Adapter Pattern

Apprentiship Program
2020

Laura Gonzalez

Laura Gonzalez Fernandez

Software Engineer in training at LifullConnect

- Culo inquieto por naturaleza
- Apasionada de los videojuegos

www.lauragonzafer.com

Contenidos

¿ Que son los Design Patterns ?

A grandes rasgos, los Design Patterns son unas técnicas para resolver problemas comunes en el desarrollo de software.

¿ Que son los Design Patterns ?

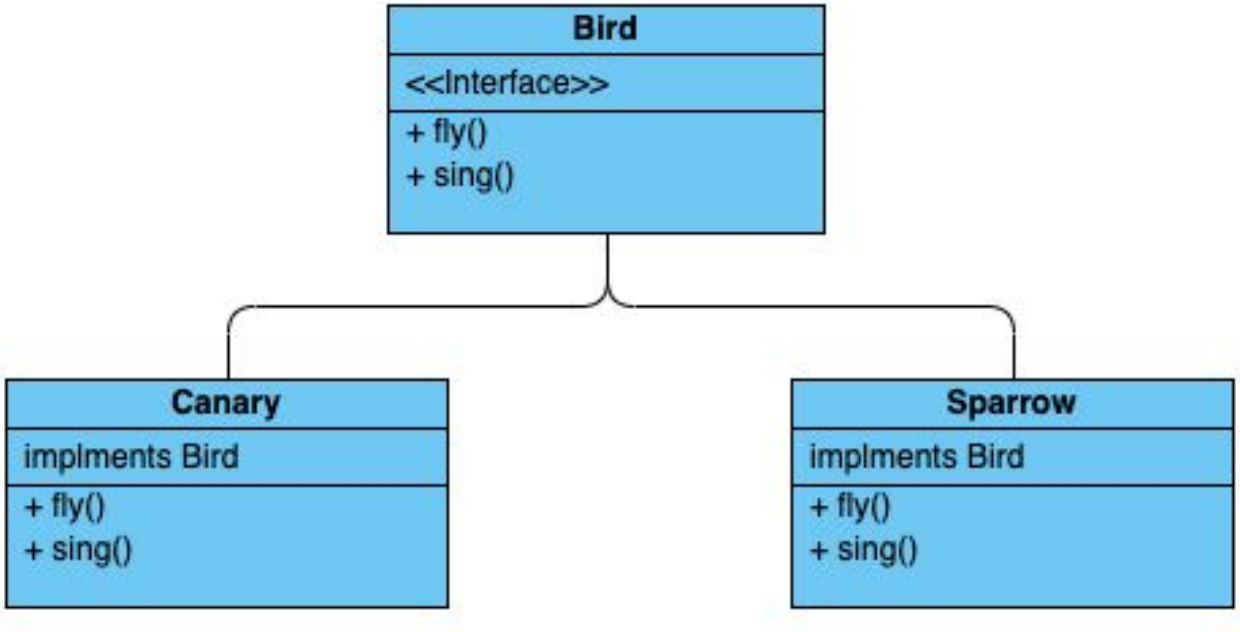
Tienen que poseer ciertas características:

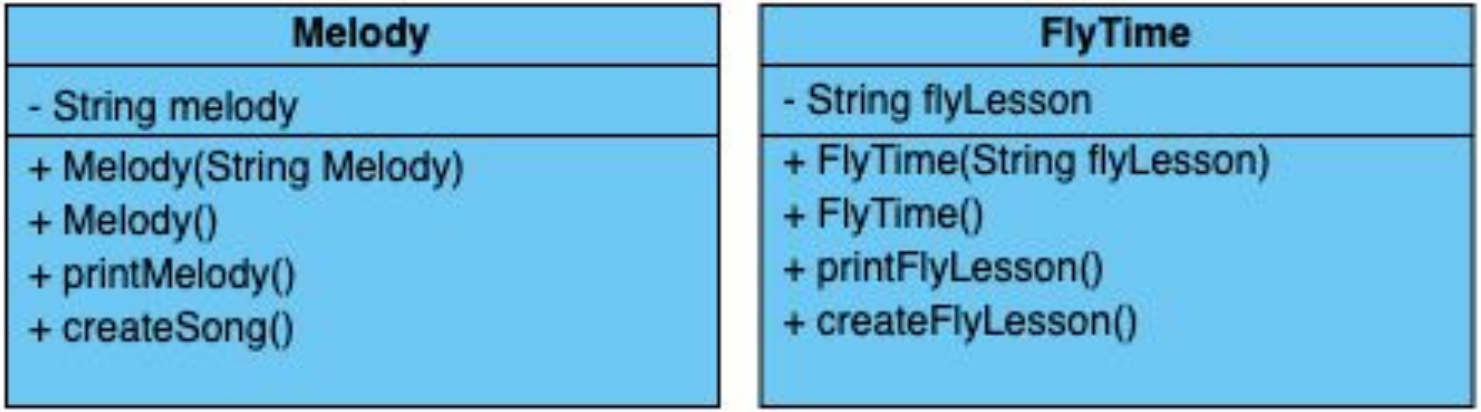
- Se debe haber comprobado su EFECTIVIDAD resolviendo problemas similares en otras ocasiones anteriores

- Debe ser REUTILIZABLE , lo que significa que se puede aplicar a diferentes problemas de diseño en distintas circunstancias

Problema a resolver

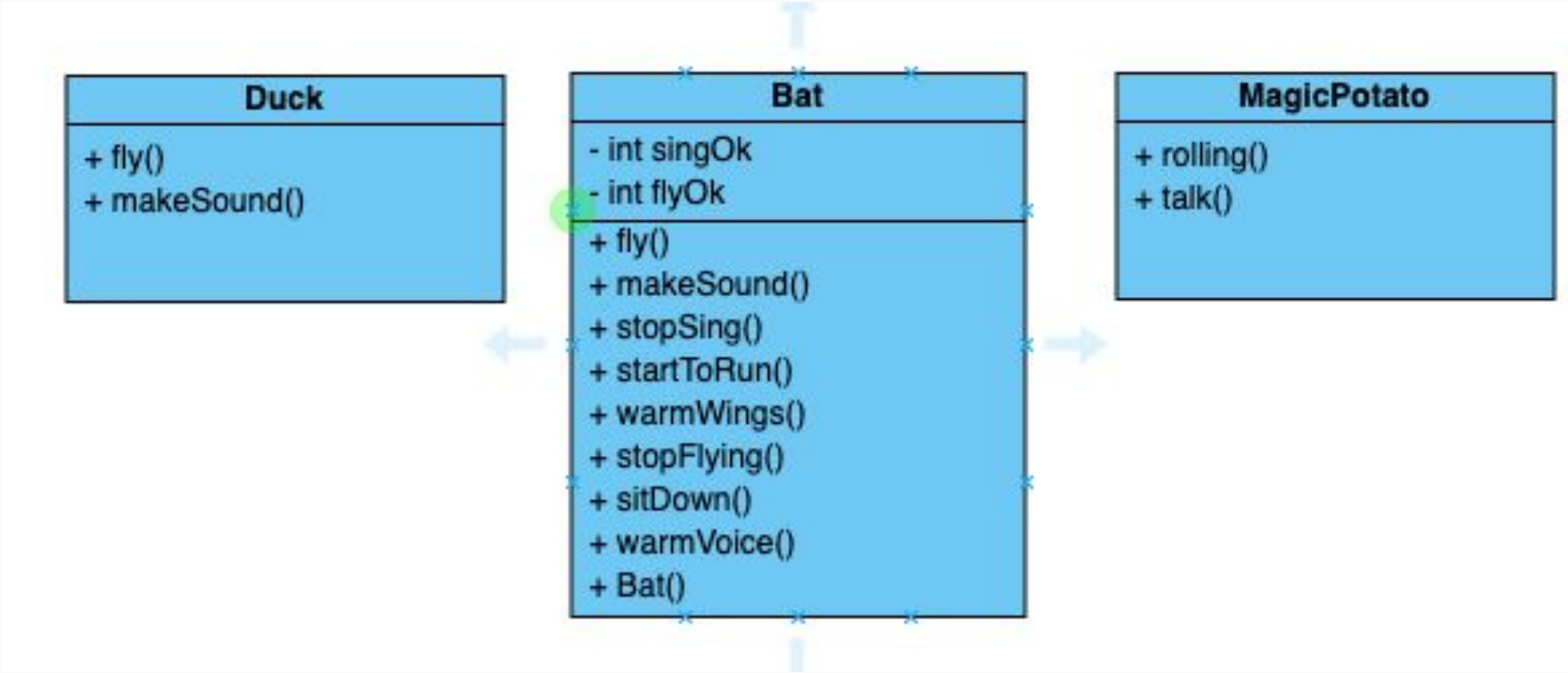
Tenemos una pequeña aplicación que genera cantos de pájaro y lecciones de vuelo. Para generarlo necesitas una lista de “Bird” que nos es proporcionada, donde nos llegan nuestros objetos “Sparrow” y “Canary”, que implementa la interfaz “Bird”, y poseen los métodos “Fly” y “Sing”


