Rapid prototyping in legacy system with dadysql, clojure.spec and reagent/re-frame

- Introduction
- Legacy system
- Why clojure?
- Dadysql

About Me

- Working for Commerz Finanz- BNP Paribas/Commerz Bank
- CORBA/C++/J2EE/Java
- Clojure
- Love to solve small problem
- Love to build prototype

Legacy system

- No API End point.
- Multiple copy of data model.
- Business logics are spread all the place.
- Almost no test case or does not maintain since long time.
- Cobol/C++/CORBA is not for agile
- Database is playing main role for verifying data and business logic

- Why Clojure?

Safe zoo

- There are groups of animal in zoo, all groups are infected by some kind of virus. Take a sample from every group and build new group for symptom.

- Sample input: x, y1 y2, z1 z2 z3
- Sample output: x y1 z1, x y2 z1, x y1 z2, x y2 z2, x y1 z3, x y2 z3

Using Java

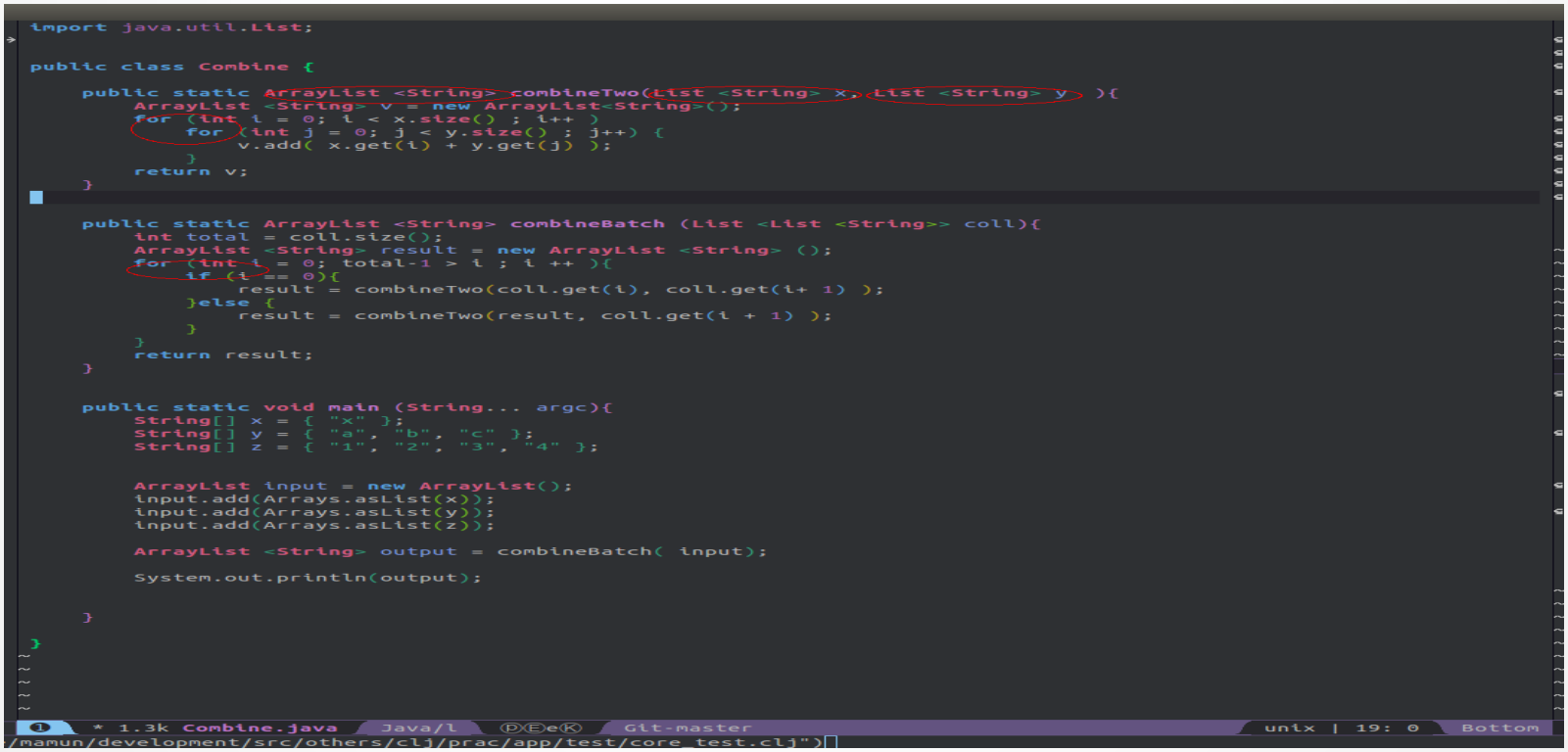
```java
import java.util.List;


public class Combine {

    public static ArrayList <String> combineTwo(List <String> x, List <String> y  ){
        ArrayList <String> v = new ArrayList<String>();
        for (int i = 0; i < x.size() ; i++ )
            for (int j = 0; j < y.size() ; j++) {
                v.add( x.get(i) + y.get(j) );
            }
        return v;
    }


    public static ArrayList <String> combineBatch (List <List <String>> coll){
        int total = coll.size();
        ArrayList <String> result = new ArrayList <String> ();
        for (int i = 0; total-1 > i ; i ++ ){
            if (i == 0){
                result = combineTwo(coll.get(i), coll.get(i+ 1) );
            }else {
                result = combineTwo(result, coll.get(i + 1) );
            }
        }
        return result;
    }


    public static void main (String... argc){
        String[] x = { "x" };
        String[] y = { "a", "b", "c" };
        String[] z = { "1", "2", "3", "4" };


        ArrayList input = new ArrayList();
        input.add(Arrays.asList(x));
        input.add(Arrays.asList(y));
        input.add(Arrays.asList(z));

        ArrayList <String> output = combineBatch( input);

        System.out.println(output);


    }

}
```

```
1  * 1.3k Combine.java  Java/l  ⓅⒺeⓀ  Git-master  unix | 19: 0  Bottom
/mamun/development/src/others/clj/prac/app/test/core_test.clj")
```

