# Practicas

- Practicas
  - Practica 2 - Refactoring
    - Ejercicio 2
      - 2.1
        i. El refactoring que usaria para corregirlo es el de Extract Superclass

        ii. El code smell que identifico es el de Duplicate Code

        iii.
```
public abstract class Empleado{
    private String nombre;
    private String apellido;
    private double sueldoBasico;

    public double getSueldoBasico(){
        return this.sueldoBasico;
}

public class EmpleadoTemporario extends Empleado{
    private double horasTrabajadas = 0;
    private int cantidadHijos = 0;

    public double sueldo() {
        return this.getSueldoBasico +
        (this.horasTrabajadas * 500) +
        (this.cantidadHijos * 1000) -
        (this.getSueldoBasico * 0.13);
    }

}
```

```
public class EmpleadoPlanta extends Empleado{
    private int cantidadHijos = 0;

    public double sueldo() {
      return this.getSueldoBasico + (this.cantidadHijos * 2000)
        - (this.getSueldoBasico * 0.13);
    }
}

public class EmpleadoPasante extends Empleado{
     public double sueldo() {
        return this.getSueldoBasico - (this.getSueldoBasico * 0.1
    }
```

iv. Identifico otro code smell, tambien Duplicate Code

v. Esta vez, aplicaria el refactoring  Form Template Method

vi.

```
public abstract class Empleado{
    private String nombre;
    private String apellido;
    private double sueldoBasico;

    public double getSueldoBasico(){
        return this.sueldoBasico;

    public double sueldo(){
        return this.sueldoBasico + this.calcularBonificacion() - this.s

    public abstract double calcularBonificacion();
}

public class EmpleadoTemporario extends Empleado{
    private double horasTrabajadas = 0;
    private int cantidadHijos = 0;
```

```
    public double calcularBonificacion(){
        return this.horasTrabajadas * 500 + this.cantidadHijos * 1000
    }
}

public class EmpleadoPlanta extends Empleado{
    private int cantidadHijos = 0;

    public double calcularBonificacion(){
        return this.cantidadHijos * 1000;
    }
}

public class EmpleadoPasante extends Empleado{
     public double calcularBonificacion(){
        return 0;
    }
}
```

i. Identifico otro code smell, el cual nuevamente es Duplicate Code

ii.Aplico Extract Superclass

iii.

```
public abstract class Empleado{
    private String nombre;
    private String apellido;
    private double sueldoBasico;

    public double getSueldoBasico(){
        return this.sueldoBasico;

    public double sueldo(){
        return this.sueldoBasico + this.calcularBonificacion() - this.su
```

```
    public abstract double calcularBonificacion();
}

public abstract EmpleadoConHijos extends Empleado{
    private int cantidadHijos = 0;

    public int getCantidadHijos(){
        return this.cantidadHijos;
    }

    public double calcularBonificacion(){
    return this.cantidadHijos * 1000 + this.calcularExtras();
    }

    public abstract double calcularExtras();
}


public class EmpleadoTemporario extends EmpleadoConHijos{
    private double horasTrabajadas = 0;

    public double calcularExtras(){
        return this.horasTrabajadas * 500;
    }
}

public class EmpleadoPlanta extends EmpleadoConHijos{

    public double calcularExtas(){
        return 0;
    }
}

public class EmpleadoPasante extends Empleado{
     public double calcularBonificacion(){
        return 0;
```

```
    }
}
```

▼ 2.2

i. Identifico el code smell de Envidia de Atributos

ii. Aplico el Refactoring de Move Method

iii.

```
public class Juego {
    ...
}

public class Jugador {
    public String nombre;
    public String apellido;
    public int puntuacion = 0;

    public void incrementar(){
        this.puntuacion += 100;
    }

    public void decrementar(){
        this.puntuacion -= 100;
    }
```

i. Identifico code smell Lazy Class

ii. Aplico Incline class

iii.

```
public class Jugador {
    public String nombre;
    public String apellido;
    public int puntuacion = 0;
```

```
    public void incrementar(){
        this.puntuacion += 100;
    }

    public void decrementar(){
        this.puntuacion -= 100;
    }
```

