

Java

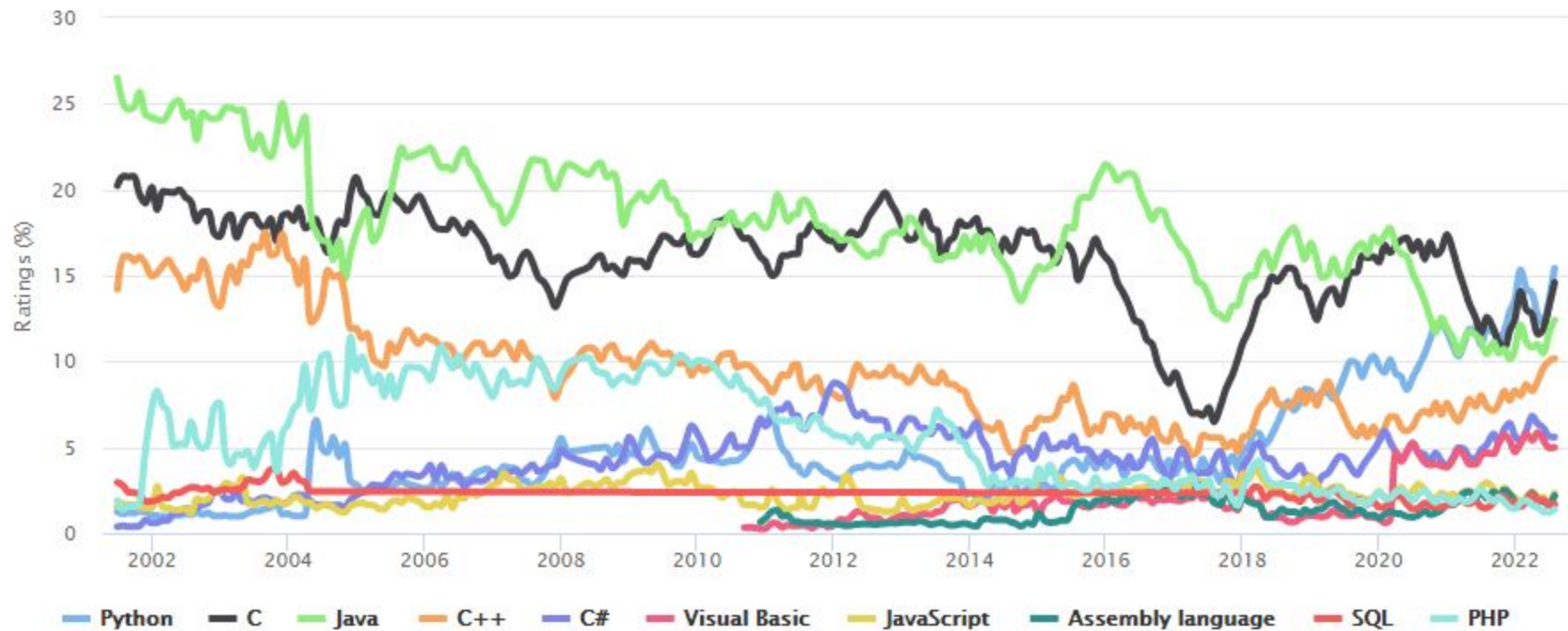
Matthias Colin

Programmation Orienté Objets

- POO / OOP
- C++
- Java
- .NET (C#, VB.NET, ..)
- Python
- JavaScript, TypeScript, Google App Script
- Php

TIOBE Programming Community Index

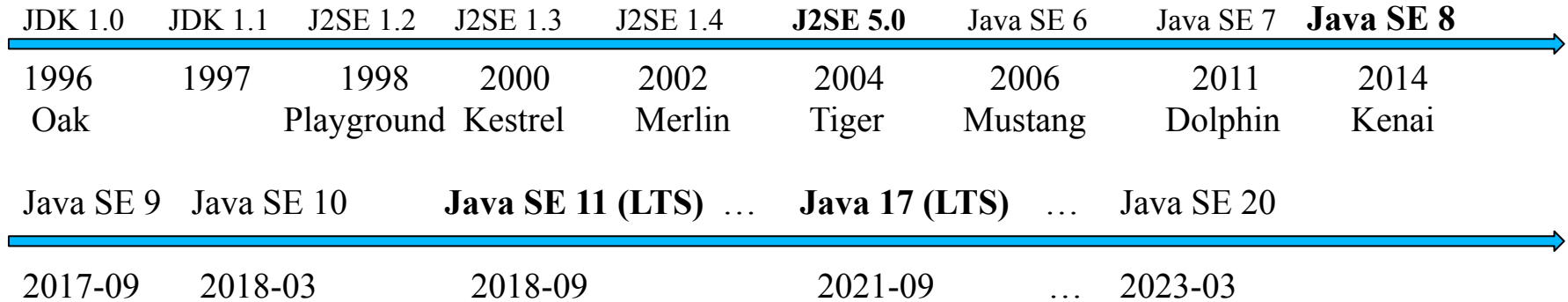
Source: www.tiobe.com



Java éditions

- Java SE (Standard Edition)
 - JVM : Java Virtual Machine
 - JRE : Java Runtime Environment (JVM + lib Java SE)
 - JDK : Java Development Kit (JRE + tools javac, jar, ...)
 - Providers: Oracle et Openjdk
- Java ME (Mobile Edition)
- Java EE / Jakarta EE (Entreprise Edition)
 - collection de spécifications
 - servlet (http), jsp (Java Server Page), el, jstl
 - JPA (Java Persistence API) : RDBMS + SQL
 - JAX-WS : web services (SOAP+WSDL), axé XML
 - JAX-RS : API Rest avec data XML ou JSON
 - JNDI : annuaire
 - Java Bean Validation
 - ...

Java SE



Vocabulary

- **JVM:** Java Virtual Machine (java or java.exe)
 - execute Java Bytecode
- **JRE**
 - JVM + libraries included in the language
- **JDK**
 - JRE + tools
 - javac: compiler
 - jar: packager
 - javadoc: documentation generator
 - jshell: Java interpreter
 - jconsole: monitoring
 -

Tools

- java (JVM)
- javac (compiler)
 - *.java (source) => *.class (bytecode)
- jar (package)
 - contains: bytecode, resource, other jar
 - examples:
 - appli.jar, library.jar
 - java -jar appli.jar
 - webapp.war (to be deployed in application server)
- javadoc (documentation)
- many others ...

