Pokédex project : SOLID Principles

Single-purpose Responsibility

```java
public Pokemon() {
}

public Pokemon(int number, IGetPokemonInformation getPokemonInformation) {
    this.number = number;
    pokemonInformation = getPokemonInformation;
}

public void setPokemonInformation() {
    setName(pokemonInformation.getName(this.number));
    setHeight(pokemonInformation.getHeight(this.number));
    setWeight(pokemonInformation.getWeight(this.number));
}

public void showPokemonInformation() {
    System.out.println("============================");
    System.out.println("Pokémon #" + this.getNumber());
    System.out.println("Nom : " + this.getName());
    System.out.println("Taille : " + this.getHeight());
    System.out.println("Poids : " + this.getWeight());
    System.out.println("============================\n");
}
```

Every function of the code has a single aim and is used for one thing only. Here it eases the reading of the code and gives a better code architecture as it gets more and more sophisticated. For instance, *showPokemonInformation()* is a basic function that prints the information we have on the pokemon. It is easily usable and it is easy to know what it does when it is called in another class.

```java
public static void main(String[] args) {

    if (args.length == 1 || Integer.parseInt(args[0]) > 5) {
        System.out.println("Vous cherchez les informations sur le Pokémon n° : " + args[0]);
        HTTPRequest httpRequest = new HTTPRequest(Integer.parseInt(args[0]));
        PokemonWithDescription pokemon = new PokemonWithDescription(Integer.parseInt(args[0]), httpRequest);
        pokemon.setPokemonInformation();
        pokemon.showPokemonInformation();
```

Open-Closed

```java
public class Pokemon {
    private int number;
    private String name;
    private long height;
    private long weight;

    private IGetPokemonInformation pokemonInformation;

    public int getNumber() { return number; }

    public String getName() { return name; }

    public long getHeight() { return height; }

    public long getWeight() { return weight; }

    public void setNumber(int number) { this.number = number; }

    public void setName(String name) { this.name = name; }

    public void setHeight(long height) { this.height = height; }

    public void setWeight(long weight) { this.weight = weight; }


    public Pokemon() {
    }

    public Pokemon(int number, IGetPokemonInformation getPokemonInformation) {
        this.number = number;
        pokemonInformation = getPokemonInformation;
    }
```

```java
public class PokemonWithDescription extends Pokemon {
    private int number;
    private String name;
    private long height;
    private long weight;
    private String description = "";

    private IGetPokemonInformation pokemonInformation;

    public int getNumber() { return number; }

    public String getName() { return name; }

    public long getHeight() { return height; }

    public long getWeight() { return weight; }

    public String getDescription() {
        return description;
    }

    public void setNumber(int number) { this.number = number; }

    public void setName(String name) { this.name = name; }

    public void setHeight(long height) { this.height = height; }

    public void setWeight(long weight) { this.weight = weight; }

    public void setDescription(String description) { this.description = description; }

    public PokemonWithDescription() {
    }

    public PokemonWithDescription(int number, IGetPokemonInformation getPokemonInformation) {
        this.number = number;
        pokemonInformation = getPokemonInformation;
    }
```

The first pokemon class does not include a description. By respecting the Open-Closed principle, we do not modify the existing *Pokemon* class but we create a new extension of this class with a description. It gives a code that is easily improvable if we want to add new features as we have done there.

Liskov Substitution

```java
if (args.length == 1 || Integer.parseInt(args[0]) > 5) {
    System.out.println("Vous cherchez les informations sur le Pokémon n° : " + args[0]);
    HTTPRequest httpRequest = new HTTPRequest(Integer.parseInt(args[0]));
    PokemonWithDescription pokemon = new PokemonWithDescription(Integer.parseInt(args[0]), httpRequest);
    pokemon.setPokemonInformation();
    pokemon.showPokemonInformation();
```

We apply the Liskov Substitution principle by creating our classes *Pokemon* and *PokemonWithDescription* so that we can instantiate a *Pokemon* as a *PokemonWithDescription*. It gives the code more flexibility, if we want to add description afterwards to a pokemon which did not have one originally.

Interface Segregation

```java
public interface IGetPokemonInformation {

    String getName(int number);

    Long getHeight(int number);

    Long getWeight(int number);

}
```

```java
public interface IGetPokemonInformationWithDescription {
    String getName(int number);

    Long getHeight(int number);

    Long getWeight(int number);

    String getDescription(int number);
}
```

We implemented two different interfaces so that if a client does not want the pokemon description (it is the case if he uses the API) he does not implement function he will not use.

Dependency Inversion Principle

```java
public interface IGetPokemonInformation {

    String getName(int number);

    Long getHeight(int number);

    Long getWeight(int number);

    String getDescription(int number);

}
```

```java
public class SQLITERequest implements IGetPokemonInformation {

    private ResultSet rs;

    public SQLITERequest(int pokemonNumber, String database) {...}

    @Override
    public String getName(int number) {
        try {
            return this.rs.getString( s: "name");
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }
```

```java
public class Pokemon {
    private int number;
    private String name;
    private long height;
    private long weight;

    private IGetPokemonInformation pokemonInformation;
```

We use a interface to generalize pokemon's information whatever the way we use to get those. For that, our low-level classes such as *SQLiteRequest* implements the interface. The *Pokemon* class which is a high-level only uses the interface and is not dependent on the low-level one. For example later in the code we use a *HTTPRequest* instead of a *SQLiteRequest* without modifying anything inside the *Pokemon* class