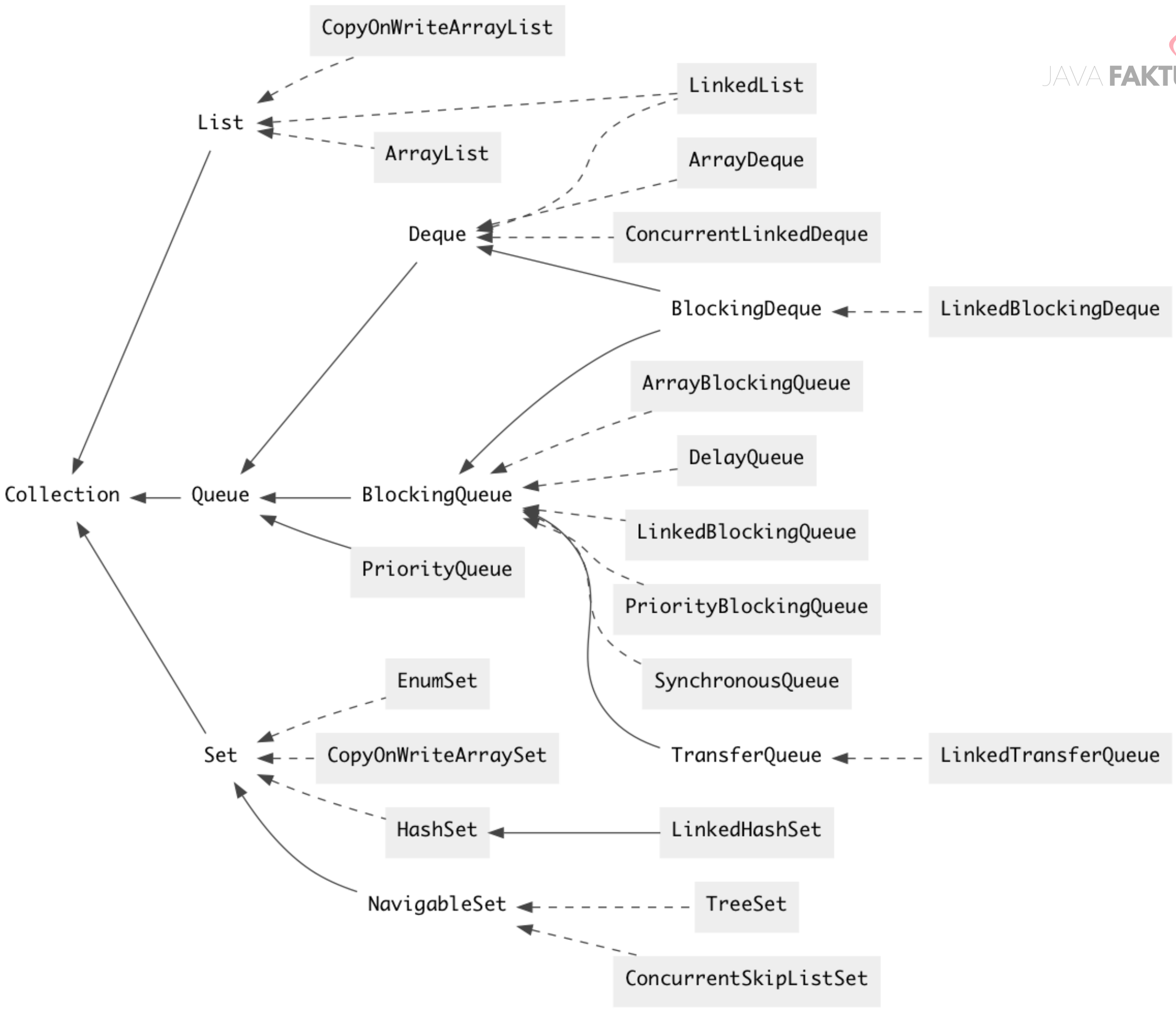
JAVA FAKTURA



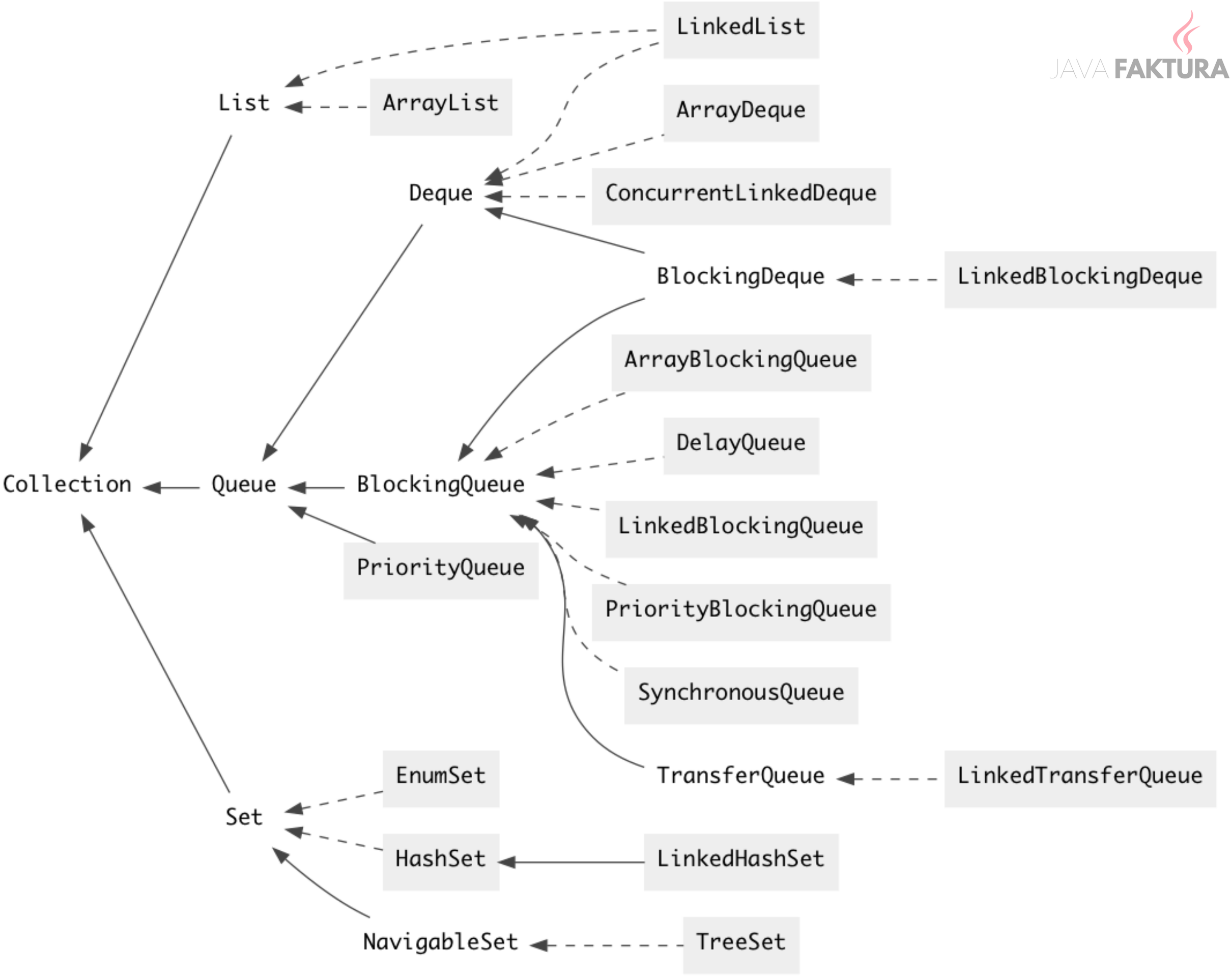