Project Manager

- Managers
 - Ant (deprecated)
 - **Maven**: default configuration file **pom.xml**
 - download dependencies
 - plugins
 - **Gradle**: default configuration file **build.gradle**
 - download dependencies
 - plugins
 - many languages, speed
- Dependencies
 - Maven Repository
 - Maven Central

IDE: Integrated Development Environment

- **IntelliJ Idea** (JetBrains)
- Eclipse
- Netbeans
- VS Code

Java/Jakarta EE

JDK 1.0	JDK 1.1	J2SE 1.2	J2SE 1.3	J2SE 1.4	J2SE 5.0	Java SE 6	Java SE 7	Java SE 8	Java SE 11		
1996	1997	1998	2000	2002	2004	2006	2011	2014	2018		
		J2EE 1.2	J2EE 1.3	J2EE 1.4	JEE 5	JEE 6	JEE 7	JEE 8	Jakarta EE 8,9,10		
		1999	2001	2003	2006	2009	2013	2017	2018		
Servlets :		2.2	2.3	2.4	2.5	3.0	3.1	4.0			
JSP :		1.1	1.2	2.0	2.1	2.2	2.3				
JSTL :			1.0	1.1	1.2	-	-				
EL :					2.1	2.2	3.0				
JPA :					1.0	2.0	2.1	2.2	3.0	3.1	
Bean validation :					1.0		2.0				
Outils :		JDBC	WS	JAXWS	JAXRS	JSON	JSON-B				
		JNDI	JSF	SAAJ	WebProfile	WebSocket					

Gestionnaire de projet (2)

- Organisation projet (Maven et Gradle)
- myproject
 - pom.xml or build.gradle
 - src
 - main
 - java
 - resources
 - test
 - java
 - resources

Maven Lifecycle

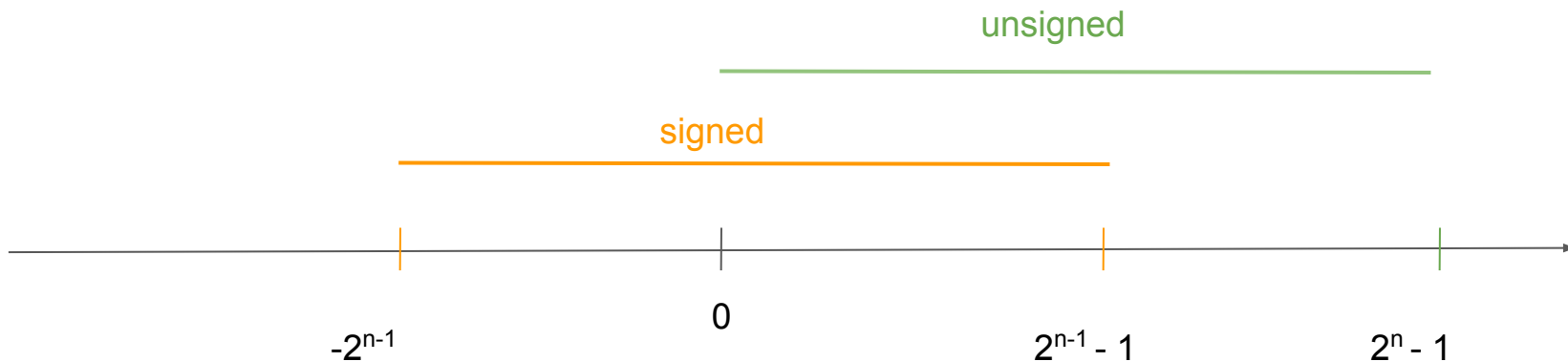
- clean
- compile: compile source code from src/main/java
- test-compile: compile source code from src/test/java
- test: execute all tests
- package: package solution

Variables et Opérateurs

- types primitifs
 - entiers : **short**, **int**, **long** : 4, -5 => +, -, *, /, %
 - flottants : **float**, **double** : 1.23, 3.45E24, NaN, Inf => +, -, *, /, %
 - **boolean** : true, false
 - **char** : 1 caractère : 'A'
 - **byte** : 1 octet (donnée binaire)
- types objets
 - **String** : "Paris" => +
- opérateurs de comparaison
 - égalité : ==, !=
 - types primitifs : contenu
 - types objets : adresse mémoire (**equals** pour = de contenu)
 - ordre : <, <=, >, >= (primitifs)
 - combinaison: !, &&, || (resp. not, and, or)
- autres calculs numériques: class java.lang.Math

Integers

- integer stored in n bits: short (16), int (32), long (64)
- by default signed, can be interpreted as unsigned
 - $n=16$, signed -32768 to 32767 unsigned 0 to 65535
- 11111111 : 255 (unsigned), -1 (signed)



Autoboxing

Pour chaque type primitif => 1 type objet

- short ⇔ Short
- int ⇔ Integer
- long ⇔ Long
- float ⇔ Float
- double ⇔ Double
- boolean ⇔ Boolean
- char ⇔ Character

Control Flow

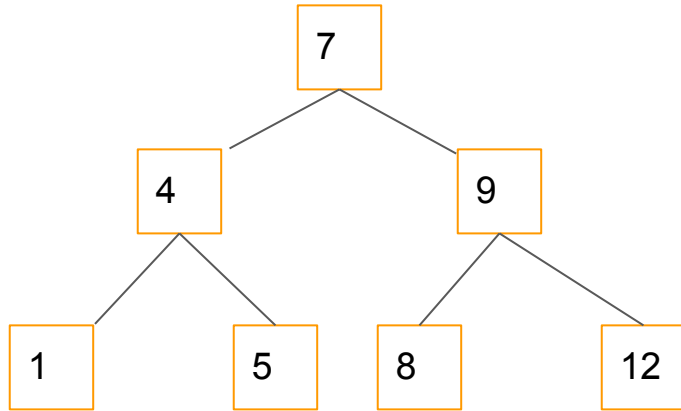
- Conditional
 - if ... else
 - switch ... case (with pattern match since Java 16)
- Loop
 - for(init; cond; incr) => index
 - for “foreach” => value
 - while
 - do ... while

Arrays

- arrays are pseudo objects in Java
 - no methods
 - 1 pseudo attribute length
- declaration:
 - `int[] myarray;`
 - `int myarray[];`
- static
 - `int[] myarray = {1, 2, 4, 5};`
- dynamic
 - `int[] myarray = new int[100];`
- toolbox to handle arrays: Arrays

Binary search / binary tree

array = [1, 4, 5, 7, 8, 9, 12]



Fonctions et méthodes

- rangée dans une classe (obligatoire)
- visibilité :
 - public : tout le monde
 - protected : package + classes filles
 - (no keyword) : package private
 - private : intérieur de la classe uniquement

Class example: Movie

Movie
title: String year: int duration: short

Constructor(s)

```
var movie = new Movie(title: "Interstellar", year: 2014, (short) 169);
```

movie = {Movie@1804}

> f title = "Interstellar"

f year = 2014

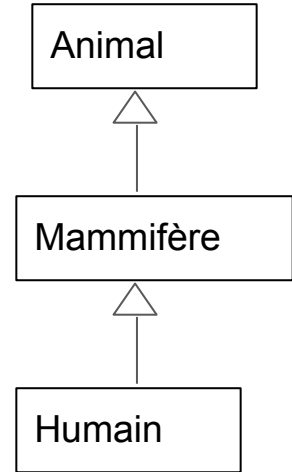
f duration = 169

Classes

- constructeur(s)
 - par défaut implicite si aucun constructeur écrit
 - explicite(s)
- attributs
 - valeur par défaut
 - 0 pour les attributs numériques
 - false pour les booléens
 - null pour les objets
 - Encapsulation
 - attribut privé
 - getter et/ou setter
- lombok pour gérer getter/setter, constructeur, ...