Using Clojure

```
(ns comb-impl)


(defn combine-two [f-coll s-coll]
  (for [f1 f-coll
        s1 s-coll]
    (str f1 s1)))


(defn combine-batch [r]
  (reduce combine-two r))


(comment

  ;"a" "abc bcn" "dca 123"

  (combine-batch [["s"] ["1" "2"] ["a" "b" "c"]  ]   )

  )
```

Using Java


import java.util.List;
public class Combine {
public static ArrayList <String> combineTwo(List <String> x, List <String> y){
ArrayList <String> v = new ArrayList<String>();
for (int i = 0; i < x.size() ; i++)
for (int j = 0; j < y.size() ; j++) {
v.add(x.get(i) + y.get(j));
}
return v;
}
public static ArrayList <String> combineBatch (List <List <String>> coll){
int total = coll.size();
ArrayList <String> result = new ArrayList <String> ();
for (int i = 0; total-1 > i ; i ++){
if (i == 0){
result = combineTwo(coll.get(i), coll.get(i+ 1));
}else {
result = combineTwo(result, coll.get(i + 1));
}
}
return result;
}
public static void main (String... argc){
String[] x = { "x" };
String[] y = { "a", "b", "c" };
String[] z = { "1", "2", "3", "4" };
ArrayList input = new ArrayList();
input.add(Arrays.asList(x));
input.add(Arrays.asList(y));
input.add(Arrays.asList(z));
ArrayList <String> output = combineBatch(input);
System.out.println(output);
}
}
1 * 1.3k Combine.java Java/l Git-master unix | 19: 0 Bottom
/mamun/development/src/others/clj/prac/app/test/core_test.clj")