Bird
<<Interface>>
+ fly()
+ sing()
Canary
implments Bird
+ fly()
+ sing()
Sparrow
implments Bird
+ fly()
+ sing()


Melody
- String melody
+ Melody(String Melody)
+ Melody()
+ printMelody()
+ createSong()
FlyTime
- String flyLesson
+ FlyTime(String flyLesson)
+ FlyTime()
+ printFlyLesson()
+ createFlyLesson()

Problema a resolver

La aplicación tiene que ampliarse y ahora también tenemos que incluir los objetos “Bat”, “Duck” y “MagicPotato”, que no parecen tener las funciones tal y como las necesitamos, además de que no pueden ser modificados para encajar en nuestro desarrollo


Duck
+ fly()
+ makeSound()
Bat
- int singOk
- int flyOk
+ fly()
+ makeSound()
+ stopSing()
+ startToRun()
+ warmWings()
+ stopFlying()
+ sitDown()
+ warmVoice()
+ Bat()
MagicPotato
+ rolling()
+ talk()

Adapter Pattern

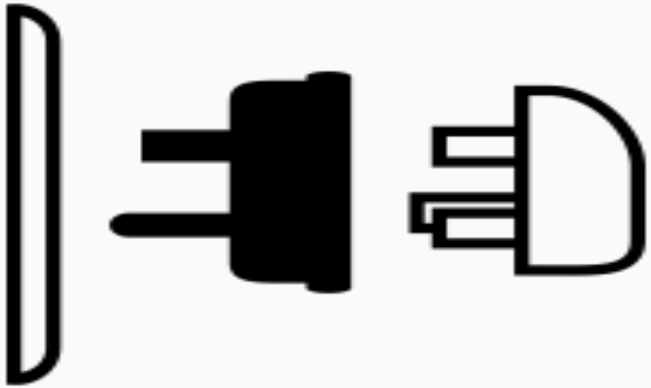
¿Que es el Adapt Pattern?

El Adapt Pattern nos permite crear una clase “adapter” que haga de intermediario entre nuestra interfaz y las nuevas clases.

Con ello podemos “simular” el comportamiento que ya teníamos sin modificar las nuevas clases.

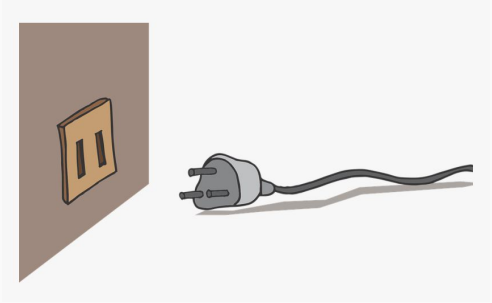
¿Que es el Adapt Pattern?