Héritage

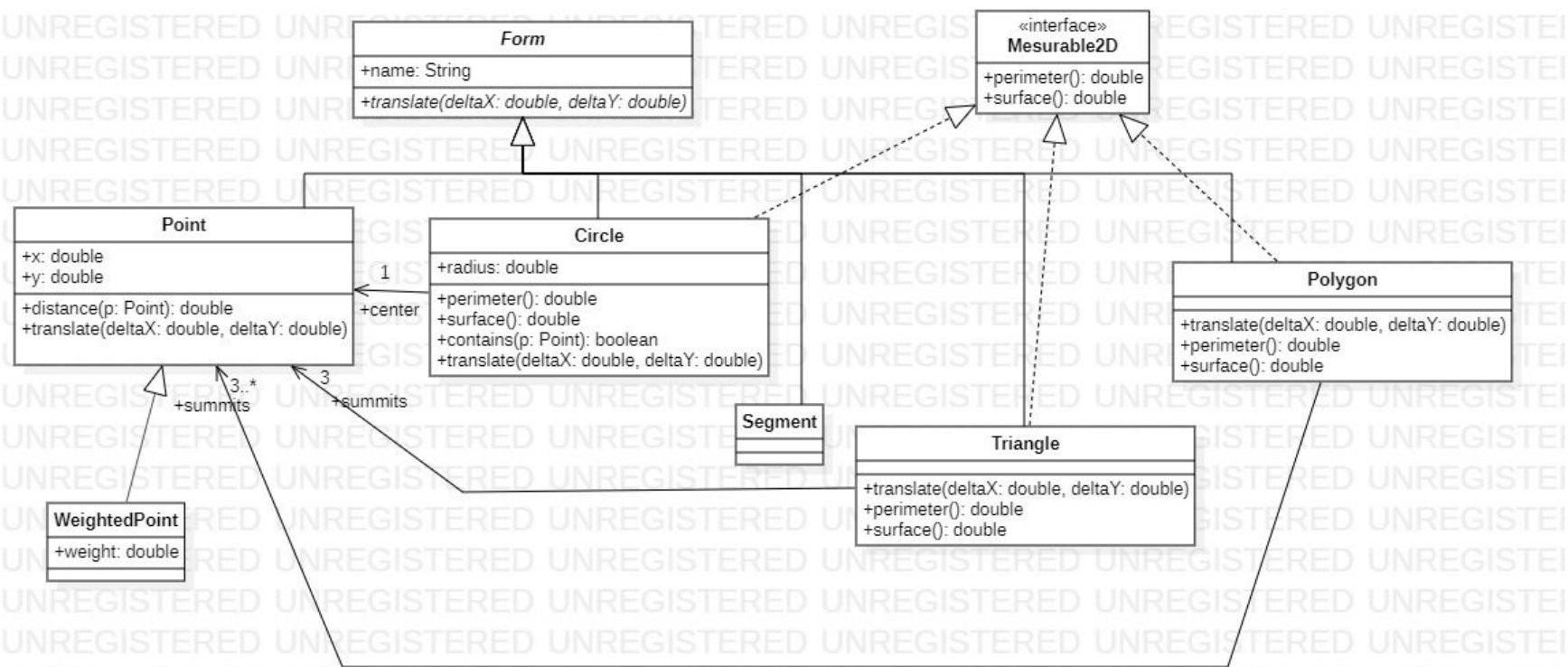
- 1 humain est un mammifère
- 1 mammifère **est** un animal
- 1 mammifère **peut être** un humain
- classe Humain hérite de la classe Mammifère
- classe Humain **spécialise** la classe Mammifère
- classe Mammifère **généralise** la classe Humain
- principe de substitution de Liskov & Wing (LSP)





Visibility

- private : UML - : only in this class
- : UML ~ : (package private) : only in this package
- protected : UML # : package + children classes
- public : UML + : everyone











Model geometry



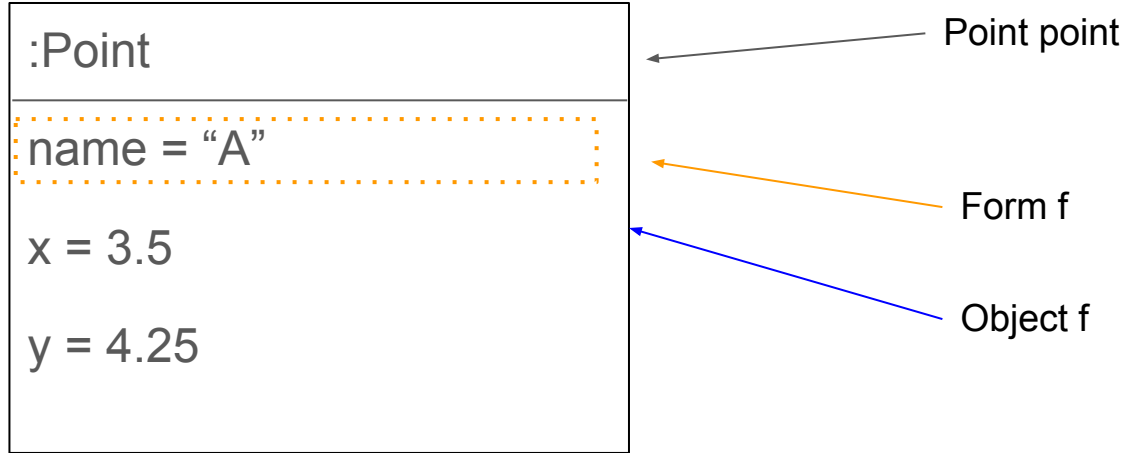
References can share the same object

```
✓  point = {Point@1815} "Point(x=3.5, y=4.25)"  
  (f) x = 3.5  
  (f) y = 4.25  
  > (f) name = "A"  
✓  f = {Point@1815} "Point(x=3.5, y=4.25)"  
  (f) x = 3.5  
  (f) y = 4.25  
  > (f) name = "A"
```

References can share the same object

```
✓  point = {Point@1944} "Point(x=4.5, y=3.25)"  
   x = 4.5  
   y = 3.25  
  >  name = "A"  
✓  f = {Point@1944} "Point(x=4.5, y=3.25)"  
   x = 4.5  
   y = 3.25  
  >  name = "A"  
✓  o = {Point@1944} "Point(x=4.5, y=3.25)"  
   x = 4.5  
   y = 3.25  
  >  name = "A"
```