JAVA FAKTURA

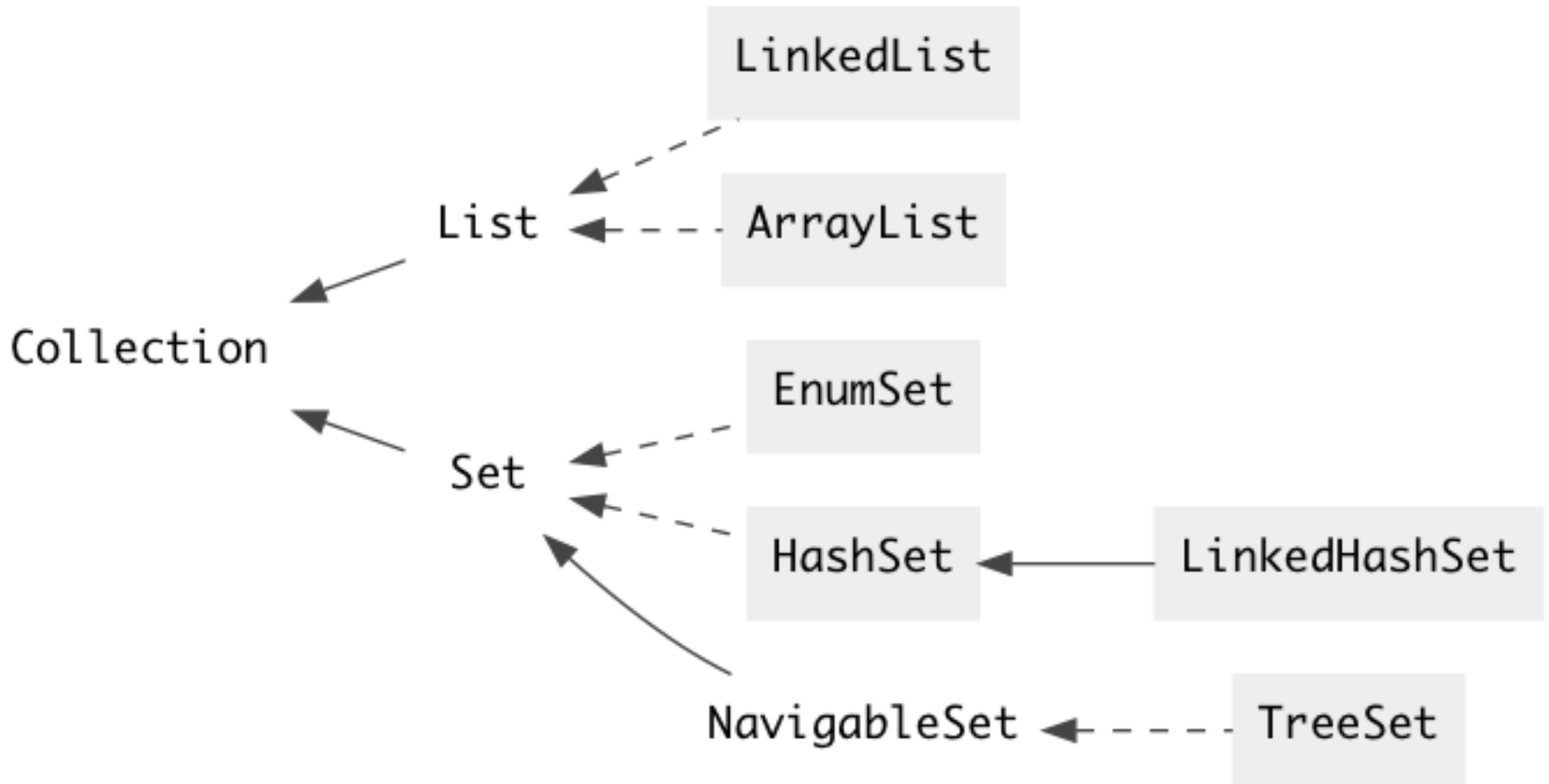


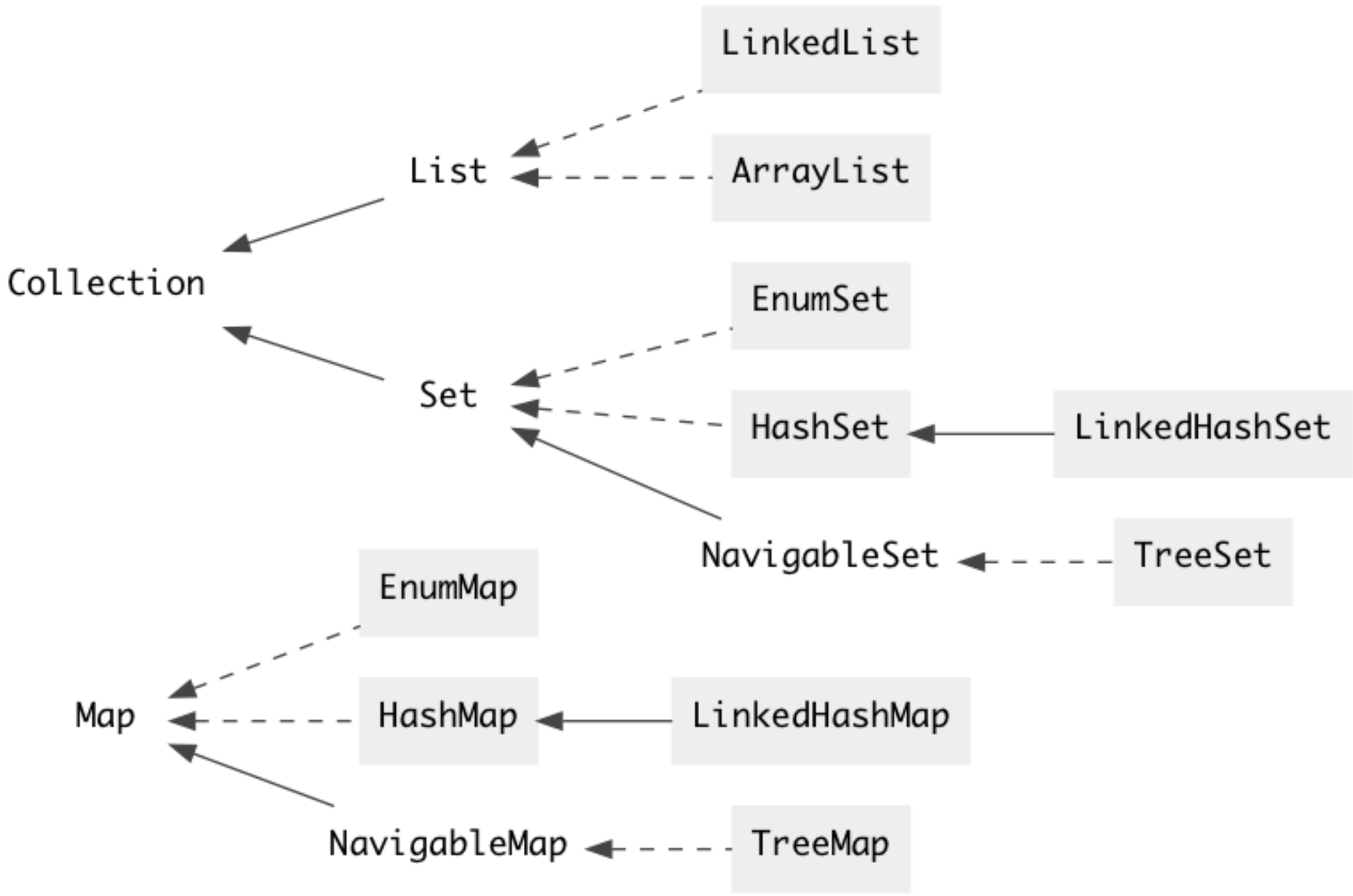
JAVA FAKTURA

