# RxJava Code Lab

Josef Raska - Ocado Technology

# Agenda

- Motivation
- Brief overview of ReactiveX
- **Let's code!**
- Coding
- Break
- Coding

# Why this Code Lab?

- Rx is powerful tool
- Rx is trendy
- We use Rx in Ocado and we want to use it more
- Different way of thinking
- Difficult to understand
- May bring lot of confusion

# Why Rx?

```java
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}
```

# Why Rx?

```java
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}

UserManager um = new UserManager();
System.out.println(um.getUser());
```

# Why Rx?

```java
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}

UserManager um = new UserManager();
System.out.println(um.getUser());


um.setName("John Doe");
System.out.println(um.getUser());
```

# Why Rx?

```java
interface UserManager {
  User getUser();
  void setName(String name); // <- now async
  void setAge(int age);
}

UserManager um = new UserManager();
System.out.println(um.getUser());


um.setName("John Doe");
System.out.println(um.getUser());
```

# Why Rx?

```
interface UserManager {
  User getUser();
  void setName(String name, Runnable callback);
  void setAge(int age, Runnable callback);
}
```

# Why Rx?

```java
interface UserManager {
  User getUser();
  void setName(String name, Runnable callback);
  void setAge(int age, Runnable callback);
}

UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new Runnable() {
  @Override public void run() {
    System.out.println(um.getUser());
  }
});
```

# Why Rx?

```java
interface UserManager {
  User getUser();
  void setName(String name, Listener callback);
  void setAge(int age, Listener callback);

  interface Listener {
    void success();
    void failure(IOException e);
  }
}
```

# Why Rx?

```java
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());
  }

  @Override public void failure(IOException e) {
    // TODO: handle error
  }
});
```

# Why Rx?

```java
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new UserManager.Listener() {
    @Override public void success() {
        System.out.println(um.getUser());
    }

    @Override public void failure(IOException e) {
        // TODO: handle error
    }
});

um.setAge( age: 33, new UserManager.Listener() {
    @Override public void success() {
        System.out.println(um.getUser());
    }

    @Override public void failure(IOException e) {
        // TODO: handle error
    }
});
```

# Why Rx?

```java
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());

    um.setAge( age: 33, new UserManager.Listener() {
      @Override public void success() {
        System.out.println(um.getUser());
      }

      @Override public void failure(IOException e) {
        // TODO: handle error
      }
    });
  }

  @Override public void failure(IOException e) {
    // TODO: handle error
  }
});
```

# Why Rx?

```java
private TextView textView;
private UserManager um = new UserManager();

void updateUserName() {
  um.setName( name: "John Doe", new UserManager.Listener() {
    @Override
    public void success() {
      runOnUiThread(new Runnable() {
        @Override
        public void run() {
          textView.setText(um.getUser().toString());
        }
      });
    }

    @Override
    public void failure(IOException e) {
      // TODO: handle error
    }
  });
```

# Why Rx?

Can you model your whole system synchronously?

- ● Removes the web of callbacks
- ● Declarative
- ● Simple asynchronous composing
- ● Threading abstraction
- ● More readable code
- ● Less bugs

**Reactive Streams**

JDK9 `j.u.c.Flow`

Akka

Reactor

**RxJava**

RxSwift

Rx.NET

**ReactiveX**

# RxJava ⊂ ReactiveX

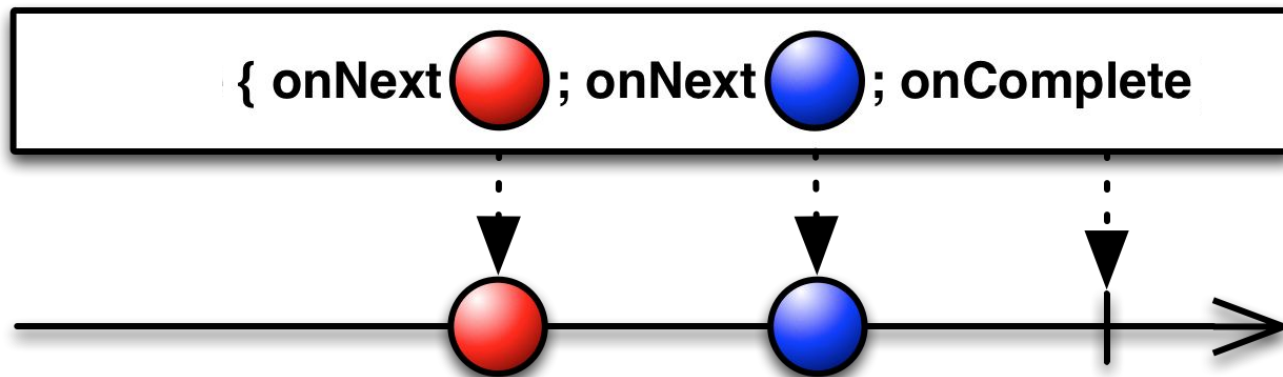"ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences."