References can share the same object

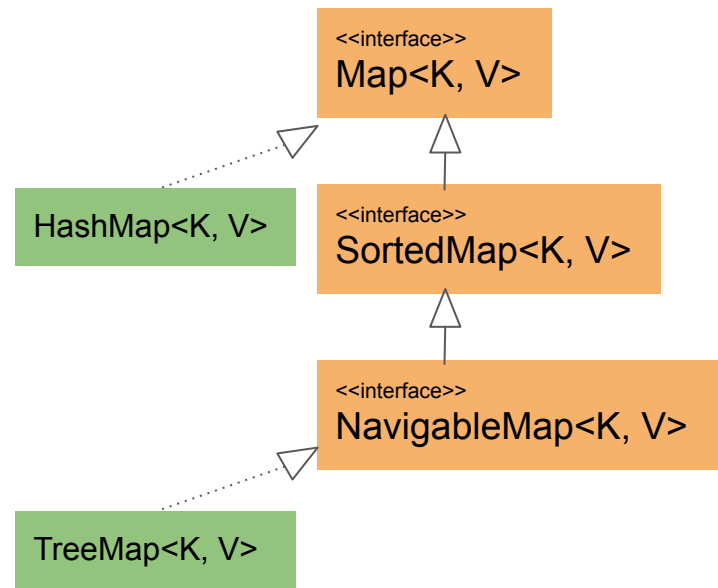
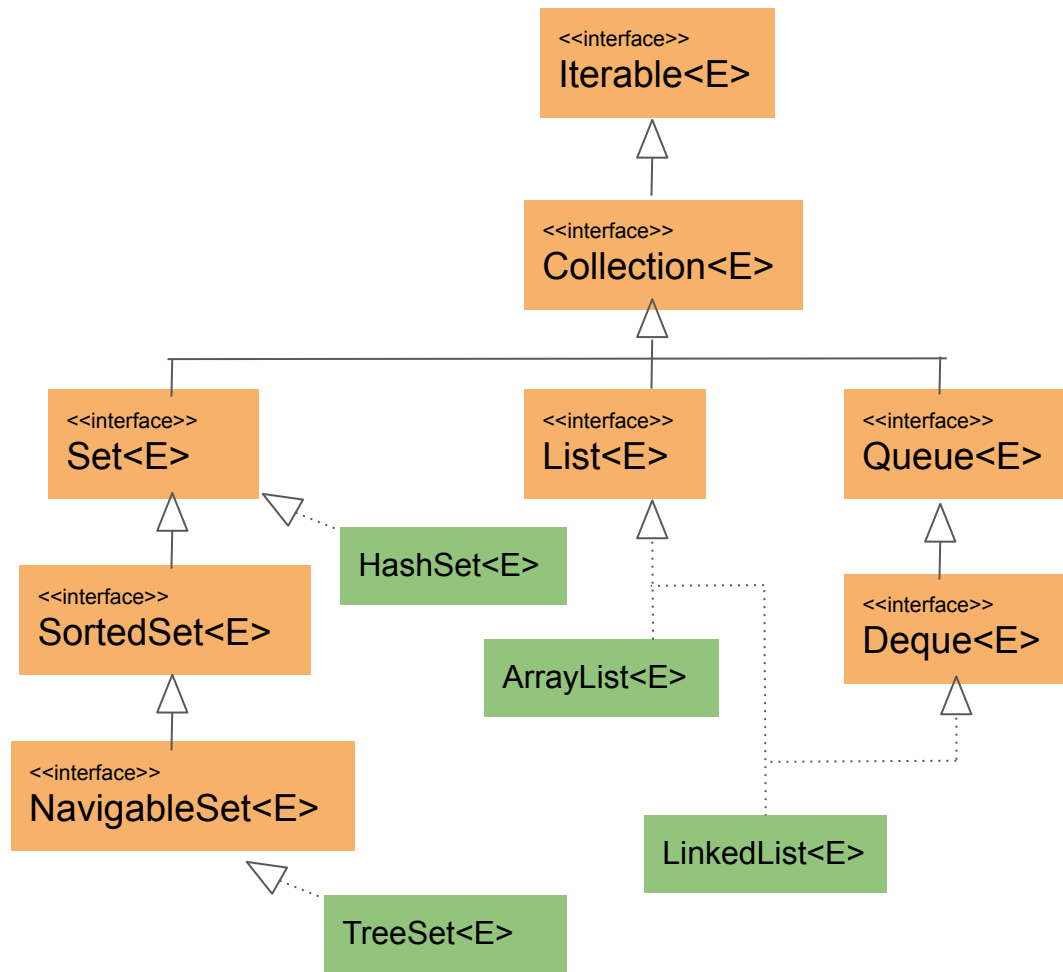


Temporal Data

- Java 1.0: Date all fields
- Java 1.1: Calendar + GregorianCalendar all fields
- Java 8: package java.time
 - LocalDate: Year, Month, Day
 - LocalDateTime: Year, Month, Day, Hour, minute, Second, ms
 - LocalTime: Hour, Minute, Second, ms
 - ZonedDateTime: DateTime + time zone
 - Duration, Period

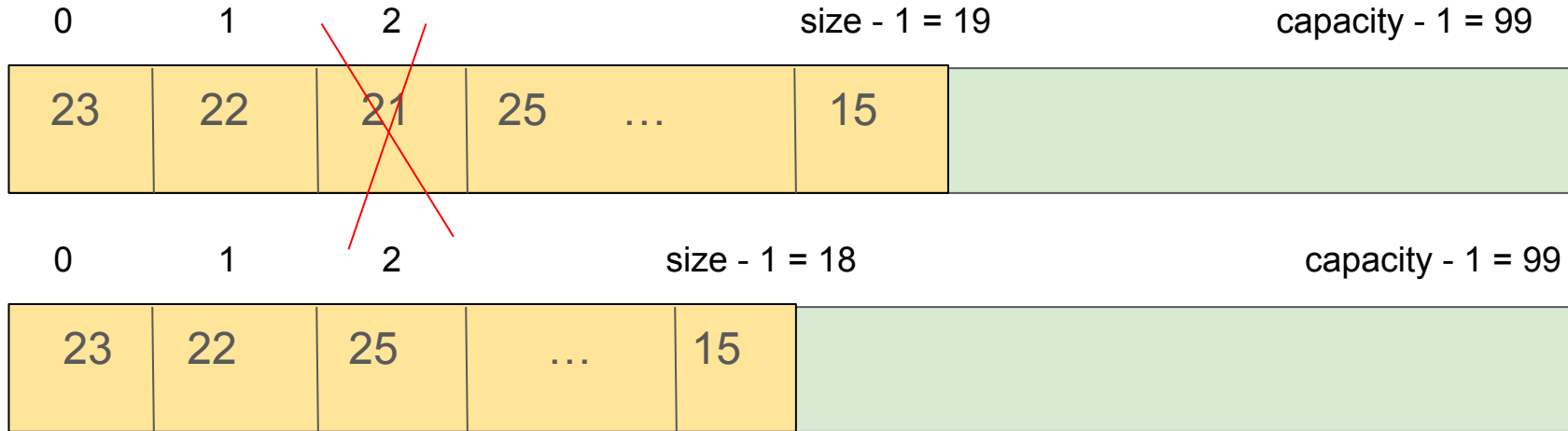
Array vs Collection

- Array : `String[]` villes
 - taille fixe
- Collection<E> : `Collection<String>`, `Collection<Double>`, `Collection<Plane>`
 - taille dynamique (en général)
 - `List<E>` : éléments rangés avec un index 0, 1, ..., size-1
 - `Set<E>` : pas de doublons
 - `SortedSet<E>`, `NavigableSet<E>` : éléments triés
- Map<K,V> : données indexées
- Type interface et plusieurs implémentations possibles
 - `List<E> => ArrayList<E>`, `LinkedList<E>`, `Vector<E>`, ...