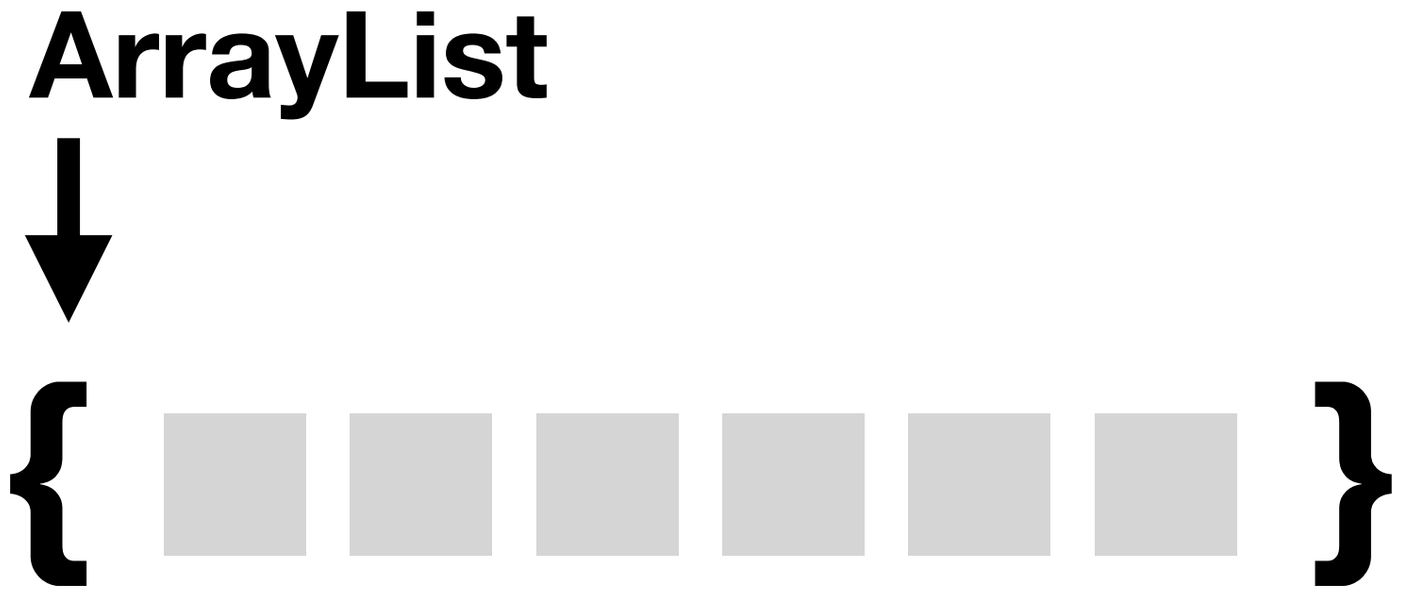


JAVA FAKTURA




LinkedList
List
ArrayList
Collection
EnumSet
Set
HashSet
LinkedHashSet
NavigableSet
TreeSet


LinkedList
List
ArrayList
Collection
EnumSet
Set
HashSet
LinkedHashSet
NavigableSet
TreeSet
EnumMap
Map
HashMap
LinkedHashMap
NavigableMap
TreeMap


