<https://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=5001&get_params=p_exam_id:1Z0-808>

<https://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=5001&get_params=p_exam_id:1Z0-809>

Practice Qs:

http://education.oracle.com/education/downloads/Exam808\_SampleQuestion.pdf

Curs 8:

* java.util
* Collections framework vs Collection interface
  + **List:** Ordered, allows duplicates, elements accessed by index
  + **Set:** Does not allow duplicates
  + **Queue:** Specific order
  + **Map:** Key value pairs, no duplicates for keys
* **Methods**:
  + **Collection**: boolean **add**(E element), boolean **remove**(Object object), boolean **isEmpty**(), int **size**(), void **clear**(), boolean **contains**(Object object)
  + **List**: void **remove**(int index), void **add**(int index, E element), E **get**(int index), int **indexOf**(Object o), E **set**(int index, E e)
  + **ArrayDeque**: boolean **add**(E e), E **element**(),  boolean **offer**(E e), boolean **offer**(E e), void **push**(E e), E **poll**(), E **peek**(), E **pop**()
  + **Map**: void **clear**(), boolean **isEmpty**(), int **size**(), V **get**(Object key), V **put**(K key, V value), V **remove**(Object key), boolean **containsKey**(Object key), boolean **containsValue**(Object), Set<K> **keySet**(), Collection<V> **values**()
* **Implementations**: ArrayList, LinkedList, HashSet, TreeSet, ArrayDeque, HashMap, TreeMap
* **ArrayList** constructor - empty, capacity, copy another list
* **LIFO** (stack) -> push/poll/peek
* **FIFO** (single-ended queue) -> offer/poll/peek
* Cannot contain primitives
* Collections class - Collections.sort and Collections.binarySearch
* Comparator vs. Comparable (   public int **compare**(T t1, T t2) public int **compareTo**(T o))

Java 8 addition: **removeIf**, **replaceAll**, **forEach**

Curs 7:

* Container: Elephant, Books -> emptyContainer, packContainer
* Generics: type parameters for code - since java 1.5
* **Raw** type - warning
* **Diamond** operator - optional on the right - 1.7
* **Generic classes**: standard letter for types: E (element), K (map key), V (map value), T and U for data types
  + - Single or multiple types

- All types, except enum types, anonymous inner classes and exception classes, can be generic

* **Generic interfaces:** 3 options:specify generic type, create generic class or raw
* **Generic methods**:
  + - Eg: Integer, Double, Character[]
    - type specified before return type
* **Wildcard**: List<Object> l  != new ArrayList<String>(); -> ? - unbounded

Immutable -> cannot add

* **Bounded**:

- **upper-bounded**: ? extends T

- **lower-bounded**: ? super T

* **Generic limitations**: primitives, cannot instantiate type, cannot declare static field, cannot use instanceof, cannot overload if generic is only diff
* for the compiler - **type erasure** - List<String> & List<Integer> (println) -> all types changed into Object, casts added by compiler

Curs 6:

* **Wrapper classes**:

- Boolean, Character,  (Byte, Short, Integer, Long, Float, Double -> extend Number)

- immutable (final, fields private, no mutators, final methods)

- assignment, constructor, static method -> valueOf (with primitive or String, float has for float    and double primitive), no empty constr, Character and Boolean

- parseInt, parseDouble, etc and NumberFormatException; no parse for Character

- autoboxing vs unboxing

- can’t appear in switch cases

- instant methods to return primitive types - (primitive)Value() - inherited from Number

- caching for values in range -128 to 127 for numeric and 0 to 127 for Character - valueOf

- no caching for floating point

- equality with == and equals, Boolean valueOf and static constants, autoboxing returns cached

* **DateTime** API

- LocalDate, LocalTime, LocalDateTime, Period and DateTimeFormatter ( no time zones)

- import java.time.\*

- all classes are immutable, all have private constructors

- **LocalDate** - of() with Month enum or number, index of month ->  LocalDate.of(2015, 1, 1)

       - now() and parse() -> parse(“9999-99-99”)  -> DateTimeException

       - getDayOfMonth(), getDayOfWeek(), getDayOfYear(), getMonth(),

         getMonthValue(), getYear()

* isBefore(), isAfter(), minusXX(), plusXX() - days, months, weeks, years, withXX()
* All additions, subtractions and replacements consider leap years - 2016.02.29

