Snappyr DB

Alternative Snappy DB

Maarten Duijn 1517279
@mjduijn

Snappy DB

Key-value database for Android

Lightweight - Java 7 - Simple - FAST

Key Technologies

LevelDB

Key-value storage library

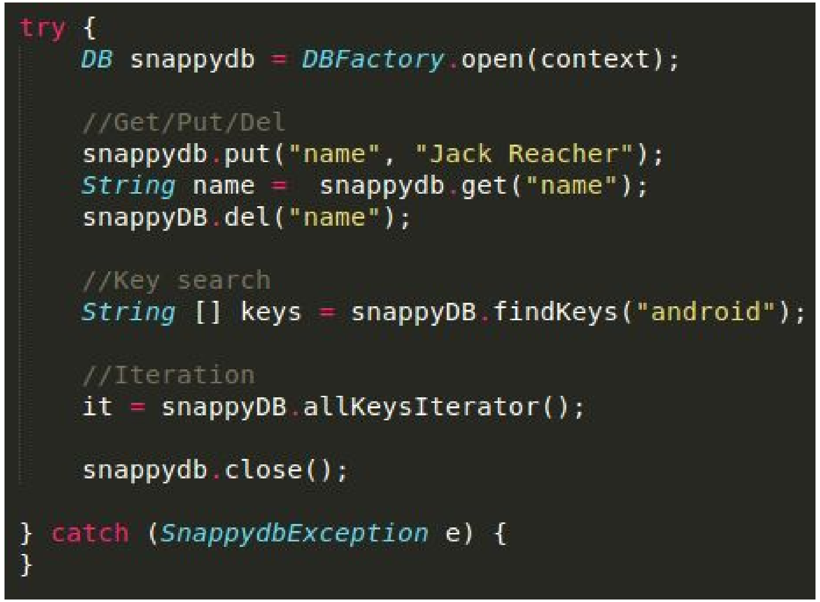
(String -> value)

Snappy

High speed compression library

Kryo

Fast and efficient object serialization framework.



```
try {
    DB snappydb = DBFactory.open(context);

    //Get/Put/Del
    snappydb.put("name", "Jack Reacher");
    String name =  snappydb.get("name");
    snappyDB.del("name");

    //Key search
    String [] keys = snappyDB.findKeys("android");

    //Iteration
    it = snappyDB.allKeysIterator();

    snappydb.close();

} catch (SnappydbException e) {
}
```

Simple to use API

Limiting...

Synchronous...

Try/Catch...

Verbose...

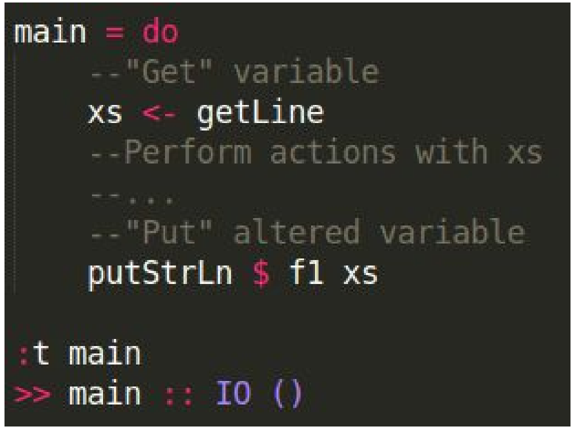
...is sooooo 2000's

Requirements

Performance like Snappy DB	-> Tech
Java 7 (Android compatibility)	-> Tech
Easy to use like Snappy DB	$\left.\begin{array}{l}\\ \\ \\ \\ \end{array}\right\}$?
More expressive	
More extensible	
Asynchronous	

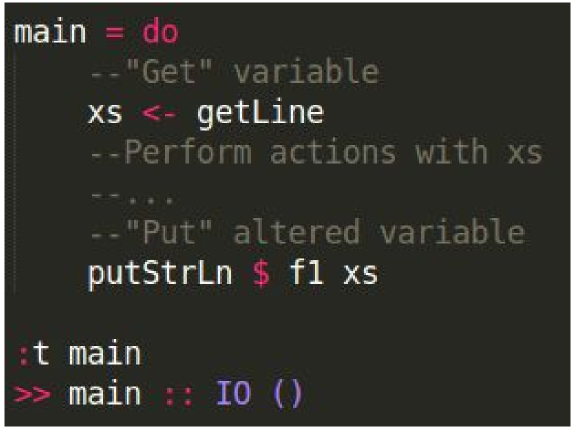
DB Monad

Getting inspired by Haskell's IO Monad

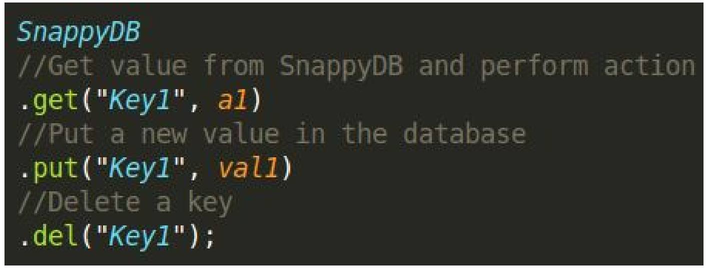


```
main = do
    --"Get" variable
    xs <- getLine
    --Perform actions with xs
    --...
    --"Put" altered variable
    putStrLn $ f1 xs

:t main
>> main :: IO ()
```

