**Currency**

Dependency

[clojurewerkz/money "1.10.0"]

**(**require '**[**clojurewerkz.money.amounts  :**as** ma**])**  
**(**require '**[**clojurewerkz.money.currencies :**as** mc**])**  
**(**require '**[**clojurewerkz.money.**format**  :**as** mf**])**  
   
**(let** **[**burgers **(**ma/multiply **(**ma/amount-of mc/USD **5.50)** **4)**   
 milkshakes **(**ma/multiply **(**ma/amount-of mc/USD **2.86)** **2)**  
 pre-tax **(**ma/plus burgers milkshakes**)**  
 tax **(**ma/multiply pre-tax **0.0765** :**up)]**  
 **(**println "Total before tax: " **(**mf/**format** pre-tax**))**  
 **(**println " Tax: " **(**mf/**format** tax**))**  
 **(**println " Total with tax: " **(**mf/**format** **(**ma/plus pre-tax tax**))))**

**Reify**

(**import** *java.io.File*)

(**let** [ff (**reify** *java.io.FilenameFilter*

(accept [this dir name]

(*.endsWith* name ".odt")))

dir (*File.* "D:/srilakshmi")] ;;list of files whose extension is .odt

(into [] (*.listFiles* dir ff))) ;; dir -> files -> listfiles -> []

proxy

Defrecord

(ns ns1)

(defrecord Person [firstName lastName])

(new Person "James" "Richard")

(def person (new Person "James" "Richard"))

(.firstName person)

Deftype

;; define a couple of shape types

(deftype Circle [radius])

(deftype rect [length width])

;; multimethod to calculate the area of a shape

(defmulti area class)

(defmethod area Circle [c]

(\* Math/PI (\* (.radius c) (.radius c))))

(defmethod area Square [s]

(\* (.length s) (.width s)))

;; create a couple shapes and get their area

(def myCircle (Circle. 10))

(def mySquare (Square. 5 11))

(area myCircle)

(area mySquare)

(**ns** sampproj.core

(:require [clojure.string :as str]

[clojure.java.jdbc :as j]))

(**def** db {:dbtype "postgresql"

:dbname "SriDb"

:host "172.16.50.14"

:user "postgres"

:password "password@123"})

(**def** state-sql (j/create-table-ddl :state [[:state\_id :serial "PRIMARY KEY"]

[:state "VARCHAR(32)"] [:abrv "VARCHAR(2)"]]))

*;;(j/execute! db [state-sql])*

*;;(j/insert! db :state {:state "Alabama" :abrv "AL"})*

*;;(j/insert-multi! db :state [{:state "Alaska" :abrv "AK"}*

*;; {:state "Arizona" :abrv "AZ"}*

*;; {:state "Arkansas" :abrv "AR"}])*

*;;(j/db-do-prepared db ["INSERT INTO state (state, abrv) VALUES (?, ?)"*

*;; ["California" "CA"]*

*;; ["Colorado" "CO"]*

*;; ["Connecticut" "CN"]] {:multi? true})*

*;;(j/update! db :state {:abrv "CT"} ["abrv = ?" "CN"])*

*;; (j/query db ["SELECT abrv FROM state WHERE state = ?" "Connecticut"])*

(j/delete! db :state ["state = ?" "California"])