Nuestro “adapter” funcionara como un adaptador de corriente.

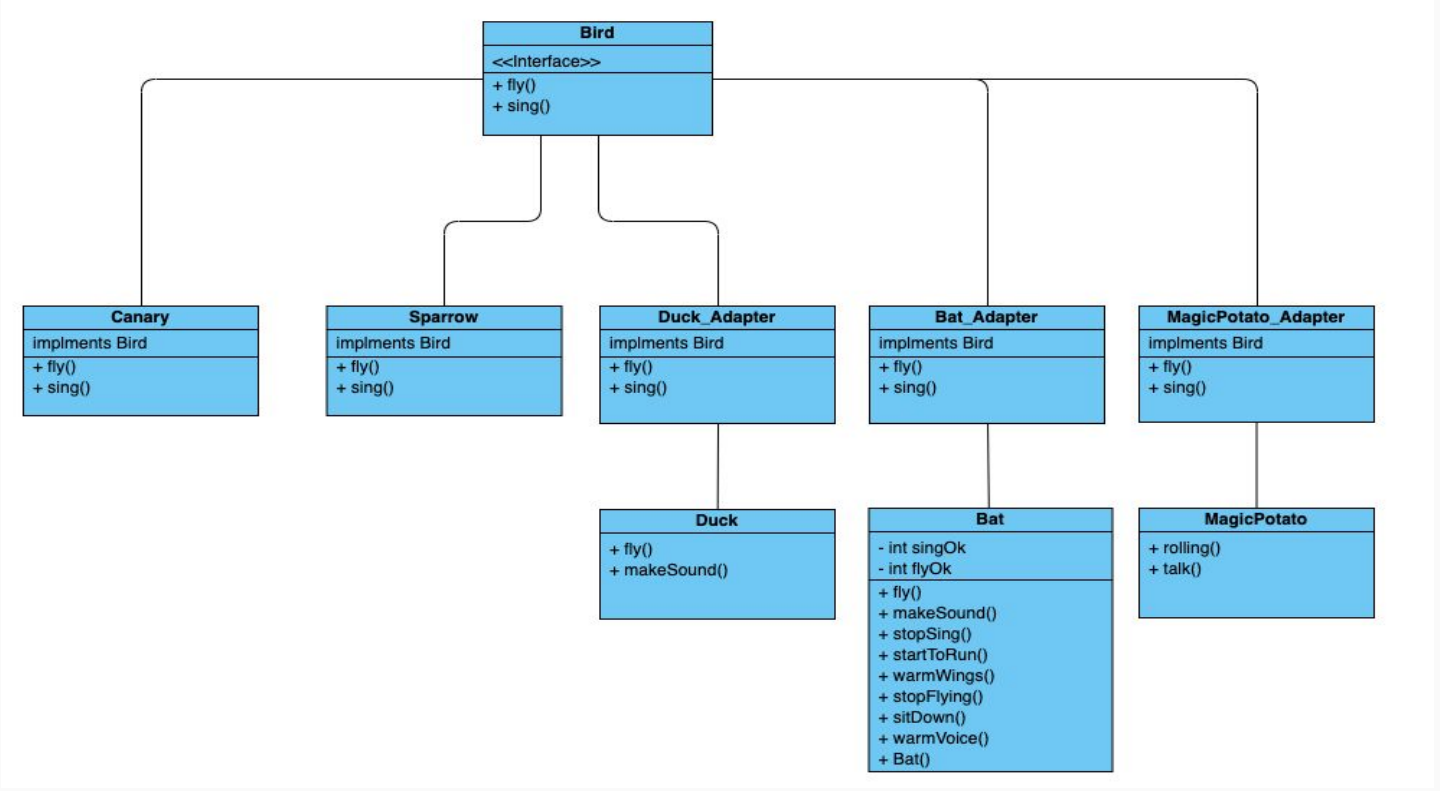


¿Que es el Adapt Pattern?