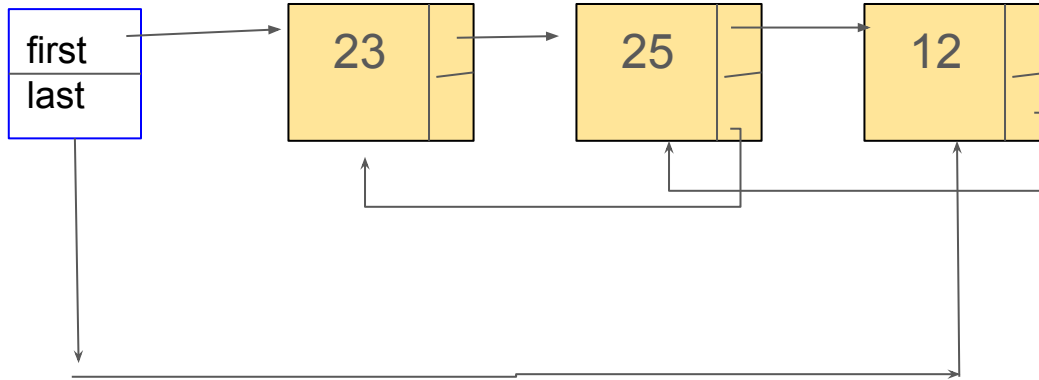
ArrayList vs LinkedList

ArrayList

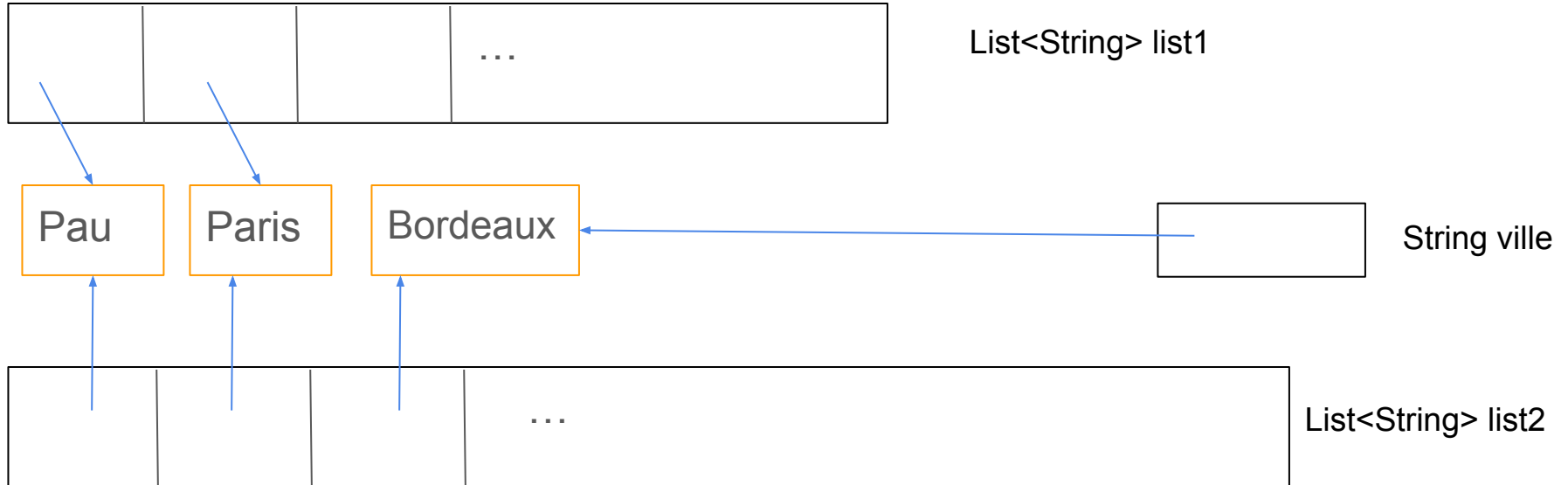


ArrayList vs LinkedList

Linked



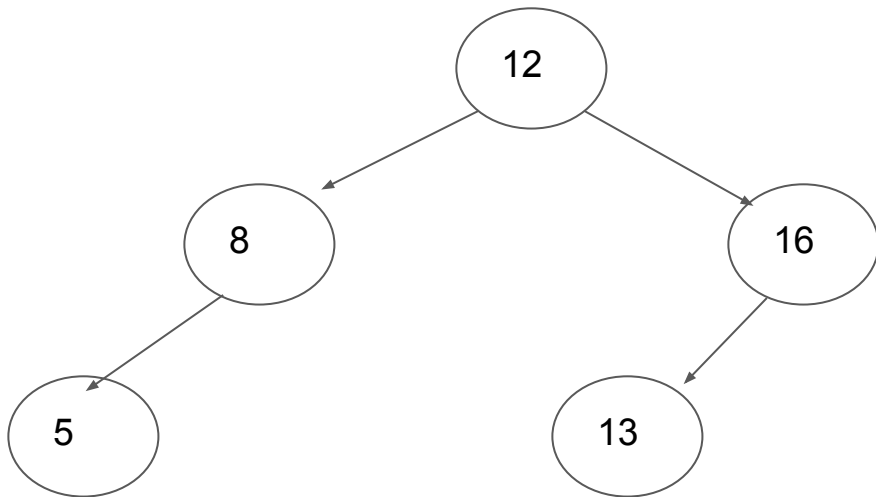
Partage de reference



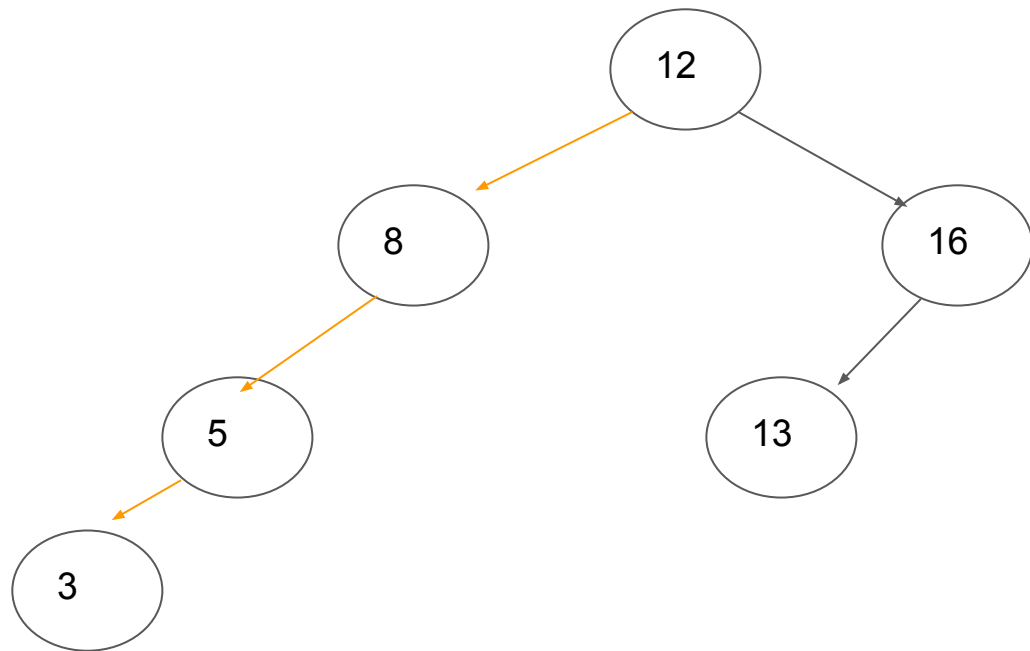
TreeSet

insert ?

