DAY3 – Lecture notes sum-up

1st hour:

We've created a new project called playground using lein: run lein new playground in a terminal window.
Using emacs we've create the playground.clj file in the src directory under the playground project.
Setup the namespace in playgorund.clj by adding this line in the beginig: (ns playground).

In emacs we compile a clj source file by hitting C-x C-f (i.e. hold Ctrl key and press succesively x and f).

We've looked into the special forms:
def, if, let, do, quote, fn, loop, recur and mentioned the existence of special forms try, throw, dot, new, set!.

Using these and the macro defn we've worked on implementing the Monte-Carlo algorithm for computing the PI number aproximation.

Homework for this section: implement a correct fibonnaci function (that receives a single argument n, which can be negative or positive) – hint: use try and throw for negative numbers and loop/recur to make the recursive implementation efficient.

Homoiconic

BREAK

2nd hour:

We've worked our way through datatypes/structures:
- numeric, characters, strings, thruthines

On numeric:
hex
octal
BigDecimal
Long
+ - / * > < <= >= = == max min inc dec rem quot zero? pos? neg? number?

Thruthines: the and, or, not macros, true and false values, if special-form the single boolean context in Clojure

Strings & characters – Java strings and charaters
The clojure.core and clojure.string namespaces – bunch of cuntions to work with string without calling Java API: str, string? subs

Symbols – identifiers used for fn params, local and global vars/bindings, class names – and pretty much everything. They are functions in conjunction with maps.

Keywords – symbolic indetifiers tha evaluate to themselves; the are repsresented with a leading **:**

e.g. :this-is-a-keyword.
Keywords like symbols are functions in conjunction with maps.
Collections: lists, vectors, sets and maps.

(list 1 2 3) OR '(1 2 3) to create lists
[1 2 3 4] OR (vector 1 2 3) to create vectors
#{:A :b 2} to create sets or (hash-set...) (sorted-set...) function calls
{:key1 1 :key2 2} to create maps or use (hash-map...) (sorted-map...) function calls

All collections implement the ISeq interface and support these functions:

count

conj

seq

Collections are: immutable, persisten, support propper equality, easy to use, interaction with Java as-is.

Sequence are – whenever possible – lazy.

Address the subject of lazyness.