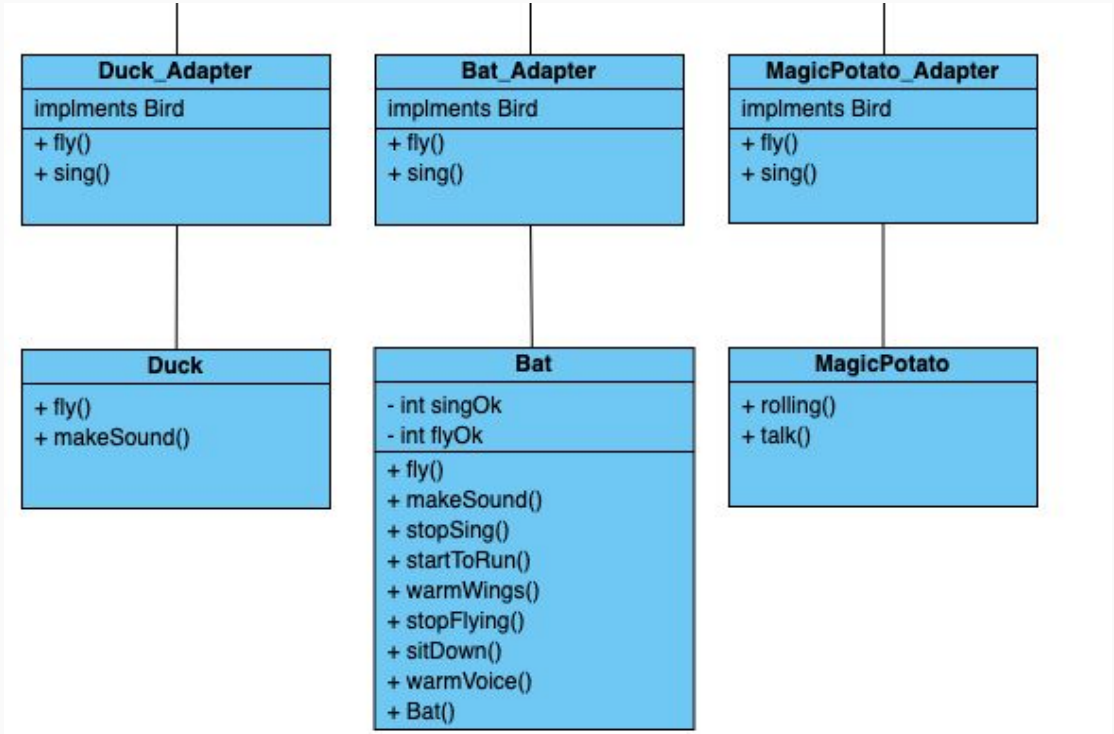
Si queremos enchufar un aparato con un conector eléctrico europeo a una toma de corriente japonesa , necesitaremos un “Adaptador” que simule las características que necesitamos para conectarnos y nos de las “órdenes” que podamos entender para funcionar.



Como lo aplicamos



Como lo aplicamos



Ejemplo con Código

```
public interface Bird {

    public String fly ();
    public String sing();


}
```

Interfaz Bird

Esta es la interface que agrupa todos los objetos de tipo Bird

Con ella podemos acceder a los métodos fly y sing que devuelven un string con el texto de las canciones o lecciones de vuelo de ese Bird

```
public class Canary implements Bird{

    public String fly()
    {
        return " Flying like a canary";
    }
    public String sing()
    {
        return "Melodious PIO PIO";
    }
}
```

Clase Canary que implementa Bird

Esta clase es una tipo Bird y por tanto tiene las dos funciones que hereda de este.

```java
public class MagicPotato {

    public String rolling()
    {
        return "rolling potato";
    }
    public String talk()
    {
        return "Magic potato RULES";
    }
}
```

MagicPotato

Esta es una clase nueva a implantar que no puede ser modificada y no implementa Bird con lo cual no tiene los métodos de esta y los que ya tiene creados no se llaman igual con lo que la implantación actual no los reconocerá.

```java
public class MagicPotato_Adapter implements Bird {

    private MagicPotato potato;

    public MagicPotato_Adapter(MagicPotato potato) {
        this.potato = potato;
    }

    @Override
    public String fly() {
        return potato.rolling();
    }

    @Override
    public String sing() {
        return potato.talk();
    }
}
```

MagicPotato_Adapter implementa Bird

Esta es la clase Adaptador que SI implementa Bird y por tanto puede hacer de “puente” entre nuestra implantación y la nueva clase.