▼ 2.3

i. Identifico code smell de Long Method

ii. Aplico el refactoring de Extract Method

iii.

```
public List<Post> ultimosPosts(Usuario user, int cantidad){
    List<Post> postsOtrosUsuarios = new ArrayList<Post>();
    for (Post post : this.posts){
        if(!post.getUsuario().equals(user)){
            postOtrosUsuarios.add(post);
        }
    }
    postOtrosUsuarios = this.ordenarPorFecha(postOtrosUsuarios)

    List<Post> ultimosPosts = new ArrayList<Post>();
    int index = 0;
    Iterator<Post> postIterator = postsOtrosUsuarios.iterator();
    while(postIterator.hasNext() && index < cantidad){
        ultimosPosts.add(postIterator.next());
    }
    return ultimosPosts;
}

public List<Post> ordenarPorFecha(List<Post> lista){
    for(int i = 0; i < lista.size(); i++){
```

```
        int masNuevo = 1;
        for (int j=1+1; j < lista.size(); j++){
            if(lista.get(j).getFecha().isAfter(lista.get(masNuevo).getFec
                masNuevo = j;
            }
        }
        Post unPost = lista.set(i, lista.get(masNuevo));
        lista.set(masNuevo, unPost);
    }
    return lista;
}
```

i. Detecto Code Smell de reinvencion de la rueda

ii. Aplico refactoring Replace Loop with Pipeline

iii.

```
public List<Post> ultimosPosts(Usuario user, int cantidad){

    List<Post> postsOtrosUsuarios = this.posts.stream()
    .filter(post → !post.getUsuario().equals(user))
    .collect(Collectors.toList());

    postOtrosUsuarios = this.ordenarPorFecha(postOtrosUsuarios)

    List<Post> ultimosPosts = new ArrayList<Post>();
    int index = 0;
    Iterator<Post> postIterator = postsOtrosUsuarios.iterator();
    while(postIterator.hasNext() && index < cantidad){
        ultimosPosts.add(postIterator.next());
    }
    return ultimosPosts;
}

public List<Post> ordenarPorFecha(List<Post> lista){
    List<post> listaOrdenada = lista.stream()
```

```
    .sorted((post1, post2) →
        post1.getFecha().compareTo(post2.getFecha())))
    .collect(Collectors.toList());
    return listaOrdenada;
}
```

i. Detecto code smell de Codigo Innecesario

ii. Lo soluciono Eliminandolo

iii.

```
public List<Post> ultimosPosts(Usuario user, int cantidad){

    List<Post> postsOtrosUsuarios = this.posts.stream()
    .filter(post → !post.getUsuario().equals(user))
    .collect(Collectors.toList());

    postOtrosUsuarios = this.ordenarPorFecha(postOtrosUsuarios)
    return postOtrosUsuarios;
}

public List<Post> ordenarPorFecha(List<Post> lista){
    List<post> listaOrdenada = lista.stream()
    .sorted((post1, post2) →
        post1.getFecha().compareTo(post2.getFecha())))
    .collect(Collectors.toList());
    return listaOrdenada;
}
```

▼ 2.4

i. Identifico Code Smell de Envidia de Atributos

ii. Aplico Refactoring de Extract Method

iii.

```
public class Producto{
    private String nombre;
    private double precio;

    public double getPrecio(){
        return this.precio();
    }
}

public class ItemCarrito {
    private Producto producto;
    private int cantidad;

    public double calcularPrecioTotal(){
        return this.cantidad * this.getProducto().getPrecio();
    }
}

public class Carrito{
    private List<ItemCarrito> items;

    public double total() {
        return this.items.stream()
        .mapToDouble(item → item.calcularPrecioTotal())
        .sum();
    }
}
```

▼ 2.5

i. Identifico el Code Smell de Envidia de Atributos

ii. Lo resuelvo con el Refactoring Extract Method

iii.

```
public class Supermercado {
   public void notificarPedido(long nroPedido, Direccion dir) {
     String notificacion = MessageFormat.format(“Estimado cliente
     , new Object[] { nroPedido, dir.getDireccionFormateada() });

     // lo imprimimos en pantalla, podría ser un mail, SMS, etc..
    System.out.println(notificacion);
  }
}

public class Cliente {
}

public class Direccion{
    private String localidad;
    private String calle;
    private String numero;
    private String departamento;

    public String getDireccionFormateada(){
        return
        this.direccion.getLocalidad() + “, ” +
        this.direccion.getCalle() + “, ” +
        this.direccion.getNumero() + “, ” +
        this.direccion.getDepartamento();
    }
```

i. Identifico el Code Smell de Middle Man

ii. La elimino

iii.

```
public class Supermercado {
   public void notificarPedido(long nroPedido, Direccion dir) {
     String notificacion = MessageFormat.format(“Estimado cliente
```

```
    , new Object[] { nroPedido, dir.getDireccionFormateada() });

    // lo imprimimos en pantalla, podría ser un mail, SMS, etc..
    System.out.println(notificacion);
  }
}

public class Direccion{
    private String localidad;
    private String calle;
    private String numero;
    private String departamento;

    public String getDireccionFormateada(){
        return
        this.direccion.getLocalidad() + “, ” +
        this.direccion.getCalle() + “, ” +
        this.direccion.getNumero() + “, ” +
        this.direccion.getDepartamento();
    }
```