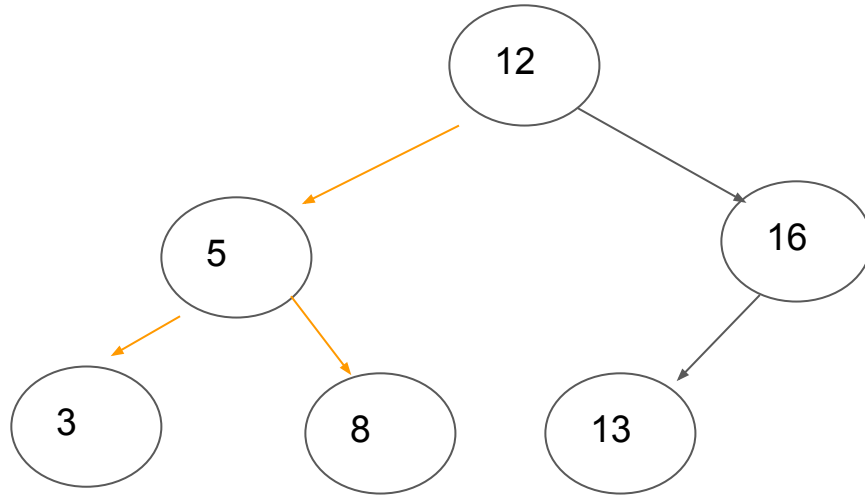
3



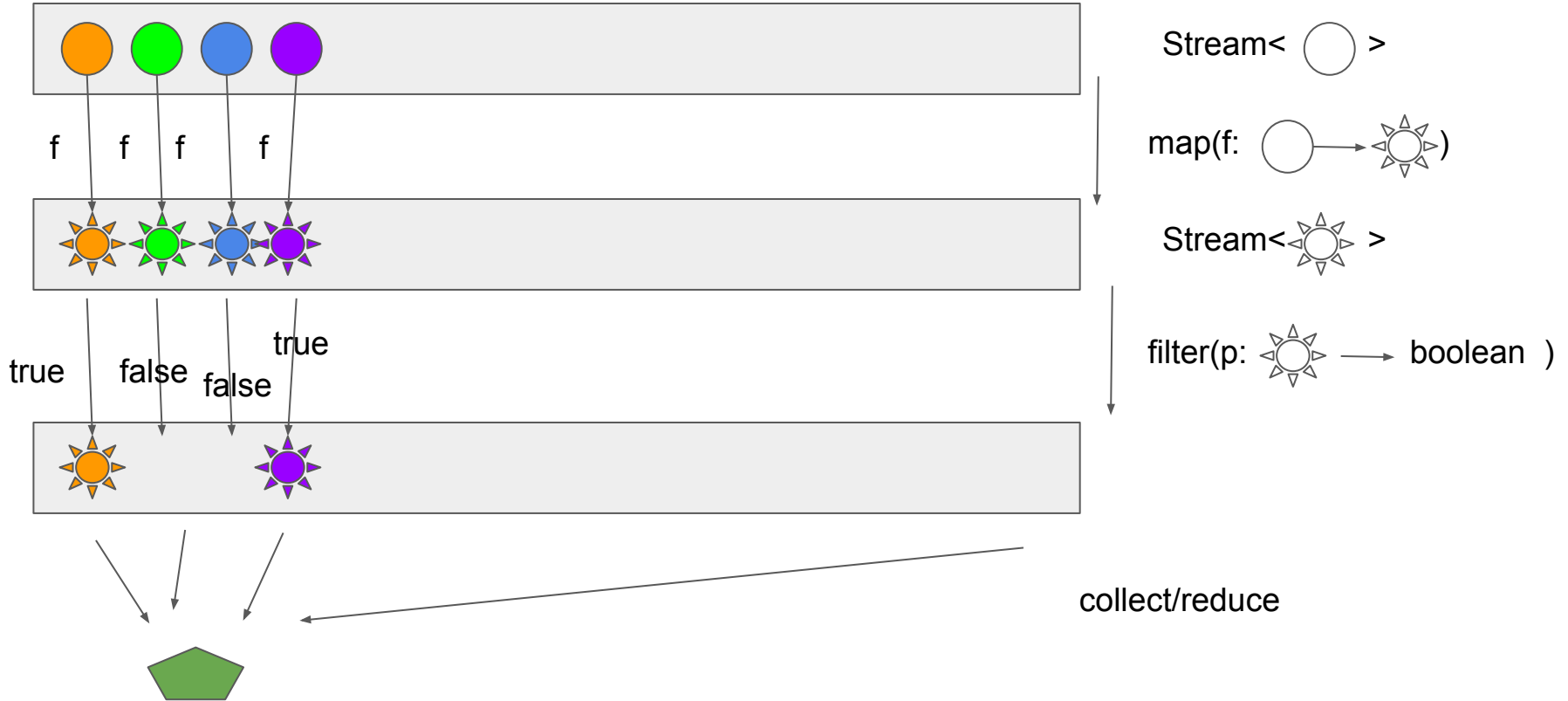
TreeSet (2)



TreeSet (3) : rééquilibrage



Stream, map/reduce



Stream: pipeline map/reduce

Etapas

- Source : Collection, JPA repository (SQL), Generator, Stream.of
- Intermédiaire(s)
 - map(f) : transformation de chaque donnée avec la fonction f
 - filter(p) : garde que les données respectant le prédicat p
 - peek() : jeter un coup
 - limit(n), skip(n) : coupe le stream après n valeurs ou les premières n valeurs
- Finale
 - Object
 - collect : toList, toCollection, stats, ...
 - forEach : print, save file, insert bdd (result void)
 - findFirst
 - entiers/flottants
 - sum, min, max, avg, statistics
 - reduce

Stream: objects vs primitive type

- `Stream<Double>`, `Stream<Integer>`, `Stream<Long>`
 - les données sont traitées en mode objet et allouées en mémoire (heap)
- `DoubleStream`, `IntStream`, `LongStream`
 - les données sont traitées sans allocation dynamique, uniquement dans le stack

Anonymous Functions (lambda function)

- `x -> x*x + 1` // valable pour short, int, long, double, float
- `(var x) -> x*x + 1` // valable pour short, int, long, double, float
- `(double x) -> x*x + 1` // valable double
- `(x, y) -> (x + y) * 3`
- `(var x, var y) -> (x + y) * 3`
- `(texte1, texte2) -> texte1 + ", " + texte2`
- `(String texte1, String texte2) -> texte1 + ", " + texte2`
- `() -> 1` // function with no arg
- `x -> {
 var y = x + 3;
 return y * 2;
}`