Haskell IO



```
main = do
    --"Get" variable
    xs <- getLine
    --Perform actions with xs
    --...
    --"Put" altered variable
    putStrLn $ f1 xs

:t main
>> main :: IO ()
```



```
SnappyDB
//Get value from SnappyDB and perform action
.get("Key1", a1)
//Put a new value in the database
.put("Key1", val1)
//Delete a key
.del("Key1");
```

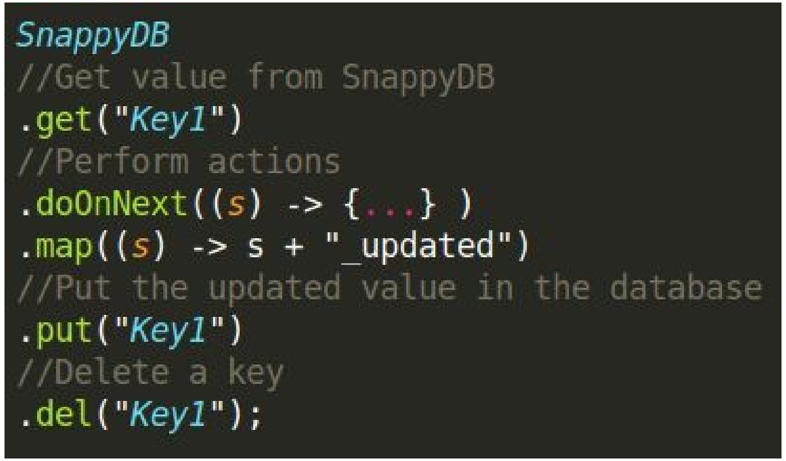
Haskell IO

Java DB Monad (attempt 1)

Expressive

Functions

Not Functional



```
SnappyDB
//Get value from SnappyDB
.get("Key1")
//Perform actions
.doOnNext((s) -> {...} )
.map((s) -> s + "_updated")
//Put the updated value in the database
.put("Key1")
//Delete a key
.del("Key1");
```

Attempt 2:

A more **Generic Monad**

Monad < DB >

Monad < String >

Monad < (key, value) >

Reactive Extensions

Don’t reinvent the wheel, just align it

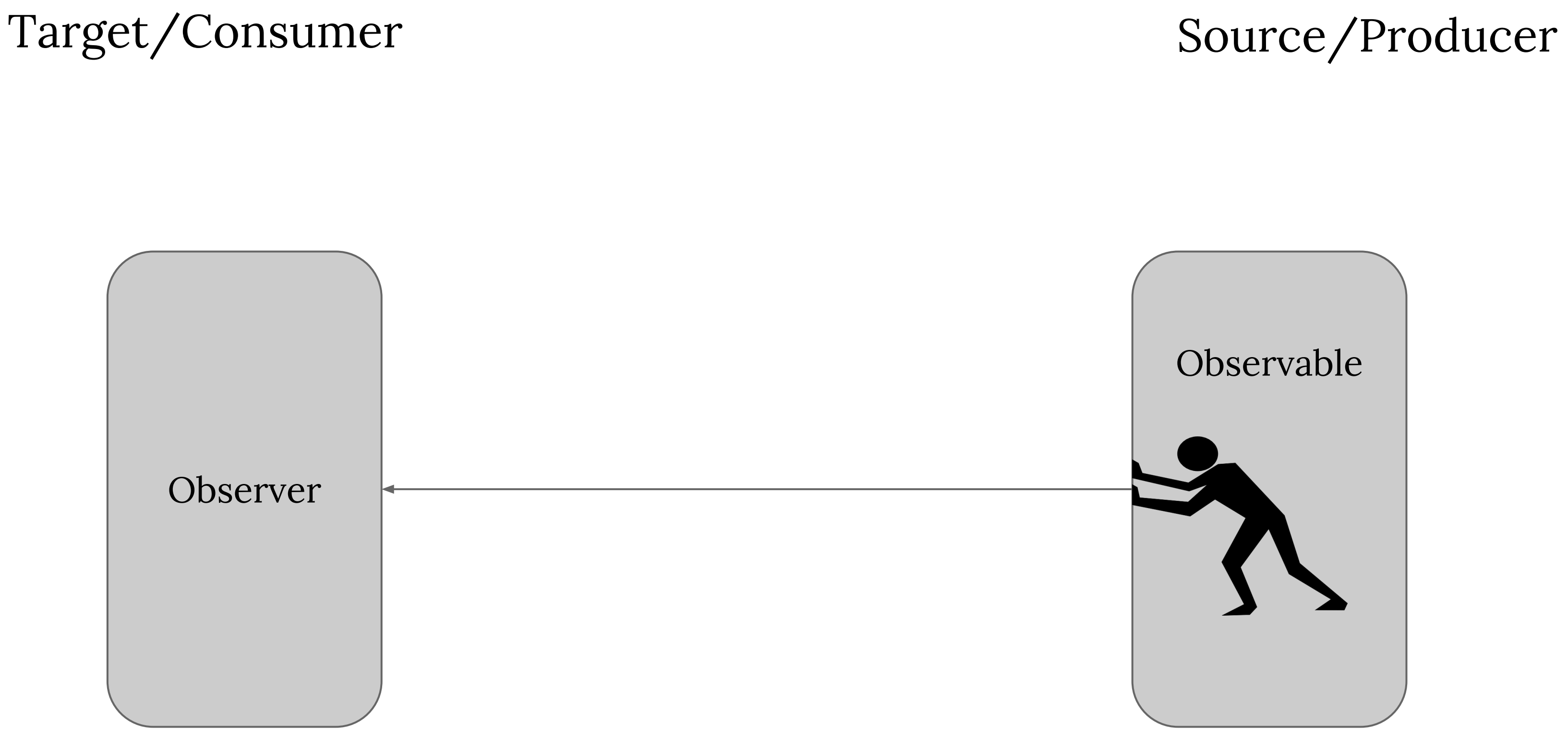
(in 2 slides)

