Java #2

Interfaces

Interface ?

```
interface CanFly {

    FlightStatus flyTo(final Point destination);
}
```

```
abstract class Sup {}

class HomeLander extends Sup implements CanFly { /* ... */}
```

Interface ?

- Expression of a **feature**
- Prescription of an API

or

- Definition of a **contract**
- Feature vs Standardization

50 shades of Classes

- **Class**: Nature of the object
- **Abstract class**: nature + sharing of common implementation
- **Interface**: additional, transversal features

Interface of Class or Abstract ?

- All super-heroes are **individuals**.
- Some **super-heroes** can **shoot lasers from their eyes**.
- **Homelander** is a super-hero.

Interface & enums

- Enums are classes
- Therefore can implements interfaces.
- Good way of being both generic and closed.

Demo

Default Methods

What if... interface could hold code ?

- Provides a `default` implementation of a method in an interface.

Demo

Functional Interfaces

- Interfaces with a single abstract method.
- Default methods are not abstract.
- `@FunctionalInterface`.

Functional Interfaces

```
@FunctionalInterface
interface CanFly {

    MovementStatus flyTo(final Point destination);
}
```

Functional Interfaces

```
@FunctionalInterface
interface CanFly {

    MovementStatus flyTo(final Point destination);

    default MovementStatus moveTo(final Point destination) {
        return flyTo(destination);
    }

    default String asDuck() {
        return "_O<";
    }

}
```

Functional Interfaces

From the JDK:

- `Runnable` : () -> void
- `Supplier` : () -> something
- `Consumer` : (something) -> void
- `BiConsumer` : (something, anotherSomething) -> void
- `Function` : (something) -> somethingElse
- `BiFunction` : (something, anotherSomething) -> somethingElse

Anonymous Class

```
    class Event {
        public final Component source;
        public final Button button;
    }

    interface EventListener {
        void onEvent(final Event event);
    }
```

Demo

Lambdas

What if... we could be free of the boiler plate for anonymous classes?

Lamdba notation

- Syntactic sugar
- Mapped to function interfaces

Demo

Lambdas

```
Supplier<Test> supplier = () -> new Test();
```

```
Function<Argument, TestWithArgument> supplier = arg -> new TestWithArgument(arg);
```

```
var Consumer<Something> = (Something arg) -> arg.doStuff;
```

- How to simplify ?

Lambdas

Method Reference

```
Supplier<Test> supplier = Test::new;
```

```
Function<Argument, TestWithArgument> supplier supplier = TestWithArgument::new;
```

```
var Consumer<Something> = Something::doStuff;
```

Stream

- Apply functional-style operations on a stream of elements
- Created from:
  - `Collection#stream`
  - `Stream#of`
  - `StreamSupport#xxx`

Demo

Lambdas and exceptions

- Library's FI don't throw.

Lambdas and exceptions

- Library's FIs don't throw
- Redefine throwing functional interfaces for core needs
- Stream only accepts library FIs
- Needs an extension mechanism

Interface Forwarding

- How can we extends interfaces
- Even the ones from the library
- With few lines of code

Interface Forwarding

```
public interface Opt<ELEMENT_TYPE> {
    boolean isPresent();
    ELEMENT_TYPE get() throws RuntimeException;
}
```

Interface Forwarding

```
public class OptImpl<ELEMENT_TYPE> implements Opt<ELEMENT_TYPE> {
    private final ELEMENT_TYPE value;

    public OptImpl(final ELEMENT_TYPE value) {
        this.value = value;
    }

    public boolean isPresent() {
        return value != null;
    }

    public ELEMENT_TYPE get() throws RuntimeException {
        if (!isPresent()) {
            throw new RuntimeException("Whoopsey");
        }
        return value;
    }
}
```

Interface Forwarding

With helpers:

```
public interface Opt<ELEMENT_TYPE> {
    boolean isPresent();
    ELEMENT_TYPE get() throws RuntimeException;

    static <ELEMENT_TYPE> Opt<ELEMENT_TYPE> of(final ELEMENT_TYPE value) {
        return new OptImpl(value)
    }

    static <ELEMENT_TYPE> Opt<ELEMENT_TYPE> opt(final ELEMENT_TYPE value) {
        return Opt.of(value);
    }
}
```

State Prescription

What if ... interfaces could manipulate state ?

- How to implement ?

Demo