Mediante este adaptador falseamos los métodos de la interfaz llamando a los necesarios en la clase MagicPotato.

```java
public class Bat {

    private int singOk;
    private int flyOk;

    public Bat() { this.singOk = 0;this.flyOk = 0; }

    public String fly(){
        if (flyOk == 3){flyOk = 0;return "Fliying in the Darkness";}
        flyOk = 0;
        return "";
    }

    public void stopSingign(){flyOk ++; }
    public void startToRun(){ flyOk ++;}
    public void warmyourWings(){flyOk ++;}

    public String makeSound(){
        if (singOk == 3){singOk = 0;return "I am the Night";}
        singOk = 0;
        return "";
    }

    public void StopFlying(){ singOk ++; }
    public void sitDown(){ singOk ++; }
    public void warmVoice(){ singOk ++;}
}
```

Bat

Esta es una clase nueva a implantar, al igual que MagicPotato no puede implementar Bird y tiene funciones que no encajan con las de nuestra implantacion.

```java
public class Bat_Adapter implements Bird {

    private Bat batsy;

    public Bat_Adapter(Bat batsy) { this.batsy = batsy;}
    @Override
    public String fly() {
        batsy.stopSingign();
        batsy.startToRun();
        batsy.warmyourWings();
        return batsy.fly();
    }
    @Override
    public String sing() {
        batsy.stopFlying();
        batsy.sitDown();
        batsy.warmVoice();
        return batsy.makeSound();
    }
}
```

Bat_Adapter implementa Bird

Esta es la clase Adaptador que SI implementa Bird y por tanto puede hacer de “puente” entre nuestra implantación y la nueva clase.

Mediante este adaptador falseamos los métodos de la interfaz llamando a los que son necesarios para que tengamos el mismo resultado que en las otras clases.

```
@Test
public void create_a_Melody() {

   Bird isCaptain = new Sparrow();
   Bird canary = new Canary();

   Bird batsy = new Bat_Adapter(new Bat());
   Bird donald = new Duck_Adapter(new Duck());
   Bird potato = new MagicPotato_Adapter(new MagicPotato());

   Melody melody = new Melody();
   Melody melodyExpected = new Melody("I am the Night
Melodious PIO PIO Cuack Cuack Magic potato RULES Chirp
Chirp");

   assertEquals( melody.createSong(new Bird[]{isCaptain,
canary, batsy, donald, potato}),melody.melody);
}
```

Test Create_a_Melody

Test que comprueba que la implantación nueva funciona correctamente al crear una canción.

```
@Test
public void teach_Flylessons(){

    Bird isCaptain = new Sparrow();
    Bird canary = new Canary();

    Bird batsy = new Bat_Adapter(new Bat());
    Bird donald = new Duck_Adapter(new Duck());
    Bird potato = new MagicPotato_Adapter(new MagicPotato());

    FlyTime flylesson = new FlyTime();
    FlyTime flyLessonExpected = new FlyTime("Fliying in the
Darkness  Flying like a canary Flying more or less rolling potato
Flying to steal your Bread ");

    assertEquals(flyLessonExpected.flyLesson ,
flylesson.createFlylesson(new Bird []
{batsy,canary,donald,potato,isCaptain}));
}
```

Test teach_Flylessons

Test que comprueba que la implantación nueva funciona correctamente al crear una leccion de vuelo.

Conclusiones

No intentes reinventar la rueda,los Design Patterns son tus amigos

Q & A

Laura Gonzalez Fernandez

www.lauragonzafer.com

Muchas Gracias

LIFULL CONNECT