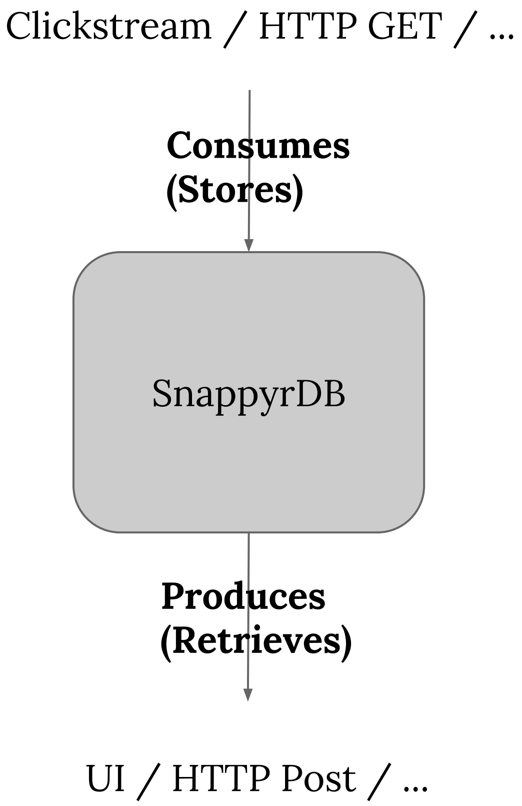
Asynchronous by design

Combine Chain

Merge **Streams** Filter

Map .... Lift(!)

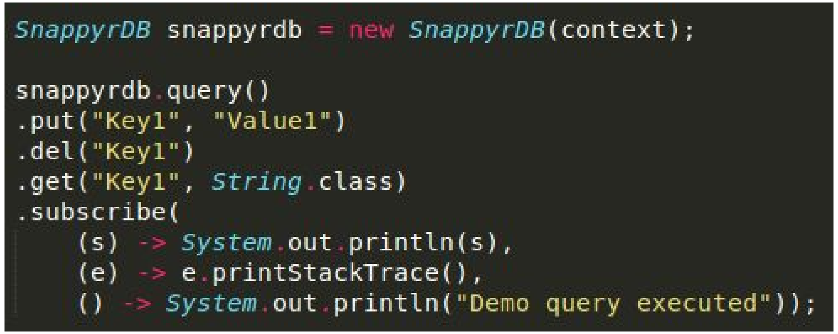

Target/Consumer
Source/Producer
Observer
Observable



Consumer and producer of streams

Basics

MVP key-value store



```
SnappyrDB snappyrdb = new SnappyrDB(context);

snappyrdb.query()
.put("Key1", "Value1")
.del("Key1")
.get("Key1", String.class)
.subscribe(
    (s) -> System.out.println(s),
    (e) -> e.printStackTrace(),
    () -> System.out.println("Demo query executed"));
```

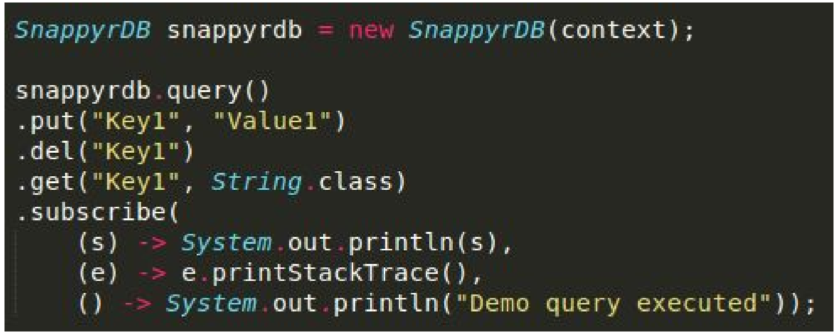
Query builder API

Create SnappyrDB

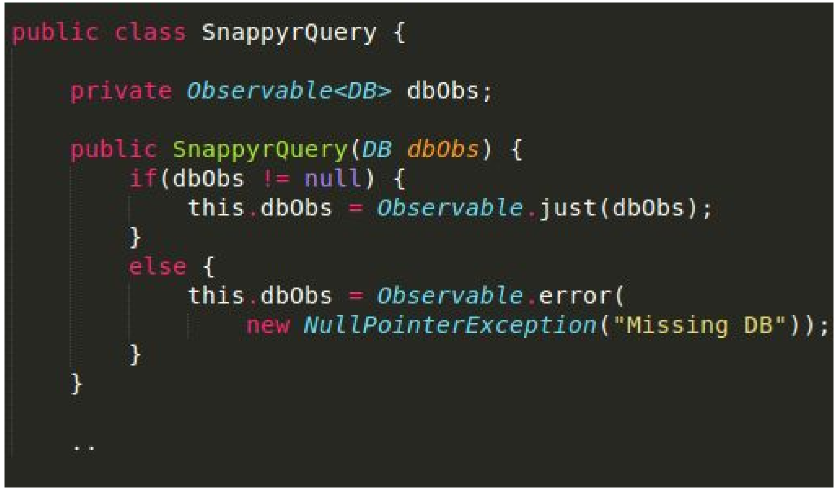
Compose query

Basic operations

Execute



```java
SnappyrDB snappyrdb = new SnappyrDB(context);

snappyrdb.query()
.put("Key1", "Value1")
.del("Key1")
.get("Key1", String.class)
.subscribe(
    (s) -> System.out.println(s),
    (e) -> e.printStackTrace(),
    () -> System.out.println("Demo query executed"));
```