▼ 2.6

i. Identifico el Code Smell de Switch Statements

ii. Lo resuelvo aplicando el refactoring de Replace Conditional with Polymorphism

iii.
```
public class Usuario {
    Subscripcion tipoSubscripcion;
    // ...

    public void setTipoSubscripcion(Subscripcion unTipo) {
     this.tipoSubscripcion = unTipo;
    }
}
```

```
public class Pelicula {
    LocalDate fechaEstreno;
    // ...

    public double getCosto() {
     return this.costo;
    }

    public double calcularCargoExtraPorEstreno(){
    // Si la Película se estrenó 30 días antes de la fecha actual,
    return (ChronoUnit.DAYS.between(this.fechaEstreno, Local
    }
}

public abstract class Subscripcion{
    public abstract double calcularCosto(Pelicula peli);
}

public class SubscripcionBasica extends Subscripcion{
    public double calcularCosto(Pelicula peli){
        return peli.getCosto() + peli.calcularCargoExtraPorEstrer
    }
}

public class SubscripcionFamilia extends Subscripcion{
    public double xcalcularCosto(Pelicula peli){
        return peli.getCosto() + peli.calcularCargoExtraPorEstrer
    }
}

public class SubscripcionPlus extends Subscripcion{
    public double calcularCosto(Pelicula peli){
        return peli.getCosto();
    }
}
```

```
public class SubscripcionPremium extends Subscripcion{
    public double calcularCosto(Pelicula peli){
        return peli.getCosto() * 0.75;
    }
}
```

i. Indentifico el Code Smell de Envidia de Atributos

ii. Aplico el refactoring de Move Method

iii.

```
public class Usuario {
    Subscripcion tipoSubscripcion;
    // ...

    public void setTipoSubscripcion(Subscripcion unTipo) {
     this.tipoSubscripcion = unTipo;
    }
}

public class Pelicula {
    LocalDate fechaEstreno;
    // ...

        public double getCosto(){
            return this.costo;
        }

    public double calcularCosto() {
     return this.costo + this.calcularCargoExtraPorEstreno();
    }

    public double calcularCargoExtraPorEstreno(){
    // Si la Película se estrenó 30 días antes de la fecha actual,
    return (ChronoUnit.DAYS.between(this.fechaEstreno, Local
```

```
    }
}

public abstract class Subscripcion{
    public abstract double calcularCostoFinal(Pelicula peli);
}

public class SubscripcionBasica extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.calcularCosto();
    }
}

public class SubscripcionFamilia extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.calcularCosto() * 0.90;
    }
}

public class SubscripcionPlus extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.getCosto();
    }
}

public class SubscripcionPremium extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.getCosto() * 0.75;
    }
}
```

i. Identifico el Code Smell de Comments

ii. Para solucionar esto elimino el comentario debido a que la firma del metodo lo vuelve innecesario

iii.

```
public class Usuario {
    Subscripcion tipoSubscripcion;
    // ...

    public void setTipoSubscripcion(Subscripcion unTipo) {
     this.tipoSubscripcion = unTipo;
    }
}

public class Pelicula {
    LocalDate fechaEstreno;
    // ...

        public double getCosto(){
            return this.costo;
        }

    public double calcularCosto() {
     return this.costo + this.calcularCargoExtraPorEstreno();
    }

    public double calcularCargoExtraPorEstreno(){
    return (ChronoUnit.DAYS.between(this.fechaEstreno, Local
    }
}

public abstract class Subscripcion{
    public abstract double calcularCostoFinal(Pelicula peli);
}

public class SubscripcionBasica extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.calcularCosto();
```

```
    }
}

public class SubscripcionFamilia extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.calcularCosto() * 0.90;
    }
}

public class SubscripcionPlus extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.getCosto();
    }
}

public class SubscripcionPremium extends Subscripcion{
    public double calcularCostoFinal(Pelicula peli){
        return peli.getCosto() * 0.75;
    }
}
```

▼ Ejercicio 3

▼ 1

Identifico, Rompe el encapsultamiento, usa variables temporales innecesariamente y Duplicated Code

Y los refactoring serian: Encapsulate Field, Replace Temp with Query, Extract Method.