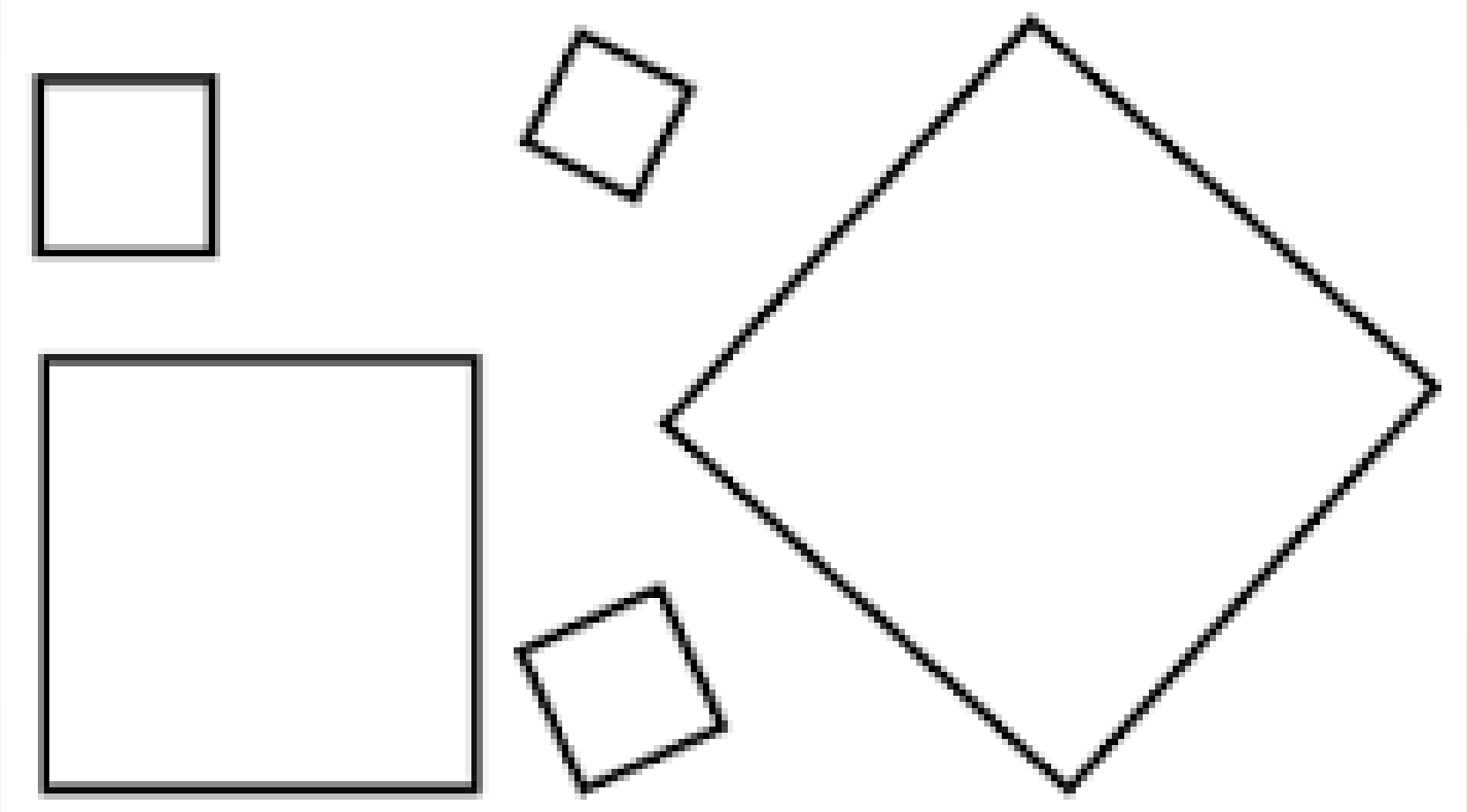
Live code demo

Tool to do fast

More in clojure!!

- Working with abstract (Seq & Data structure)
- Lot of Seq function like map, reduce, select etc
- Abstract structural binding - destructing
- Immutable and persistence data structure
- REPL (Feel like lufthansa nonstop you)
- ClojureScirpt, figwheel, devcard and more

What about data shape?



Data shape

Format

- Web as string {:credit-amount “3,000.00”}
- API end point {:credit-id 1213 :customer {:c-name “Max”}}
- Messaging with type {:credit {:credit-id 1234}}
- Batch precessing [{:credit {:credit-id 1234}}]
- Messaging

Constraint

- Credit-amount > 0 with decimal format
- Interest rate > 0.01
- C-name not empty
- Email with correct format

Clojure Spec

- Unify specification
- Parsing and conformation for string value and others
- Test data generation
- Error reporting

DadySQL

- Make sql data available as fast as possible as service or lib
- Write sql with metadata in sql file, inspired from yesql
- Param validate using spec
- Default param for primary key
- Versioning using namespace (com.app, v2.app or app.v2)
- Only two API pull and push(for transaction)
- Three level of metadata grouping (SQL, group of sql, database level)
- Don’t need to compile or load just run sql and continue

Thanks

- Dadysql github url- https://github.com/Mamun/dadysql
- Demo url https://github.com/Mamun/dadysql-demo