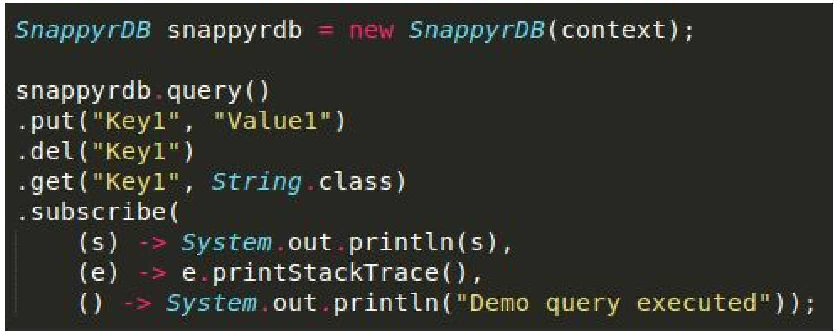


```java
public class SnappyrQuery {

    private Observable<DB> dbObs;

    public SnappyrQuery(DB dbObs) {
        if(dbObs != null) {
            this.dbObs = Observable.just(dbObs);
        }
        else {
            this.dbObs = Observable.error(
                new NullPointerException("Missing DB"));
        }
    }

    ..
```

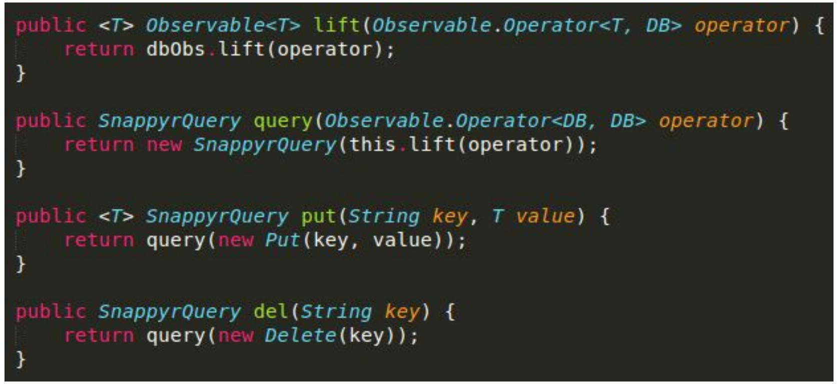
SnappyrQuery

Wrapper for Observable<DB>

Implements Put/Get/...



```
SnappyrDB snappyrdb = new SnappyrDB(context);

snappyrdb.query()
.put("Key1", "Value1")
.del("Key1")
.get("Key1", String.class)
.subscribe(
    (s) -> System.out.println(s),
    (e) -> e.printStackTrace(),
    () -> System.out.println("Demo query executed"));
```



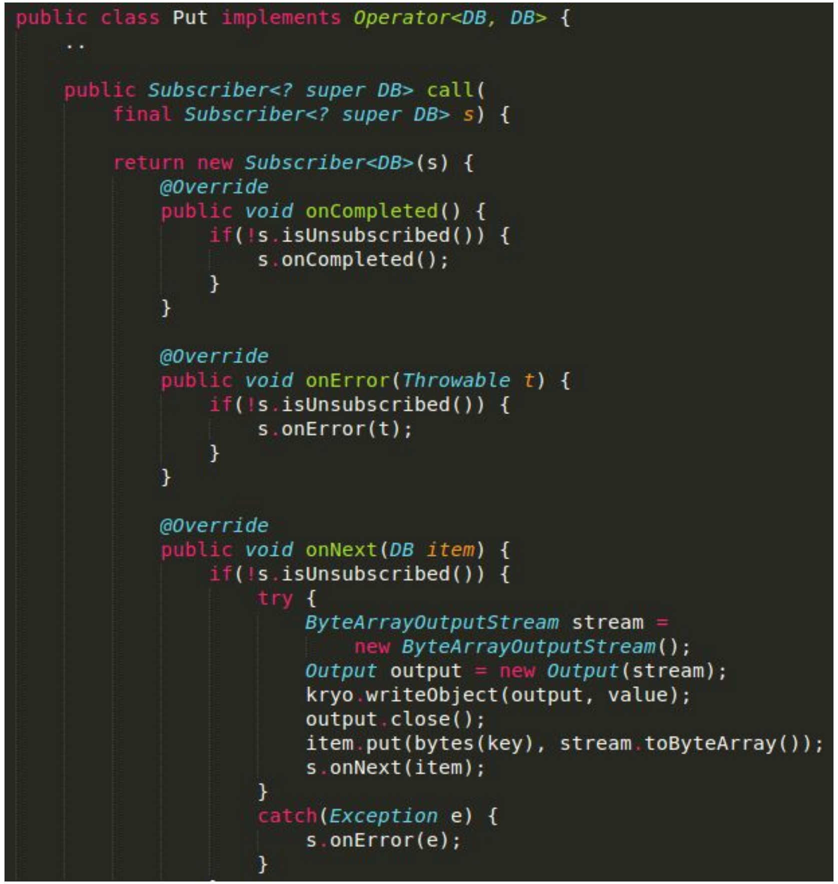
```
public <T> Observable<T> lift(Observable.Operator<T, DB> operator) {
    return dbObs.lift(operator);
}

public SnappyrQuery query(Observable.Operator<DB, DB> operator) {
    return new SnappyrQuery(this.lift(operator));
}

public <T> SnappyrQuery put(String key, T value) {
    return query(new Put(key, value));
}

public SnappyrQuery del(String key) {
    return query(new Delete(key));
}
```

Put / Del

Put / Del : Observable<DB> -> Observable<DB>

Lift!

