CLEAN CODE

GESTIÓN DE ERRORES

Daniel Blanco Calviño

LA GESTIÓN DE ERRORES ES FUNDAMENTAL

- El código debe ser limpio, pero también robusto.
- La gestión de errores puede ensuciar mucho el código, por lo que hay que prestarle especial atención.

- Devolver **excepciones** en lugar de códigos de error.
- Usar **excepciones *unchecked***
- **No devolver ni pasar null.**

EXCEPCIONES EN LUGAR DE CÓDIGOS DE ERROR

```
// Ejemplo del libro Clean Code, de Robert C Martin
public class DeviceController {

    public void sendShutDown() {
        DeviceHandle handle = getHandle(DEV1);
        // Check the state of the device
        if(handle != DeviceHandle.INVALID) {
            // Save the device status to the record field
            DeviceRecord record = retrieveDeviceRecord(handle);
            //  If not suspend, shut down
            if(record.getStatus() != DEVICE_SUSPENDED) {
                pauseDevice(handle);
                clearDeviceWorkQueue(handle);
                closeDevice(handle);
            }else {
                logger.log("Device suspended. Unable to shut down");
            }
        }else {
            logger.log("Invalid handle for: " + DEV1.toString());
        }
    }
```

EXCEPCIONES EN LUGAR DE CÓDIGOS DE ERROR

Usando excepciones no tienes por qué tratar el error en el momento.

```
// Ejemplo del libro Clean Code, de Robert C Martin
public class DeviceController {

    public void sendShutDown() {
        DeviceHandle handle = getHandle(DEV1);
        // Check the state of the device
        if(handle != DeviceHandle.INVALID) {
            // Save the device status to the record field
            DeviceRecord record = retrieveDeviceRecord(handle);
            //  If not suspend, shut down
            if(record.getStatus() != DEVICE_SUSPENDED) {
                pauseDevice(handle);
                clearDeviceWorkQueue(handle);
                closeDevice(handle);
            }else {
                logger.log("Device suspended. Unable to shut down");
            }
        }else {
            logger.log("Invalid handle for: " + DEV1.toString());
        }
    }
```

```
public void sendShutDown() {
    try {
        tryToShutDown();
    } catch(DeviceShutDownError e) {
        logger.log(e);
    }
}

private void tryToShutDown() throws DeviceShutDownError {
    DeviceHandle handle = getHandle(DEV1); // Throws DeviceShutDownError
    DeviceRecord record = retrieveDeviceRecord(handle);

    pauseDevice(handle);
    clearDeviceWorkQueue(handle);
    closeDevice(handle);
}
```

UNCHECKED EXCEPTIONS

- Se trata de excepciones que heredan de ***RuntimeException***. Ejemplo, la *NullPointerException*.
- Se pueden tratar con try-catch, pero no es estrictamente necesario.
- Las **excepciones checked**, son las excepciones "normales", heredan de *Exception*. **Si no las tratas el programa no compila.**
- Recomendado usar checked Exceptions si se trata de una librería crítica, que sea de obligado cumplimiento tratar las excepciones.

UNCHECKED EXCEPTIONS

```
public class Driver {
    public void startDriving() {
        try {
            car.start();
        } catch(BatteryFailureException | EngineFailureException e) {
            logger.log("Could not start driving. Car failure: " + e);
        }
    }
}

public class Car {
    public void start() throws BatteryFailureException, EngineFailureException{
        battery.connect();
        engine.start();
    }
}

public class Battery {
    public void connect() throws BatteryFailureException;
}

public class Engine {
    public void start() throws EngineFailureException;
}
```

NO DEVOLVER NULL

```java
public void processRequestBatch() {
    List<Request> requestList = getRequestList();
    if(requestList != null) {
        for(Request r : requestList) {
            processRequest(r);
        }
    }
}
```

```java
public void processRequestBatch() {
    List<Request> requestList = getRequestList();
    for(Request r : requestList) {
        processRequest(r);
    }
}
```

- *NullPointerException* es una unchecked exception.
- Muchos errores son debido a devolver null y no tratarlo correctamente.

NO PASAR NULL

```java
public Point getMiddlePoint(Point a, Point b) {
    if(a != null && b != null) {
        Double resultPointX = (a.getX() - b.getX()) / 2;
        Double resultPointY = (a.getY() - b.getY()) / 2;

        return newPoint(resultPointX, resultPointY);
    }

    // Return?
}
```

```java
public Point getMiddlePoint(Point a, Point b) {
    Double resultPointX = (a.getX() - b.getX()) / 2;
    Double resultPointY = (a.getY() - b.getY()) / 2;

    return newPoint(resultPointX, resultPointY);
}
```

- No hay una manera perfecta de tratar con el problema de la recepción de *null* en los parámetros de una función.

CLEAN CODE

GESTIÓN DE ERRORES

Daniel Blanco Calviño