▼ 2

```
public class Document {
    private List<String> words;

    public long characterCount() {
```

```
        return this.words
        .stream()
        .mapToLong(w -> w.length())
        .sum();
        }
    public long calculateAvg() {
            return this.characterCount() / this.words.size();
        }
// Resto del código que no importa
}
```

▼ 3

Los errores que identifico, son error en el tipo del metodo calculateAvg, el cual deberia ser double y no checkea que la lista no tenga 0 elementos, ya que en ese caso dividiria por 0. Los errores se mantienen luego de los refactorings. Y corregir esto, no seria tecnicamente un refactoring ya que, estaria agregando funcionalidad en el caso de los checkeos o estaria cambiando el rango de valores que el metodo puedo retornar.

▼ Ejercicio 4

▼ 1.

```
01:    public class Pedido {
02:  private Cliente cliente;
03:  private List<Producto> productos;
04:  private String formaPago;
05:  public Pedido(Cliente cliente, List<Producto> productos, Stri
06:     if (!"efectivo".equals(formaPago)
07:        && !"6 cuotas".equals(formaPago)
08:        && !"12 cuotas".equals(formaPago)) {
09:          throw new Error("Forma de pago incorrecta");
10:    }
11:    this.cliente = cliente;
12:    this.productos = productos;
```

```
13:    this.formaPago = formaPago;
14:   }
15:   public double getCostoTotal() {
16:     double costoProductos = 0;
17:     for (Producto producto : this.productos) {
18:       costoProductos += producto.getPrecio();
19:     }
20:     double extraFormaPago = 0;
21:     if ("efectivo".equals(this.formaPago)) {
22:       extraFormaPago = 0;
23:     } else if ("6 cuotas".equals(this.formaPago)) {
24:       extraFormaPago = costoProductos * 0.2;
25:     } else if ("12 cuotas".equals(this.formaPago)) {
26:       extraFormaPago = costoProductos * 0.5;
27:     }
28:     int añosDesdeFechaAlta = Period.between(this.cliente.getF
29:     // Aplicar descuento del 10% si el cliente tiene más de 5 añ
30:     if (añosDesdeFechaAlta > 5) {
31:       return (costoProductos + extraFormaPago) * 0.9;
32:     }
33:     return costoProductos + extraFormaPago;
34:   }
35: }
36: public class Cliente {
37:   private LocalDate fechaAlta;
38:   public LocalDate getFechaAlta() {
39:     return this.fechaAlta;
40:   }
41: }
42: public class Producto {
43:   private double precio;
44:   public double getPrecio() {
45:     return this.precio;
46:   }
47: }
```

i. Aplico replace Loop With Pipeline

ii.

```
01:    public class Pedido {
02:  private Cliente cliente;
03:  private List<Producto> productos;
04:  private String formaPago;
05:  public Pedido(Cliente cliente, List<Producto> productos, Stri
06:     if (!"efectivo".equals(formaPago)
07:        && !"6 cuotas".equals(formaPago)
08:        && !"12 cuotas".equals(formaPago)) {
09:          throw new Error("Forma de pago incorrecta");
10:    }
11:    this.cliente = cliente;
12:    this.productos = productos;
13:    this.formaPago = formaPago;
14:  }
15:  public double getCostoTotal() {
16:     double costoProductos = this.productos
              .stream()
              .mapToDouble(producto → producto.getCosto())
              .sum();
20:     double extraFormaPago = 0;
21:     if ("efectivo".equals(this.formaPago)) {
22:       extraFormaPago = 0;
23:     } else if ("6 cuotas".equals(this.formaPago)) {
24:       extraFormaPago = costoProductos * 0.2;
25:     } else if ("12 cuotas".equals(this.formaPago)) {
26:       extraFormaPago = costoProductos * 0.5;
27:     }
28:     int añosDesdeFechaAlta = Period.between(this.cliente.getF
29:     // Aplicar descuento del 10% si el cliente tiene más de 5 añ
30:     if (añosDesdeFechaAlta > 5) {
31:       return (costoProductos + extraFormaPago) * 0.9;
32:     }
```

```
33:     return costoProductos + extraFormaPago;
34:   }
35: }
36: public class Cliente {
37:   private LocalDate fechaAlta;
38:   public LocalDate getFechaAlta() {
39:     return this.fechaAlta;
40:   }
41: }
42: public class Producto {
43:   private double precio;
44:   public double getPrecio() {
45:     return this.precio;
46:   }
47: }
```