Extensible



```java
public class Put implements Operator<DB, DB> {
    ..

    public Subscriber<? super DB> call(
        final Subscriber<? super DB> s) {

        return new Subscriber<DB>(s) {
            @Override
            public void onCompleted() {
                if(!s.isUnsubscribed()) {
                    s.onCompleted();
                }
            }

            @Override
            public void onError(Throwable t) {
                if(!s.isUnsubscribed()) {
                    s.onError(t);
                }
            }

            @Override
            public void onNext(DB item) {
                if(!s.isUnsubscribed()) {
                    try {
                        ByteArrayOutputStream stream =
                            new ByteArrayOutputStream();
                        Output output = new Output(stream);
                        kryo.writeObject(output, value);
                        output.close();
                        item.put(bytes(key), stream.toByteArray());
                        s.onNext(item);
                    }
                    catch(Exception e) {
                        s.onError(e);
                    }
```

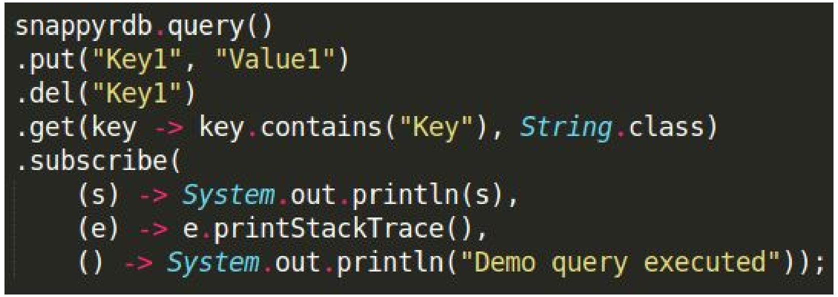
Put

Rx Operator

Put :: (String, String) -> Observable<DB>

Del :: (String) -> Observable<DB>

Kryo

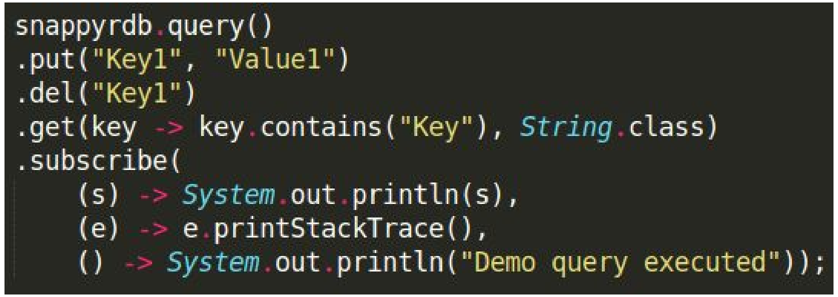


```
snappyrdb.query()
.put("Key1", "Value1")
.del("Key1")
.get(key -> key.contains("Key"), String.class)
.subscribe(
    (s) -> System.out.println(s),
    (e) -> e.printStackTrace(),
    () -> System.out.println("Demo query executed"));
```

Get

Get :: Observable<DB> -> Observable<T>

null/Object -> Observable

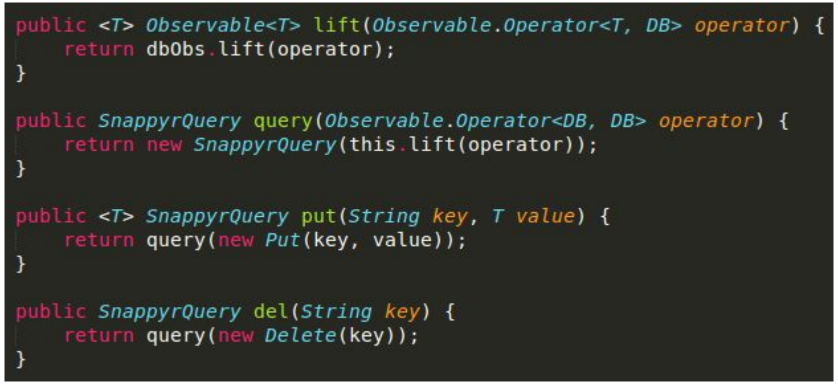


```
snappyrdb.query()
.put("Key1", "Value1")
.del("Key1")
.get(key -> key.contains("Key"), String.class)
.subscribe(
    (s) -> System.out.println(s),
    (e) -> e.printStackTrace(),
    () -> System.out.println("Demo query executed"));
```

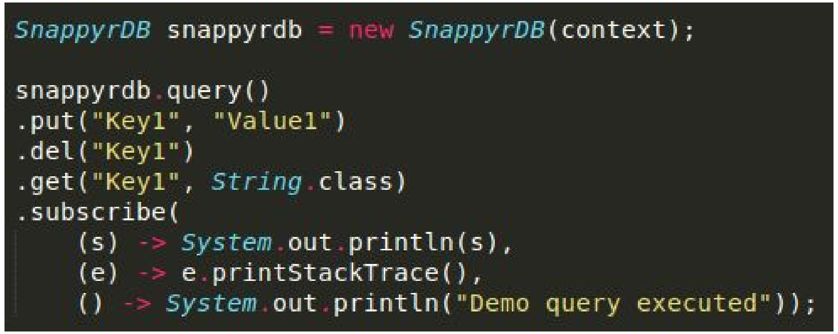
Get

Get :: Observable<DB> -> Observable<T>

null/Object -> Observable



```
public <T> Observable<T> lift(Observable.Operator<T, DB> operator) {
    return dbObs.lift(operator);
}

public SnappyrQuery query(Observable.Operator<DB, DB> operator) {
    return new SnappyrQuery(this.lift(operator));
}

public <T> SnappyrQuery put(String key, T value) {
    return query(new Put(key, value));
}

public SnappyrQuery del(String key) {
    return query(new Delete(key));
}
```



```
SnappyrDB snappyrdb = new SnappyrDB(context);

snappyrdb.query()
.put("Key1", "Value1")
.del("Key1")
.get("Key1", String.class)
.subscribe(
    (s) -> System.out.println(s),
    (e) -> e.printStackTrace(),
    () -> System.out.println("Demo query executed"));
```