Function references

- `String::length` // ref (object) method of type `String` i.e `s -> s.length()`
- `String::split` // `(text, regex) -> text.split(regex)`
- `Math::sqrt` // ref static method of class `Math` i.e `n -> Math.sqrt(n)`
- `String city = "Toulouse"`
 - `city::length` // ref method of the object `city` i.e. `() -> city.length()`
 - `city::split` // i.e `regex -> city.split(regex)`
- `TreeSet::new` // ref constructor(s) of class `TreeSet`

// NB: need to be resolved (4 possibilities)

Functional type = interface with one method

Comparator<T> : T x T -> int

Function<T,R> : T -> R

BiFunction<T,U,R> : T x U -> R

Supplier<T> : () -> T

Consumer<T> : T -> void

BiConsumer<T,U> : T x U -> void

Predicate<T> : T -> boolean

BiPredicate<T,U> : T x U -> boolean

UnaryOperator<T> : T -> T

BinaryOperator<T> : T x T -> T

Fonctions et types fonctionnels

- 1 type fonctionnel :
 - 1 interface avec une seule méthode à implémenter
 - annotée avec `@FunctionalInterface` (pas obligatoire)
 - anciennes interfaces : `Comparator`, `ActionListener`
 - nouvelles interfaces : package `java.util.function`
 - `Function<T,R> : T -> R`
 - `UnaryOperator<T> : T -> T`
 - `Predicate<T> : T -> boolean`
 - `Consumer<T> : T -> void`
 - `Supplier<T> : () -> T`
 - `BiFunction<T,U,R> : T x U -> R`
 - `BinaryOperator<T> : T x T -> T`
 - `BiPredicate<T> : T x T -> boolean`
 - `BiConsumer<T, U> : T x U -> void`
 - + toutes les variantes avec types primitifs : `IntFunction`, `ToIntFunction`,

Comparable, Comparator, Sort

- par défaut, un objet n'est pas comparable
- le sont:
 - types primitifs (<) ou via leur type objet correspondant
 - exemple: int et Integer
 - String
 - données temporelles
- interface Comparable<E>
 - définit un ordre naturel pour le type E
 - méthode: int compareTo(E other)
 - Exemple:
 - `int cmp = a.compareTo(b)`
 - `cmp < 0 : a < b`
 - `cmp = 0 : a = b`
 - `cmp > 0 : a > b`

Comparable, Comparator, Sort (2)

- interface `Comparator<T>`
 - méthode `int compare(T t1, T t2)`
 - même sémantique que `compareTo` sur le résultat
 - exemple
 - `Comparator<String> comparator = ???`
 - `int cmp = comparator .compare("Nancy", "naNTes")`

JUnit

Framework unit testing for Java (functional testing too)

- JUnit 3: no annotation, method starting with test
- JUnit 4: annotation `@Test`, `@BeforeClass`, ...
- TestNG: code factorization
- JUnit 5: include junit 4 legacy + modern jupiter
 - stream, lambda, `assertAll`
 - `@ParameterizedTest`, `@Repeat`
 - `@BeforeAll`, `@BeforeEach`, `@AfterAll`, `@AfterEach`
 - assertions

JUnit assertions

- `assertAll`: collect all errors from multiple assertions
- `assertEquals(expectedValue, actualValue, [message])`
 - for Objects: method `equals`
 - for primitives: `==`
- `assertEquals(expectedDoubleValue, actualDoubleValue, delta, [message])`
- `assertSame(expectedObject, actualObject, [message])`: `==` (same object in memory)
- `assertTrue(actualBoolean, [message])`: check condition ok
- `assertFalse(actualBoolean, [message])`: check condition not ok
- `assertNull`, `assertNotNull`: check reference null
- `assertThrows`, `assertDoesNotThrow`: check exception is [not] thrown
- `assertArrayEquals`, `assertIterableEquals`: for arrays and collections

Inheritance and genericity

static <T>

boolean addAll(Collection<? super T> c, T... elements)

static <T extends Object & Comparable<? super T>>

T max(Collection<? extends T> coll)

static <T>

T max(Collection<? extends T> coll, Comparator<? super T> comp)

static <T>

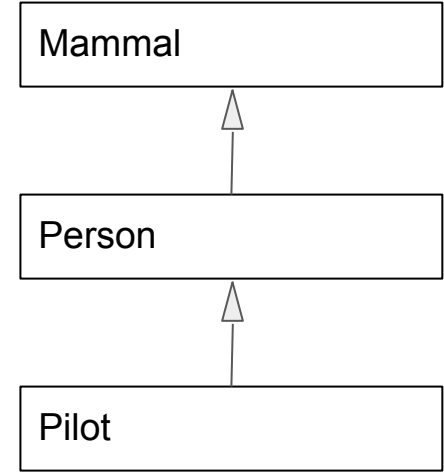
boolean addAll(Collection<? super T> c, T... elements)

static <T extends Object & Comparable<? super T>>

T max(Collection<? extends T> coll)

static <T>

T max(Collection<? extends T> coll, Comparator<? super T> comp)



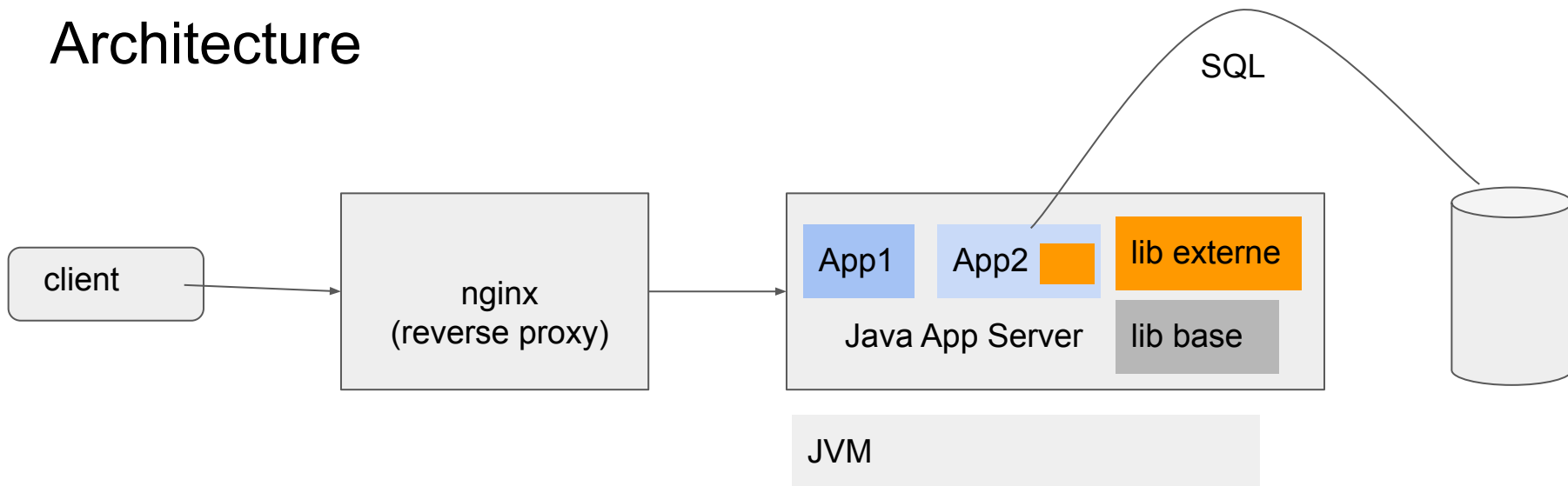
Application Java Backend

- Webapp MVC (Model View Controller)
- API Rest (Web Services)
 - micro-services
 - framework Spring
- Plateforme d'exécution (Java EE, JEE, Jakarta EE)
 - JVM : java / java.exe
 - Serveur Application Java
 - Tomcat (JEE Profile Web)
 - Full JEE:
 - JBoss / WildFly (RedHat)
 - Oracle WebLogic
 - IBM WebSphere

