CLEAN CODE

FUNCIONES

Daniel Blanco Calviño

CARACTERÍSTICAS DE BUENAS FUNCIONES

- **Pequeñas**. Muy pequeñas.
- Hacen **una sola cosa.**
- **Nivel de abstracción** único
- Reciben **pocos argumentos**
- No tienen efectos secundarios.
- Devuelven **excepciones** en lugar de códigos de error.

```
public Boolean shouldIWrite100LineFunctions() {
    return false;
}
```

LAS FUNCIONES HACEN UNA COSA

```
public Integer readNumbersFromFileAndCalculateTotal(String fileUrl) {
    List<String> numbers =
            Files.readAllLines(Paths.get(fileUrl), StandardCharsets.UTF_8);

    Integer total = 0;
    for (String numberString : numbers) {
        total += Integer.valueOf(numberString);
    }

    return total;
}
```

LAS FUNCIONES HACEN UNA COSA

```java
public Integer getMonthlySales() {
    List<String> sales = readNumbersFromFile(MONTHLY_SALES_FILE_PATH);
    return calculateSummatory(sales);
}

public List<String> readNumbersFromFile(String fileUrl) {
    return Files.readAllLines(Paths.get(fileUrl), StandardCharsets.UTF_8);
}

public Integer calculateSummatory(List<String> values) {
    Integer total = 0;
    for (String numberString : numbers) {
        total += Integer.valueOf(numberString);
    }

    return total;
}
```

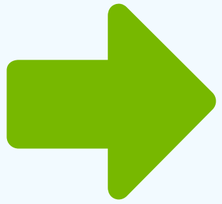
UN ÚNICO NIVEL DE ABSTRACCIÓN

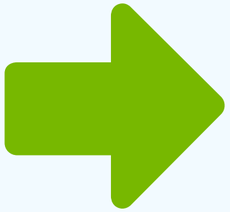
```
public class Person {

    public void drive(Car car) {
        car.openDoor();
        car.setDriver(this);
        car.getBattery().connect();
        car.getEngine().start();
    }
}
```

```
public class Car {
    Battery battery;
    Engine engine;

    public void start() {
        battery.connect();
        engine.start();
    }
}
```

```
public class Person {

    public void drive(Car car) {
        car.openDoor();
        car.setDriver(this);
        car.start();
    }
}
```

UN ÚNICO NIVEL DE ABSTRACCIÓN





```java
public Integer getMonthlySales() {
    List<String> sales = readNumbersFromFile(MONTHLY_SALES_FILE_PATH);
    return calculateSummatory(sales);
}



public List<String> readNumbersFromFile(String fileUrl) {
    return Files.readAllLines(Paths.get(fileUrl), StandardCharsets.UTF_8);
}

public Integer calculateSummatory(List<String> values) {
    Integer total = 0;
    for (String numberString : numbers) {
        total += Integer.valueOf(numberString);
    }

    return total;
}
```

POCOS ARGUMENTOS

- Se deben evitar las funciones con más de 3 argumentos.
- Un gran número de argumentos suele indicar una mala encapsulación.

```
public User createUser(String username, String password, String firstName,
        String lastName, String address);

public User createUser(UserFormData userFormData);
```

```
public Circle createCircle(double x, double y, double radius);

public Circle createCircle(Point center, double radius);
```

EVITAR LOS EFECTOS SECUNDARIOS

- Evitar las acciones "ocultas" de las funciones

```
public Boolean login(User user) {
    User databaseUser = userRepository.findUserByUsername(user.getUsername());

    if(validCredentials(user, databaseUser)) {
        return true;
    }
    if(checkNumberOfTries(user.getIp()) > MAXIMUM_TRIES) {
        blockIpAddress(user.getIp());
    }

    return false;
}
```

- Sería preferible llamarle *loginAndBlockIpIfMaximumTriesExceded,* pero viola el principio de "una sola cosa" => Refactorizar en dos funciones.

EXCEPCIONES EN LUGAR DE CÓDIGOS DE ERROR

- Si devuelves códigos de error, debes tratarlo al momento, con excepciones no.

```
public void restoreLastDatabaseBackup() {
    if(backupCurrentData() == OK) {
        Backup backup = getLastBackup();
        if(backup.restore() == OK) {
            Logger.log("Backup restored");
        }else {
            Logger.log("Error restoring backup")
        }
    }else {
        Logger.log("Could not backup current data. Restore backup canceled.")
    }
}
```

EXCEPCIONES EN LUGAR DE CÓDIGOS DE ERROR

- Utilizar excepciones te permite tener el camino de ejecución correcto separado del camino de errores.

```
public void safelyRestoreLastDatabaseBackup() {
    try {
        backupCurrentData();            // Throws CreateBackupException
        Backup backup = getLastBackup();
        backup.restore();               //Throws RestoreBackupException
        Logger.log("Backup restored");

    } catch (CreateBackupException e) {
        Logger.log("Could not backup current data. Restore backup canceled.")
    } catch (RestoreBackupException e) {
        Logger.log("Error restoring backup")
    }
}
```

EXCEPCIONES EN LUGAR DE CÓDIGOS DE ERROR

- Lo ideal sería separar también los try-catch en una función.

```
public void safelyRestoreLastDatabaseBackup() {
    try {
        restoreLastDatabaseBackup();

    } catch (CreateBackupException e) {
        Logger.log("Could not backup current data. Restore backup canceled.")
    } catch (RestoreBackupException e) {
        Logger.log("Error restoring backup")
    }
}

private void restoreLastDatabaseBackup() throws CreateBackupException, RestoreBackupException {
    backupCurrentData();
    Backup backup = getLastBackup();
    backup.restore();
}
```

CLEAN CODE

FUNCIONES

Daniel Blanco Calviño