Exceptions handling

Unfortunatelly, exceptions handling in this library is a bit difficult in some places, but that have at least two reasons: flexibility and usability.

"In place" handling

In case you know, where exceptions are happening, you may use several tools for exceptions catching:

- Catching with result
- Catching with callback

Catching with result

If you prefer to receive `Result` objects instead of some weird callbacks, you may use the next syntax:

```
safelyWithResult {
    // do something
}.onSuccess { // will be called if everything is right
    // handle success
}.onFailure { // will be called if something went wrong
    // handle error
    it.printStackTrace()
}.getOrThrow() // will return value or throw exception
```

Catching with callback

Also there is more simple (in some cases) way to handle exceptions with callbacks:

```
safely(
  {
      // handle error
      it.printStackTrace()
```

```
      null // return value
  }
) {
    // do something
}
```

Bonus: different types of handling

There are two types of handling:

- Just safely - when you are using something to obviously retrieve value or throw exception. When handling callback has been skipped, it will throw exception by default. For example:

```
safely(
    {
        it.printStackTrace()
        "error"
    }
) {
    error("Hi :)") // emulate exception throwing
    "ok"
} // result will be with type String
```

- Safely without exceptions - almost the same as `safely`, but this type by default allow to return nullable value (when exception was thrown) instead of just throwing (as with `safely`):

```
safelyWithouExceptions {
    // do something
} // will returns nullable result type
```

Global exceptions handling

The most simple way to configure exceptions handling is to change `CoroutineContext` when you are creating your `CoroutineScope` for bot processing:

```
val bot = telegramBot("TOKEN")


bot.buildBehaviour (
```

```
    scope = scope,
    defaultExceptionsHandler = {
        it.printStackTrace()
    }
) {
    // ...
}
```

OR

```
val bot = telegramBotWithBehaviour (
    "TOKEN",
    scope = scope,
    defaultExceptionsHandler = {
        it.printStackTrace()
    }
) {
    // ...
}
```

Here we have used `ContextSafelyExceptionHandler` class. It will pass default handling of exceptions and will call the block in most cases when something inside of your bot logic has thrown exception.

Revision #2
Created 3 September 2021 17:06:20 by InsanusMokrassar
Updated 4 September 2021 11:33:33 by InsanusMokrassar