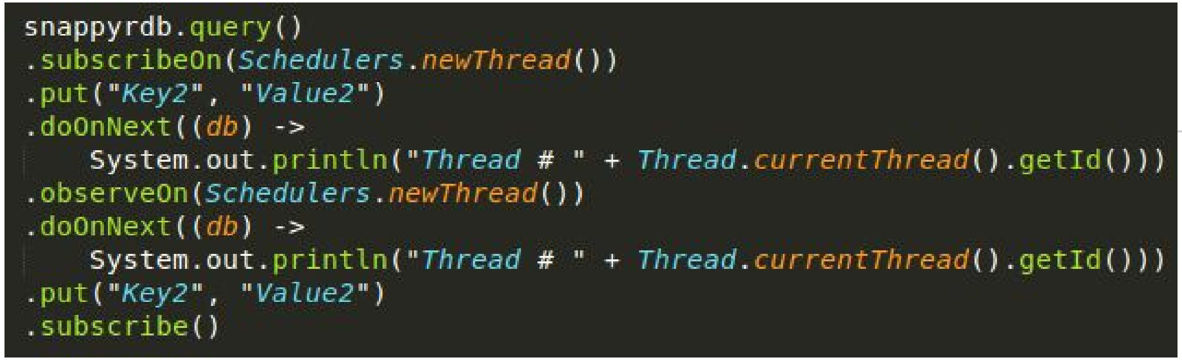
Subscribe

Rx Subscribe

Query execution

Fun stuff!

Basics down, what else?!

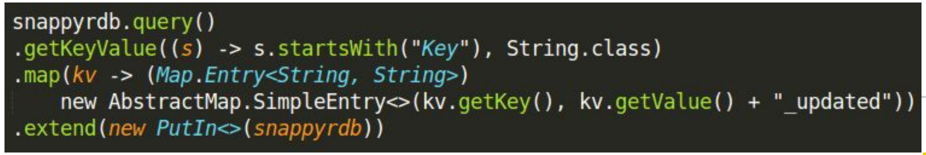


```
snappyrdb.query()
.subscribeOn(Schedulers.newThread())
.put("Key2", "Value2")
.doOnNext((db) ->
    System.out.println("Thread # " + Thread.currentThread().getId()))
.observeOn(Schedulers.newThread())
.doOnNext((db) ->
    System.out.println("Thread # " + Thread.currentThread().getId()))
.put("Key2", "Value2")
.subscribe()
```

Scheduling

ObserveOn

SubscribeOn



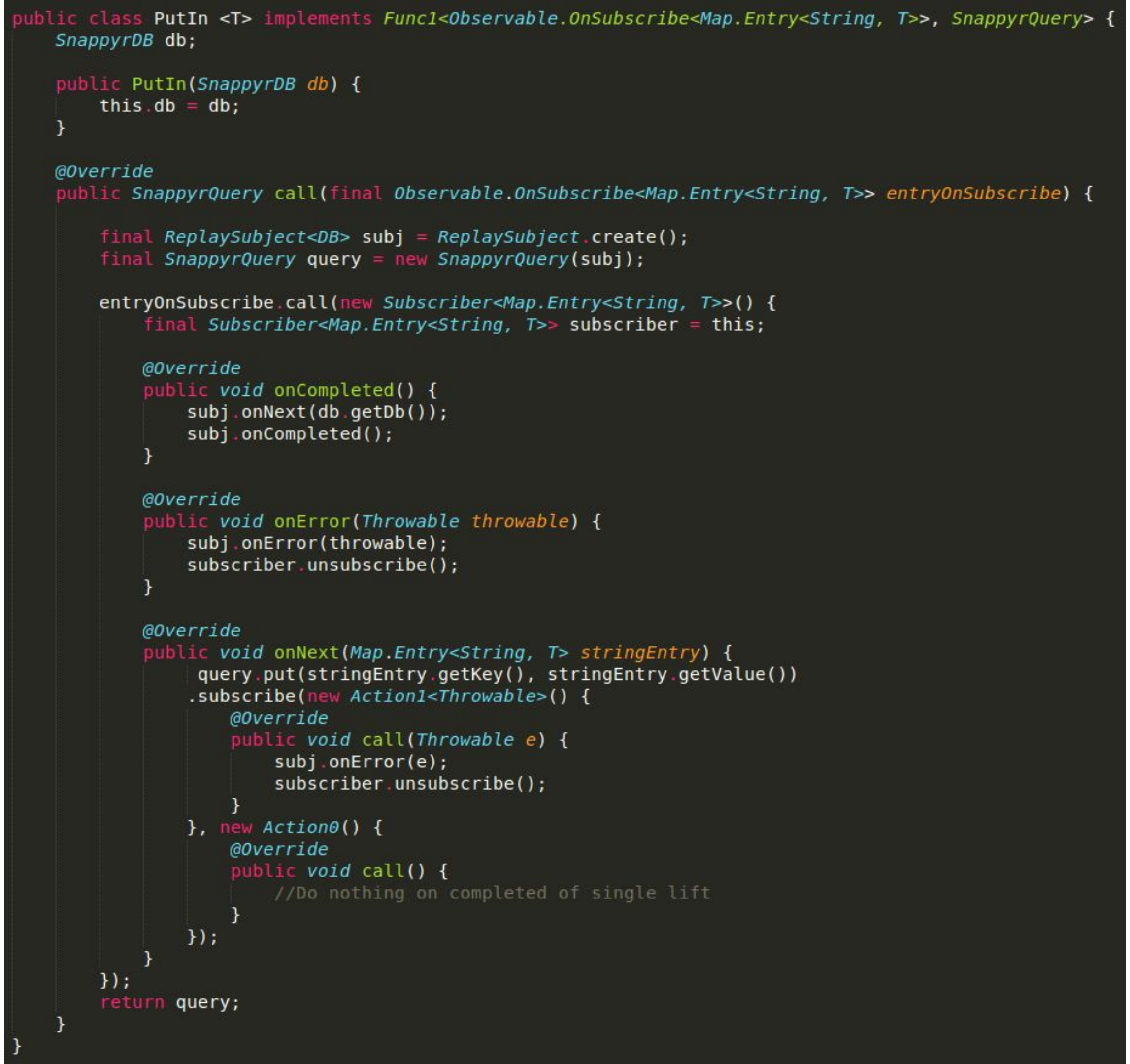
```
snappyrdb.query()
.getKeyValue((s) -> s.startsWith("Key"), String.class)
.map(kv -> (Map.Entry<String, String>)
    new AbstractMap.SimpleEntry<>(kv.getKey(), kv.getValue() + "_updated"))
.extend(new PutIn<>(snappyrdb))
```