ArrayList


LinkedList

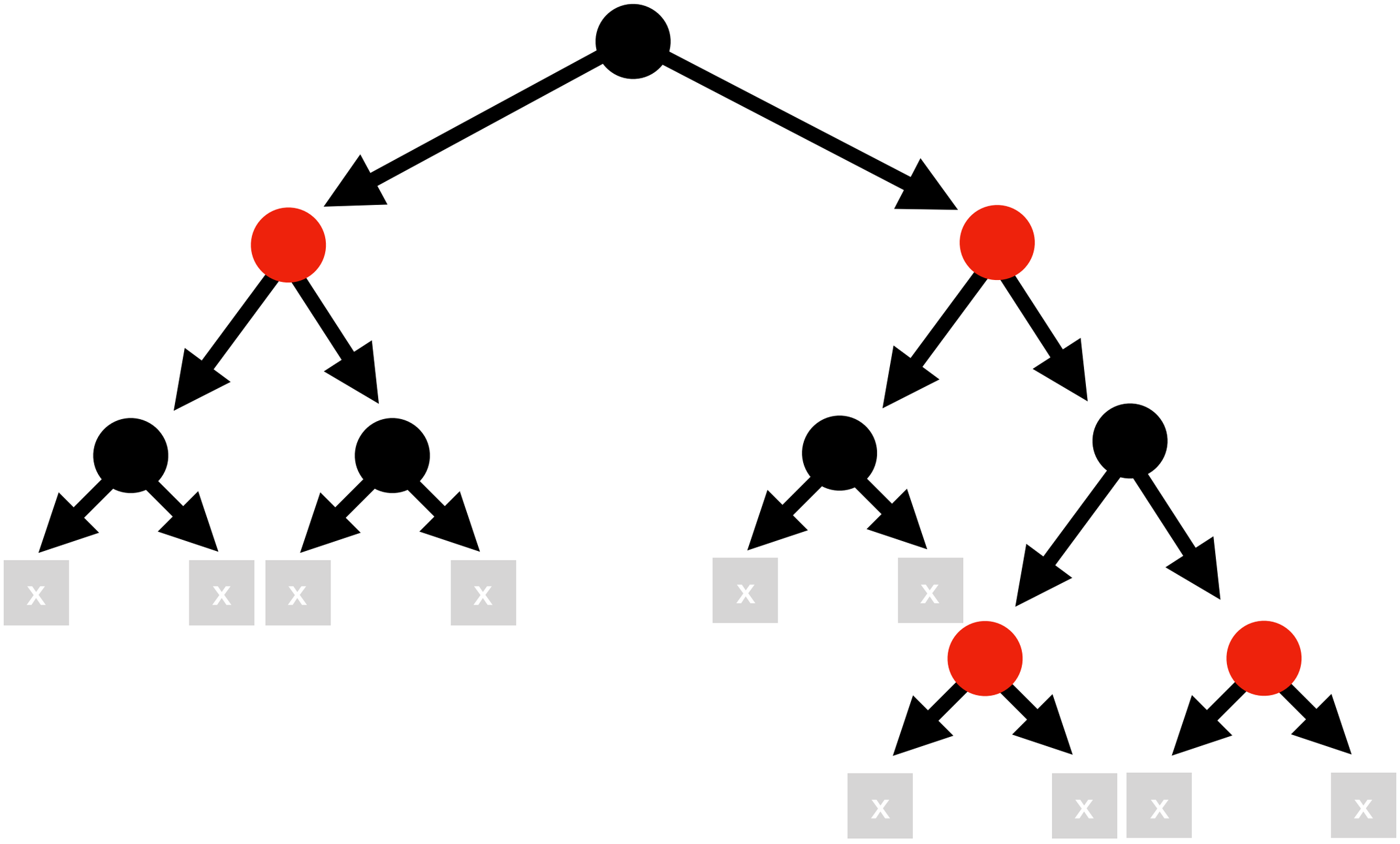
ArrayList	LinkedList
Opakowana tablica	Obiekty-ogniwa z wartościami i wskazaniami na następny i poprzedni
Minimalna ilość zajmowanej pamięci	Stosunkowo duża ilość zajmowanej pamięci
Dostęp do „środka” po indeksie tablicy – zawsze względnie szybki	Dostęp do „środka” poprzez „skakanie” po ogniwach listy – powolne
Dodawanie i usuwanie elementów wiąże się ze zmianą rozmiaru tablicy	Dodawanie i usuwanie elementów wymaga jedynie aktualizacji wskazań „sąsiadów”
Iteracja względnie szybka w obu przypadkach	
Najlepszy wybór ogólnego użytku	**Dodatkowo implementuje Deque** **Stanowi podstawę innych kolekcji z Linked* w nazwie**

Przedwczesna optymalizacja źródłem wszelkiego zła

Kod musi być przede wszystkim czytelny.
Wydajność należy przede wszystkim mierzyć, nie przewidywać.
Dopóki nie trzeba jej zwiększyć, nie trzeba jej mierzyć, ergo nie trzeba jej poprawiać.

Tree*

JAVA FAKTURA



Jedno proste spostrzeżenie – ilość przejść od korzenia drzewa do dowolnej wartości
w węzłach (liście nie zawierają wartości) jest ***bardzo mała*** *w stosunku do ilości przechowywanych wartości*