1. Aplico Replace Conditional with Polymorphism
2. 

```
01:    public class Pedido {
02:  private Cliente cliente;
03:  private List<Producto> productos;
04:  private FormaDePago formaPago;
05:  public Pedido(Cliente cliente, List<Producto> productos, Stri
06:     if (!"efectivo".equals(formaPago)
07:        && !"6 cuotas".equals(formaPago)
08:        && !"12 cuotas".equals(formaPago)) {
09:          throw new Error("Forma de pago incorrecta");
10:    }
11:    this.cliente = cliente;
12:    this.productos = productos;
13:    this.formaPago = formaPago;
14:   }
15:   public double getCostoTotal() {
16:     double costoProductos = this.productos
```

```
            .stream()
            .mapToDouble(producto → producto.getCosto())
            .sum();
20:     double extraFormaPago = this.formaPago.calcularPrecio(co
28:     int añosDesdeFechaAlta = Period.between(this.cliente.getF
29:     // Aplicar descuento del 10% si el cliente tiene más de 5 añ
30:     if (añosDesdeFechaAlta > 5) {
31:      return (costoProductos + extraFormaPago) * 0.9;
32:     }
33:     return costoProductos + extraFormaPago;
34:   }
35: }
36: public class Cliente {
37:   private LocalDate fechaAlta;
38:   public LocalDate getFechaAlta() {
39:     return this.fechaAlta;
40:   }
41: }
42: public class Producto {
43:   private double precio;
44:   public double getPrecio() {
45:     return this.precio;
46:   }
47: }


public abstract class FormaDePago{
   public abstract double calcularPrecio(double costo);
}

public class Efectivo extends FormaDePago{
   public double calcularPrecio(double costo){
      return 0;
   }
}
```

```
public class SeisCuotas extends FormaDePago{
   public double calcularPrecio(double costo){
      return costo * 0.2;
   }
}

public class SeisCuotas extends FormaDePago{
   public double calcularPrecio(double costo){
      return costo * 0.5;
   }
}
```

1. Aplico Extract Method y Move Method
2. 

```
01:    public class Pedido {
02:  private Cliente cliente;
03:  private List<Producto> productos;
04:  private FormaDePago formaPago;
05:  public Pedido(Cliente cliente, List<Producto> productos, Stri
06:     if (!"efectivo".equals(formaPago)
07:        && !"6 cuotas".equals(formaPago)
08:        && !"12 cuotas".equals(formaPago)) {
09:          throw new Error("Forma de pago incorrecta");
10:    }
11:    this.cliente = cliente;
12:    this.productos = productos;
13:    this.formaPago = formaPago;
14:   }
15:   public double getCostoTotal() {
16:     double costoProductos = this.productos
              .stream()
              .mapToDouble(producto → producto.getCosto())
              .sum();
20:     double extraFormaPago = this.formaPago.calcularPrecio(co
```

```
30:     return this.calcularDescuento(costoProductos, extraFormaP
34:   }
35:
// Aplicar descuento del 10% si el cliente tiene más de 5 años de a
          public doble calcularDescuento(double costoP, double ext
                    if(this.cliente.calcularAntiguedad(); > 5){
                       return(costoP + extra) * 0.9;
                    }
                    return costoP + extra;
                }
       }
36: public class Cliente {
37:   private LocalDate fechaAlta;

          public int calcularAntiguedad(){
             return Period.between(this.fechaAlta, LocalDate.now().g
          }
42: public class Producto {
43:   private double precio;
44:   public double getPrecio() {
45:     return this.precio;
46:   }
47: }



public abstract class FormaDePago{
   public abstract double calcularPrecio(double costo);
}

public class Efectivo extends FormaDePago{
   public double calcularPrecio(double costo){
      return 0;
   }
}

public class SeisCuotas extends FormaDePago{
```

```
    public double calcularPrecio(double costo){
        return costo * 0.2;
    }
}

public class SeisCuotas extends FormaDePago{
    public double calcularPrecio(double costo){
        return costo * 0.5;
    }
}
```

1. Aplico