Producing key values

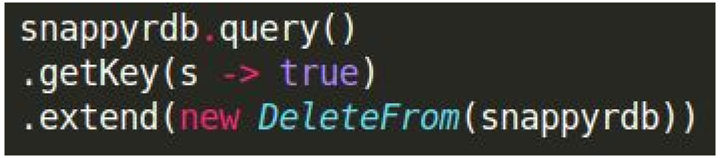
getKeyValue: Observable<DB> -> Observable<(key, value)>

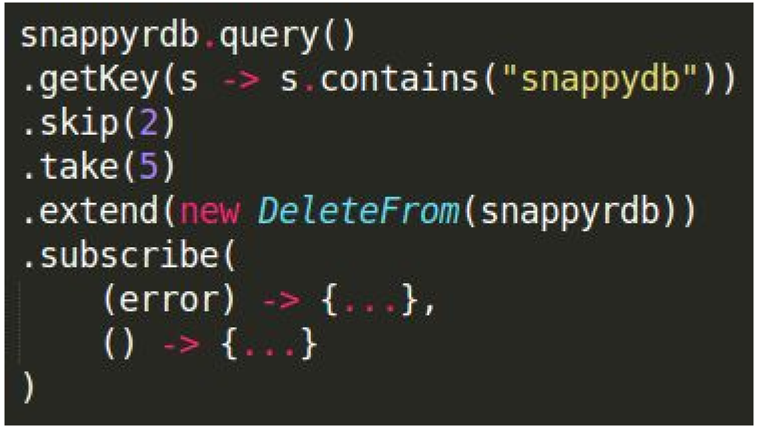
Consuming key values

PutIn: Observable<(key, value)> -> Observable<DB> // SnappyrQuery



```java
public class PutIn <T> implements Func1<Observable.OnSubscribe<Map.Entry<String, T>>, SnappyrQuery> {
    SnappyrDB db;

    public PutIn(SnappyrDB db) {
        this.db = db;
    }

    @Override
    public SnappyrQuery call(final Observable.OnSubscribe<Map.Entry<String, T>> entryOnSubscribe) {

        final ReplaySubject<DB> subj = ReplaySubject.create();
        final SnappyrQuery query = new SnappyrQuery(subj);

        entryOnSubscribe.call(new Subscriber<Map.Entry<String, T>>() {
            final Subscriber<Map.Entry<String, T>> subscriber = this;

            @Override
            public void onCompleted() {
                subj.onNext(db.getDb());
                subj.onCompleted();
            }

            @Override
            public void onError(Throwable throwable) {
                subj.onError(throwable);
                subscriber.unsubscribe();
            }

            @Override
            public void onNext(Map.Entry<String, T> stringEntry) {
                 query.put(stringEntry.getKey(), stringEntry.getValue())
                .subscribe(new Action1<Throwable>() {
                    @Override
                    public void call(Throwable e) {
                        subj.onError(e);
                        subscriber.unsubscribe();
                    }
                }, new Action0() {
                    @Override
                    public void call() {
                        //Do nothing on completed of single lift
                    }
                });
            }
        });
        return query;
    }
}
```


```
snappyrdb.query()
.getKey(s -> true)
.extend(new DeleteFrom(snappyrdb))
```

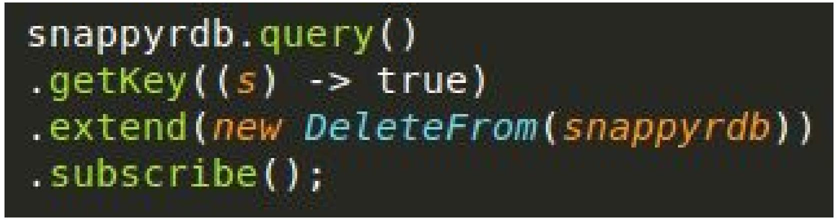

```
snappyrdb.query()
.getKey(s -> s.contains("snappydb"))
.skip(2)
.take(5)
.extend(new DeleteFrom(snappyrdb))
.subscribe(
    (error) -> {...},
    () -> {...}
)
```