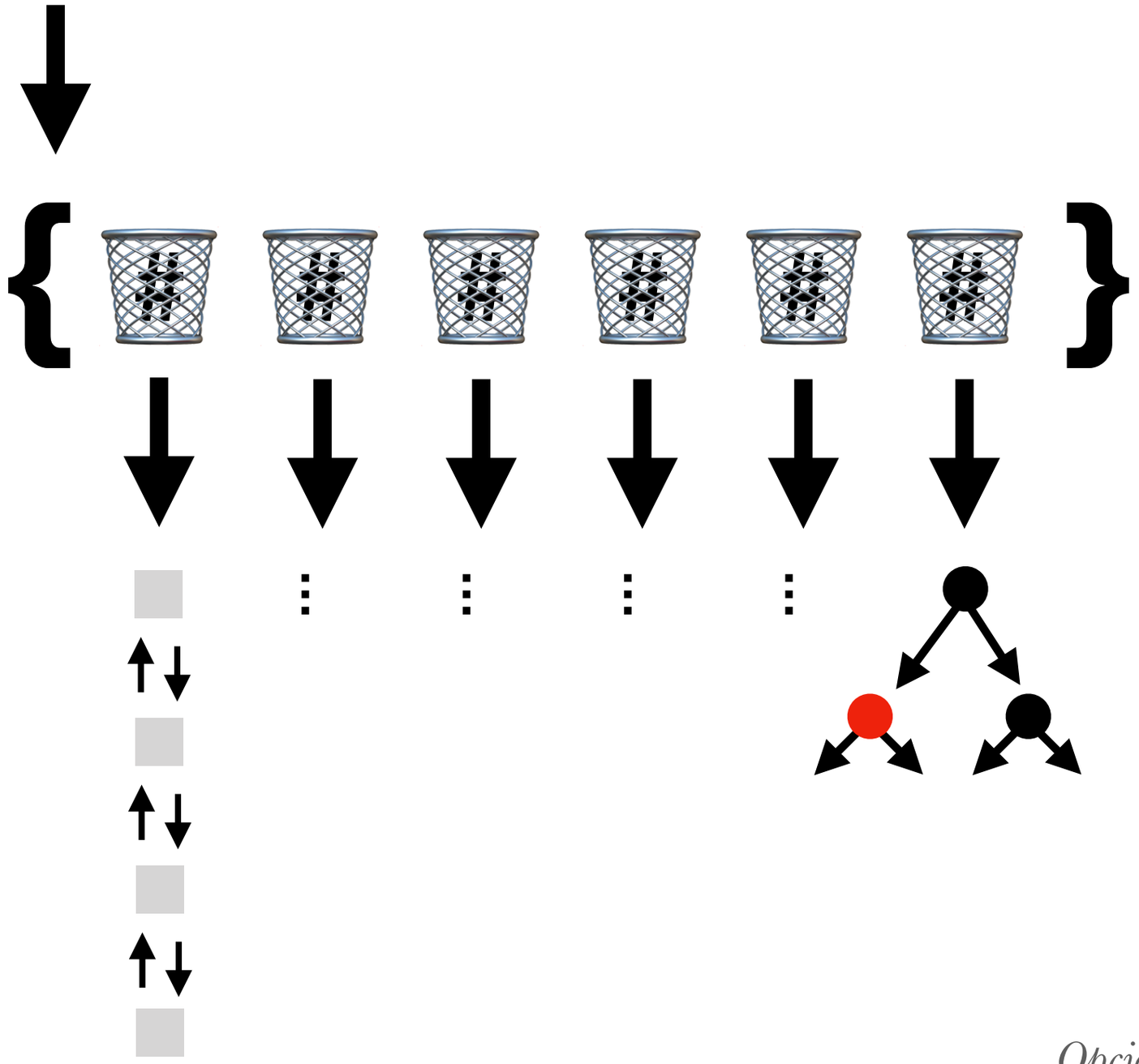
Wydajność list (mniej = lepiej)

JAVA FAKTURA

	Get	Add	Insert	Iterate	Remove
ArrayList	1	1	40	1	40
LinkedList	5800	1	350	2	325
TreeList (commons)	3	5	1	2	1

https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/list/TreeList.html

HashMap



Opcjonalny TreeNode -> nowość Javy 8
Jeden bucket „zawiera” wiele hash-code’ów, zgodnie z wielkością tablicy bucketów
Dobra mapa powinna być zbalansowana (mieć podobną ilość wartości w każdym buckecie),
a to wynika z dobrej funkcji hashcode()