- Observer pattern
- Concepts language independent

# Observable<T>

```
onNext(T)
onError(Throwable)
onCompleted()
```



{ onNext ●; onNext ●; onComplete

http://rxmarbles.com/

# Observable is "dual" to Iterable

| event | Iterable (pull) | Observable (push) |
|---|---|---|
| retrieve data | `T next()` | `onNext(T)` |
| discover error | throws `Exception` | `onError(Exception)` |
| complete | `!hasNext()` | `onCompleted()` |

# Why Rx?

```java
interface UserManager {
  User getUser();
  void setName(String name, Listener callback);
  void setAge(int age, Listener callback);

  interface Listener {
    void success();
    void failure(IOException e);
  }
}
            ↓   ↓   ↓

interface UserManager {
  Observable<User> getUser();
  Observable<User> setName(String name);
  Observable<User> setAge(int age);
}
```

# Why Rx?

```java
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new UserManager.Listener() {
    @Override public void success() {
        System.out.println(um.getUser());
    }

    @Override public void failure(IOException e) {
        // TODO: handle error
    }
});

um.setAge( age: 33, new UserManager.Listener() {
    @Override public void success() {
        System.out.println(um.getUser());
    }

    @Override public void failure(IOException e) {
        // TODO: handle error
    }
});
```

# Why Rx?

```java
UserManager um = new UserManager();

um.setName("John Doe")
  .concatWith(um.setAge(33))
  .subscribe(System.out::println);
```

# Why Rx?

```java
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());

    um.setAge( age: 33, new UserManager.Listener() {
      @Override public void success() {
        System.out.println(um.getUser());
      }

      @Override public void failure(IOException e) {
        // TODO: handle error
      }
    });
  }

  @Override public void failure(IOException e) {
    // TODO: handle error
  }
});
```

# Why Rx?

```java
UserManager um = new UserManager();

um.setName("John Doe")
  .flatMap((user) -> um.setAge(33))
  .subscribe(System.out::println);
```

# Why Rx?

```java
private TextView textView;
private UserManager um = new UserManager();

void updateUserName() {
  um.setName( name: "John Doe", new UserManager.Listener() {
    @Override
    public void success() {
      runOnUiThread(new Runnable() {
        @Override
        public void run() {
          textView.setText(um.getUser().toString());
        }
      });
    }

    @Override
    public void failure(IOException e) {
      // TODO: handle error
    }
  });
```

# Why Rx?

```java
TextView textView;
UserManager um = new UserManager();

void updateUserName() {
  um.setName("John Doe")
    .subscribeOn(schedulers.io())
    .observeOn(schedulers.mainThread())
    .subscribe(user -> textView.setText(user.toString()));
```