Exemple

- Développement d'une application avec
 - Java SE 11
 - servlet (spec JEE)
 - JPA (spec JEE)
 - dépendance spring
- Déploiement & exécution
 - JRE 11
 - serveur application Java
 - Profile Web JEE : tomcat + jar JPA + conf + jar spring
 - Full JEE :
 - RedHat Wildfly + jar spring

Architecture



Wildfly

- <https://www.wildfly.org/>
- modes
 - standalone : 1 process java
 - domain : plusieurs processus java
 - 1 domain controller (process controller)
 - 1 host controller par host/machine
 - server(s)
- interfaces
 - public : 8080, 8443
 - management : 9990
- HAL: appli web d'administration
- jboss-cli : client en ligne de commande d'administration

Déploiement d'une webapp

http://192.168.56.106:8081/bonjour/index.html

http://192.168.56.106:8081/bonjour/Goodbye

- 192.168.56.106 : hostname ou ip serveur
- 8081 : port pour atteindre le serveur
- /bonjour : contexte de l'application
- /index.html : ressource publique (html, css, image, jsp)
- /Goodbye : ressource privé routée (servlet, api rest, ...)

RDBMS

- Relational Database Management System
- Created in the 70's
- Main vendors
 - Oracle Database
 - Microsoft SQL Server
 - PostgreSQL
 - MySQL / MariaDB
 - Sqlite
- Standard SQL (1974 - 2019)

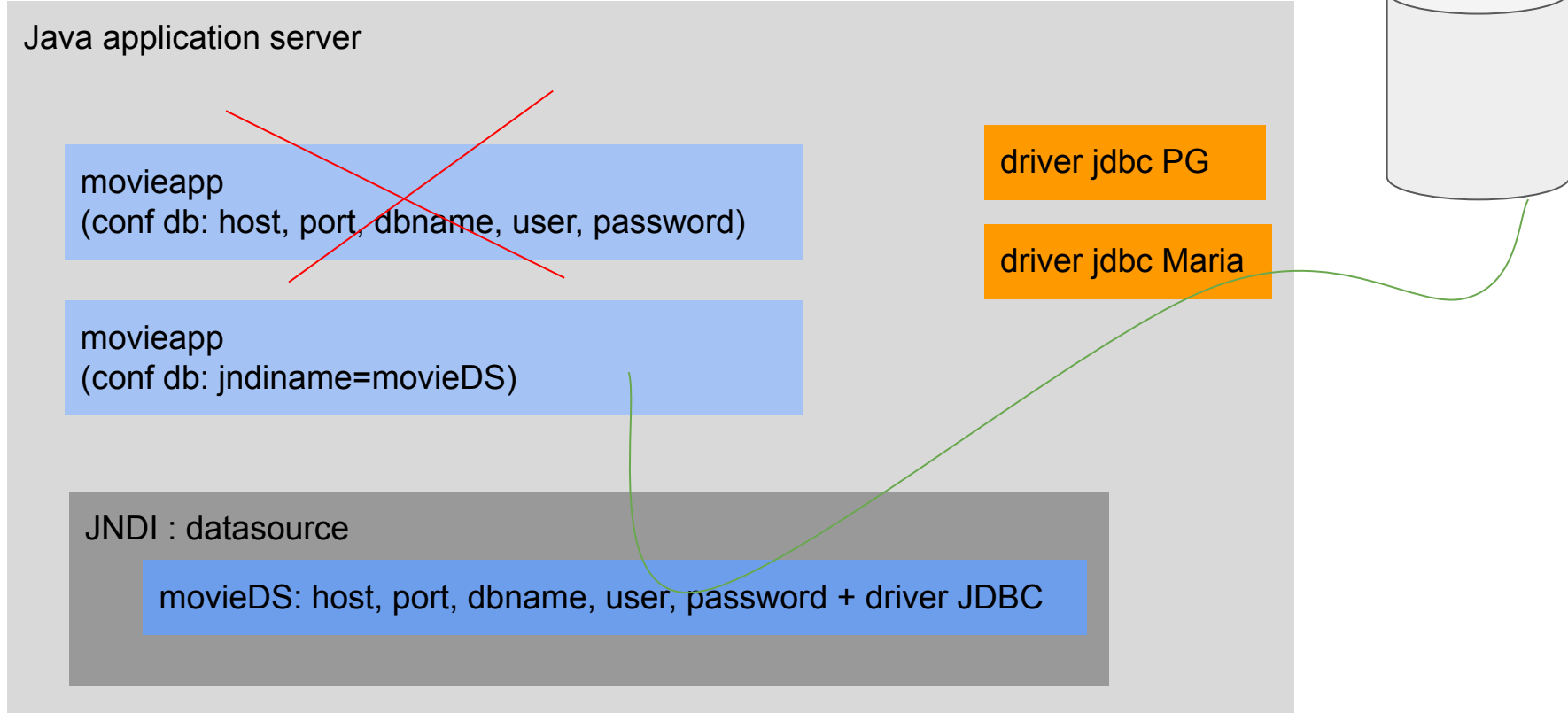
Java application with Relational Database

- communication appli Java <-> RDBMS
- langage commun de communication SQL
- JDBC : Java Database Connectivity (inclus Java SE)
 - Comment gérer des requêtes (insert, update, delete, select)
 - package java.sql et javax.sql
 - Driver : spécification d'un driver éditeur
 - Connection : établir une connexion avec la base de données
 - host, port, dbname, user (, password)
 - DataSource : pool de connexion(s)
 - Statement : exécuter une requête
 - select * from movies where year = 2020
 - PreparedStatement : exécuter une requête préparée
 - select * from movies where year = ?
 - paramètre #1 pourra être 2020, 2021, ...
 - ResultSet : résultat d'une requête
 - Driver JDBC apporté par l'éditeur ou la communauté
 - postgresql-42.2.20.jar
- JNDI : externaliser les settings JDBC de l'appli => serveur appli

Java with RDBMS (2)

- JDBC
- specification java JEE: JPA (Java Persistence API) :
 - main provider Hibernate
 - ORM: Object Relational Mapper
- Spring Data: JpaRepository

JNDI



Configure JNDI Datasource

- tomcat: ajouter en XML l'entrée JNDI
 - <http://tomcat.apache.org/tomcat-9.0-doc/jndi-datasource-examples-howto.html>
- wildfly

Files

- package java.io (old) and java.nio (v1 and v2: modern)
- classes File (old) or Path (new) : file, directory, link, ...
- classes toolbox: Files, Paths, FileSystems
- content
 - in: InputStream, Reader, BufferedReader, ...
 - out: OutputStream, Writer, PrintWriter, ...