- **LocalTime** - of() - with or without seconds and nanoseconds, 24 hour clock

       - now() and parse() -> “99:99:99”, MIN (00:00), MAX (23:59:59.99999), MIDNIGHT

         (00:00), NOON (12:00) -> LocalTIme.MIN.equals(LocalTime.MIDNIGHT) -> true

       - getHour(), getMinute(), getSecond(), getNano() -> singular, not plural;

       - isAfter(), isBefore(), minusXX(), plusXX() - hour**s**, minute**s**, seconds, nanos,

       - withHour(), withMinute(), withSecond(), withNano()

- **LocalDateTime** - 2050-06-18**T**14:20:30:908765

       - atTime() to convert LocalDate to LocalDateTime -> overloaded

       - atDate() to convert LocalTime to LocalDateTime

       - overloaded of()

- **Period** - private constructor, static methods

 - of(), ofDays(), ofWeeks(), ofMonths(), ofYears() => used with LocalDate and LocalDateTime; for LocalTime it throws UnsupportedTemporarTypeException

- a period of 35 days is not stored as 1 month and 5 days

- can also pass negative values

- has a parse() method that takes a pattern: PnYnMnD -> can also be lowercase; can take - either before the whole pattern or before the individual number

- between two diff LocalDates

- subtracting individual elements -> P1M - P10D = P1M-10D, not P20D

- used with plus() or minus()

- can’t chain, only the last one is valid

- **DateTimeFormatter** - java.time.format

- private constructor

           - static methods : ofLocalizedDate, ofLocalizedTime, ofLocalizedDateTime -> they take a FormatStyle value -> FULL, LONG, MEDIUM, SHORT

- static fields: BASIC\_ISO\_DATE, etc

- a pattern: ofPattern(“yyyy MM dd”)

- call format() on date/time objects or datetimeformatter

- call parse() method on date/time object (static) or datetimeformatter class (instance)

Curs 5:

* Try with resources - order + suppressed exceptions
* **Math** - abs(), ceil(), floor(), min(), max(), round(), random(), pow(), sqrt(), Math.PI
* **System** - in, out, err, currentTimeMillis, exit(), gc()
* **Scanner** - nextInt(), nextLine()
* **Random** - nextInt()
* **String** - length(), charAt(), indexOf(), substring(), toLowerCase(), toUpperCase(), equals() and equalsIgnoreCase(),  startsWith(), endsWith(), contains(), replace(), trim() - chaining
* Strings in memory: immutable, new vs “”, == vs equals(),
* **StringBuilder** - mutable, overloaded constructor - String, StringBuilder, int capacity, nothing; append(), insert(), delete(), replace(), reverse(), substring() !!!, no trim()
* **StringBuffer** - synchronized methods, thread safe

Curs 4:

* Examples of catch returning a value and finally modifying it - primitive vs object
* Scope of variables
* Multicatch - OCP (unrelated, variable name, reassign)
* Nested try-catch blocks
* Try-with-resources - Cloaseable and Autocloseable interfaces - close method - OCP
* Suppressed exceptions
* Creating custom exceptions - OCP (extend Exception or RuntimeException) - LoginException - 3 constructors
* Overriding methods that throw exceptions
* Errors
  + StackOverflowError - stack memory
  + OutOfMemoryError - heap memory
  + NoClassDefFoundError
  + ExceptionInInitializerError - not the cause

Curs 3:

* Benefits of exceptions

- separating logic code from error handling code

- stack trace

* Categories - checked vs unchecked
* Throwing an exception - throw vs throws, checked vs unchecked
* Try-catch-finally blocks
* Returning a value from catch and finally
* Modifying the value before returning it from a catch or finally block
* Common exceptions: NullPointer, Arithmethic, Index & ArrayIndexOOB, ClassCast, IllegalArgumentException, NumberFormatException (with \_ , +, - and base16)

Curs 2

Clase anonime + Lambda

Clase anonime:

* Utilitate
* SerialTV + Validator
* Declarare si instantiere
* Extind/Implementeaza
* ;
* Constructori
* final/effectively final
* Pasare in metode
* Sortare cu Comparator (Object, cast)

Lambda:

* Utilitate
* Interfete functionale - @FunctionalInterface
* Exemple - sortare String si SerialTV + Validator
* Sintaxa - componente obligatorii/optionale
* Executie imediata/intarziata
* Predicate / test

Exercise:

interface Calculator, method calculate