# Codelab project

- RxJava 2
- Code playground is unit test
- Separate tasks for particular areas
- Each task has a solution in `solutions` package
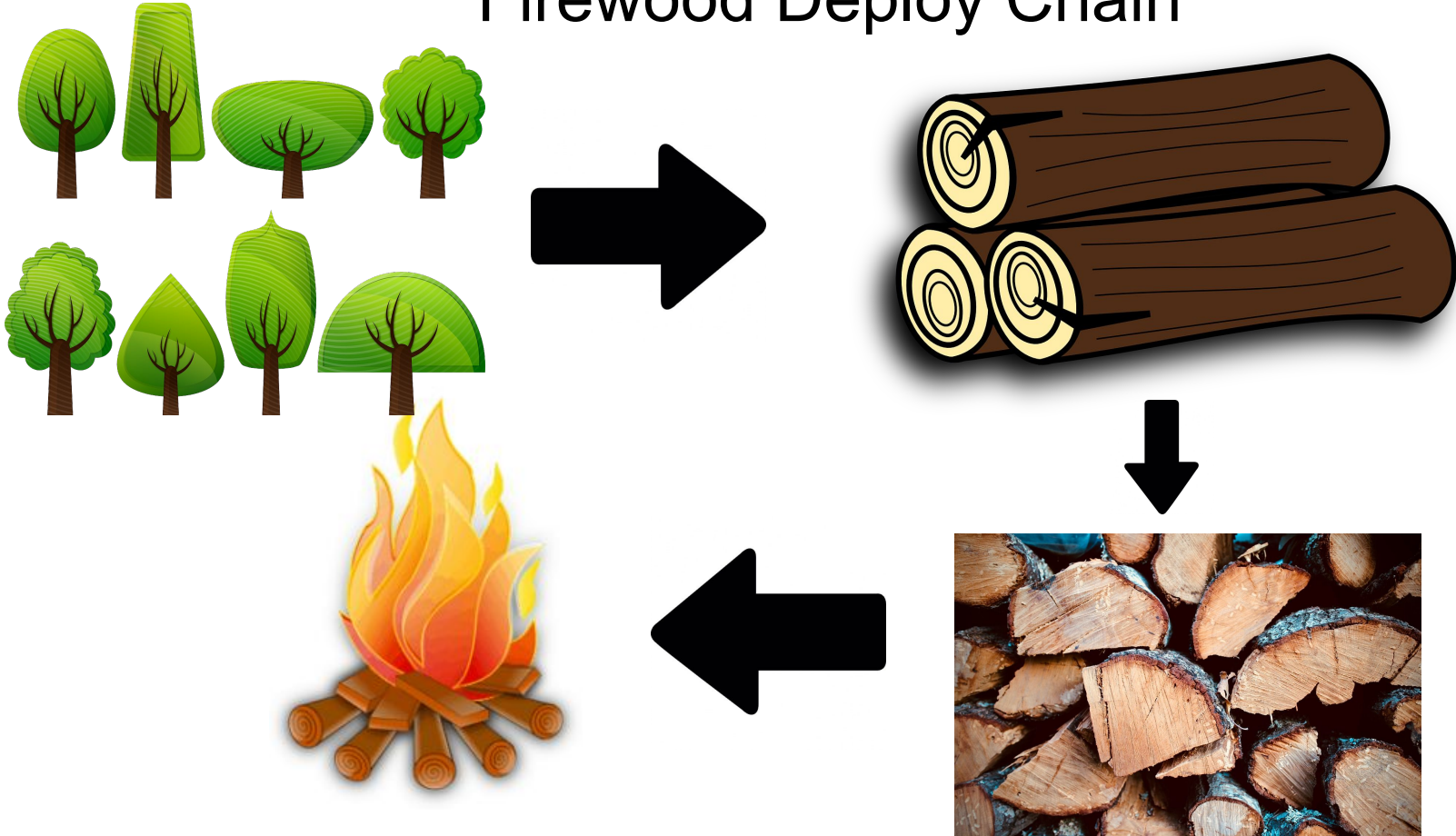
# Project setup

- Running unit tests with icon → 
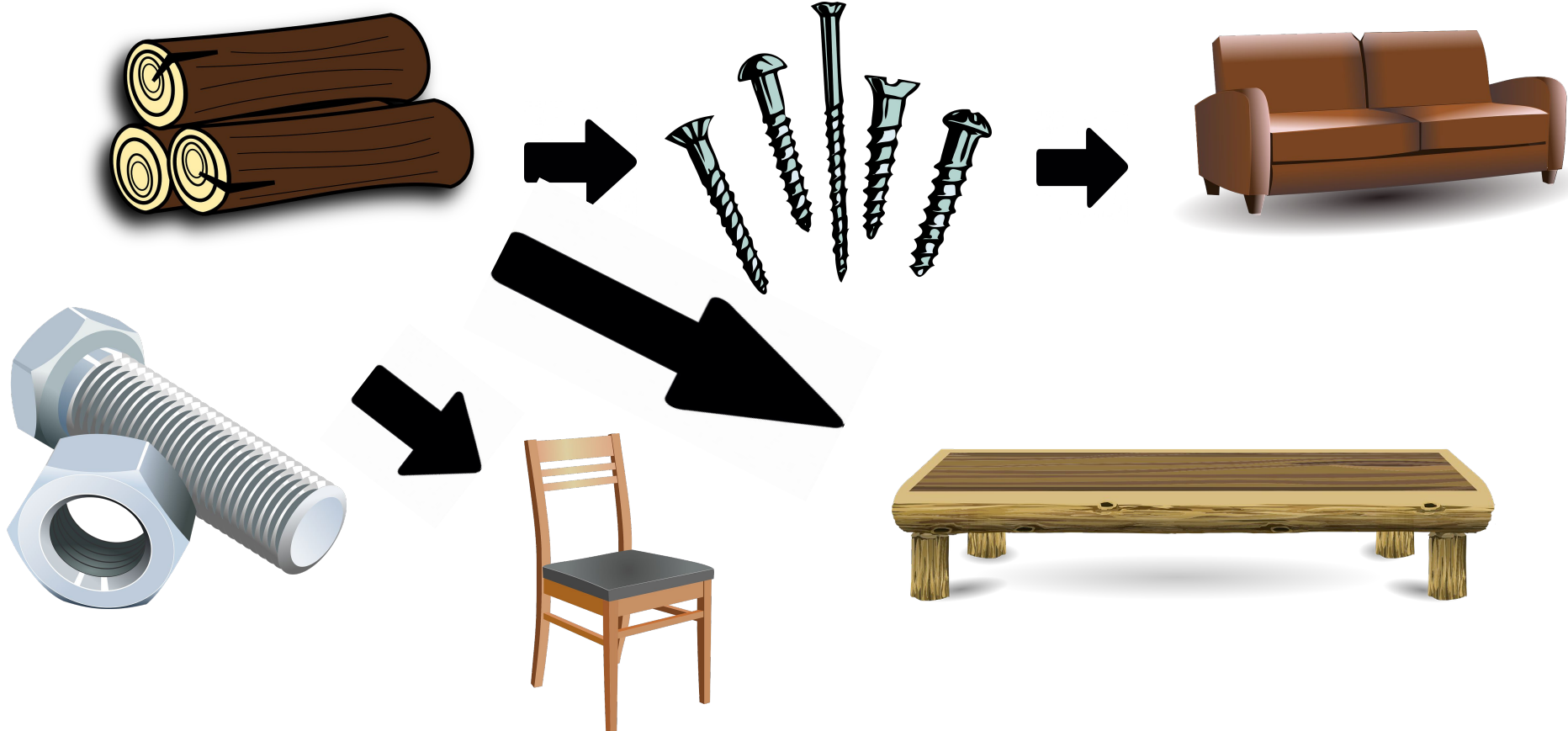- Is everyone ready?
- Entertainment meanwhile:

  http://reactivex.io/intro.html

  http://rxmarbles.com/

Firewood Deploy Chain

# Now let's make some furniture

# Hot vs. Cold Observable

- **Producer**
- Gets values and pass them to `observer.next(value)`
- Hot producer is active regardless the subscriptions
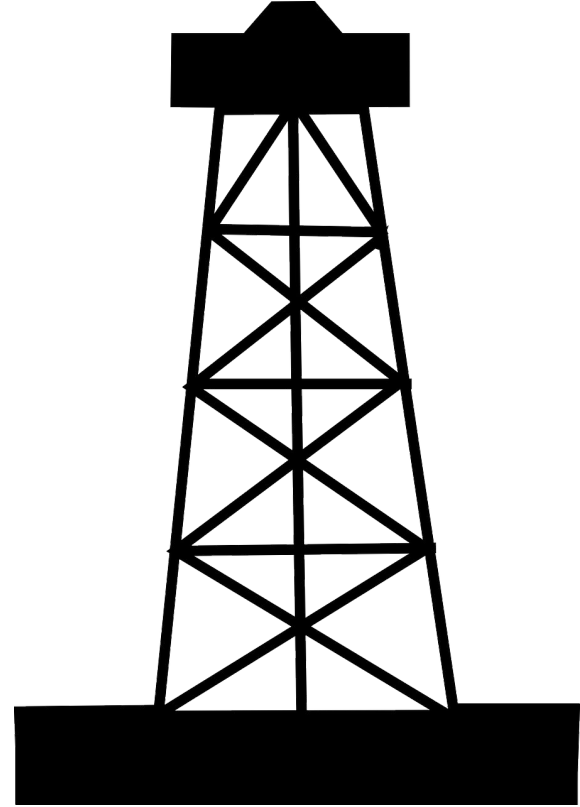- Cold producer gets activated by subscription

# Hot vs. Cold Observable

| Emits whether the observer is ready or not | Emits at controlled rate whenever requested by observer |
|---|---|
| Mouse & keyboard events<br>System events<br>Stock prices<br>Time | Iterable<br>Web request<br>Database query<br>Reading file |

# River - Hot Observable

# Oil Well - Cold Observable

# Reactive Streams

JDK9 `j.u.c.Flow`

Akka

Reactor

## RxJava

RxSwift

Rx.NET

## ReactiveX

# References

- First place to go: http://reactivex.io/

- Marble diagrams for operators - http://rxmarbles.com

- https://github.com/Reactive-extensions/Rx.Net

- https://github.com/ReactiveX

- Dávid Karnok's blog - http://akarnokd.blogspot.com

- http://www.reactive-streams.org/

- https://github.com/Froussios/Intro-To-RxJava

- https://github.com/jraska/RxJava-Codelab