Database cleanup

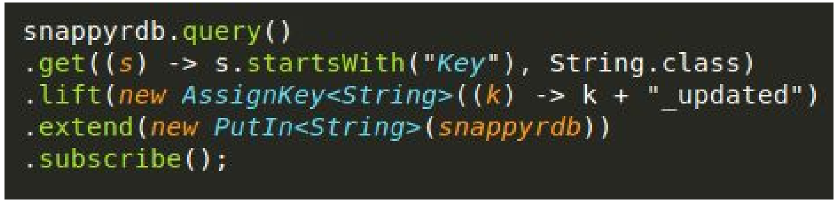
getKey: Observable<DB> -> Observable<string>

DeleteFrom: Observable<key> -> Observable<DB>

Similar to PutIn



```
snappyrdb.query()
.getKey((s) -> true)
.extend(new DeleteFrom(snappyrdb))
.subscribe();
```



```
snappyrdb.query()
.get((s) -> s.startsWith("Key"), String.class)
.lift(new AssignKey<String>((k) -> k + "_updated"))
.extend(new PutIn<String>(snappyrdb))
.subscribe();
```

Create your own operator!

Lift

SnappyrQuery -> SnappyrQuery

Observable<?> -> Observable<?>

Extend

Observable<?> -> SnappyrQuery

Summary

Easy to use

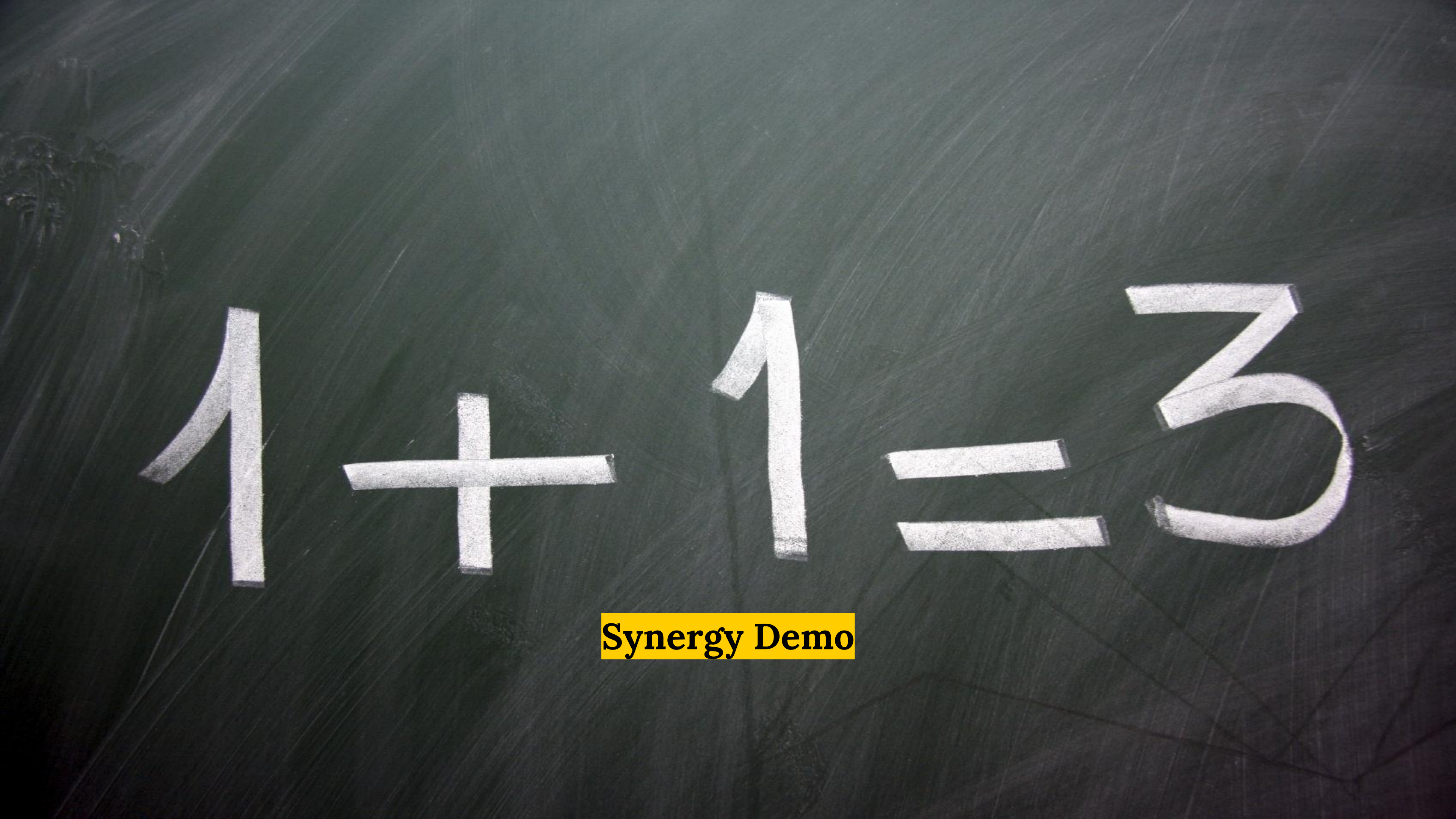
Performant

Java 7

Expresssive

Extensible

DB Monad



Synergy Demo