Buenas Prácticas de Programación

Ejercicio Práctico

Se desea crear un programa que realice la venta de perros y hamburguesas. El programa será operado por consola y debe calcular el total de la venta realizada.

Solución Ejercicio Práctico

En el siguiente enlace se encuentra el repositorio con el código original y la refactorización:

Repositorio (Cambiar)

A continuación se muestra la solución del problema:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

int valPerro = 2000, valHamburguesa = 3000, cantPerro = 0, cantHamburguesa = 0,
proSel = 0, cant, selFin = 0, total, totalPer = 0, totalHamb = 0;
boolean initTrans = true;
        Scanner entradaEscaner = new Scanner (System.in);

        System.out.println("Venta de perros y hamburguesas");

        while(initTrans){

            System.out.println("Seleccione producto: ");
            System.out.println("1. Perro \n2. Hamburguesa");
            proSel = Integer.parseInt(entradaEscaner.nextLine());
            System.out.println("Indique cantidad: ");
            cant = Integer.parseInt(entradaEscaner.nextLine());
            if (proSel == 1) {
                cantPerro = cantPerro + cant;
            }else if (proSel == 2){
                cantHamburguesa = cantHamburguesa + cant;
            } else {
                break;
            }
            System.out.println("Desea finalizar o agregar otro producto ?");
            System.out.println("1. Agregar \n2. Finalizar");
            selFin = Integer.parseInt(entradaEscaner.nextLine());
            if(selFin == 1) {
                initTrans = true;
            }else if (selFin == 2) {
                initTrans = false;
```

```
            }else {
                break;
            }

        }

        totalHamb = valHamburguesa*cantHamburguesa;
        totalPer = valPerro*cantPerro;
        total = totalHamb + totalPer;
        System.out.println("Total: "+total);

    }
}
```

Si se corre el programa se puede comprobar que la aplicación se puede ejecutar sin ningún problema y que satisface las necesidades para las que fue desarrollado. Sin embargo el código es poco entendible y desordenado, si alguien deseara realizar un cambio, le tomaría más tiempo, porque principalmente debe comprender que realiza el código. Se realizarán una serie de cambios buscando un mejor orden y legibilidad del código, además de explicar cada uno de los métodos de refactorización que se mostraron.

1. **Rename:**

   Como se vió en la sección de buenas prácticas de programación, es importante que todos los elementos de código que se desarrollen tengan nombres con sentido, que den información de su propósito y que sean pronunciables.
   Por el momento se aplicará el rename a las variables que no tienen nombres muy dicientes. El resultado es el siguiente:

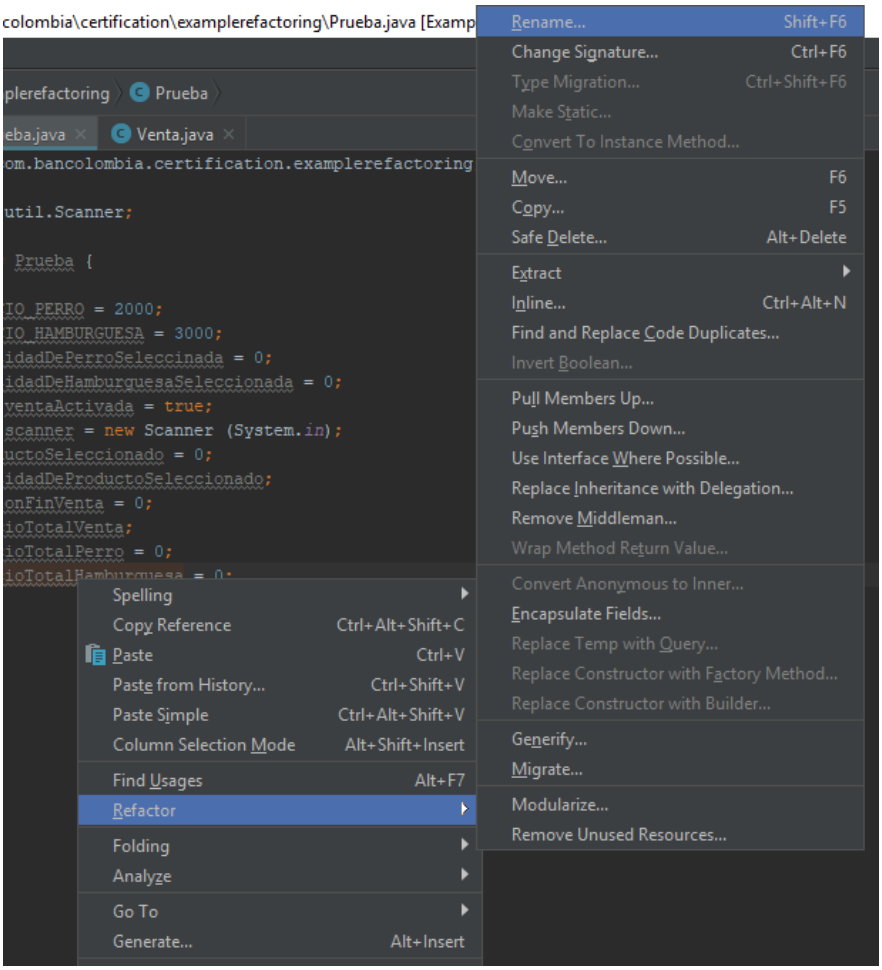
```
int PRECIO_PERRO = 2000;
int PRECIO_HAMBURGUESA = 3000;
int cantidadDePerroSeleccinada = 0;
int cantidadDeHamburguesaSeleccionada = 0;
boolean ventaActivada = true;
Scanner scanner = new Scanner (System.in);
int productoSeleccionado = 0;
int cantidadDeProductoSeleccionado;
int opcionFinVenta = 0;
int precioTotalVenta;
int precioTotalPerro = 0;
int precioTotalHamburguesa = 0;
```

   Además de cambiar los nombres se declararon las variables en una línea de código diferente, esto mejora el orden y la legibilidad del código.

   Para nombrar las variables booleanas se deben leer como si fueran preguntas.

También es importante notar que las constantes se nombraron en mayusculas y separados por guion bajo, este es el estándar que se debe tener para nombrar las constantes.

Hoy en día la mayoría de los IDE tienen herramientas para facilitar el rename, de tal manera que si cambias el nombre de la variable, este busca todas las partes donde fue utilizada y también cambia los nombres. A continuación se muestra el caso particular de IntelliJ:



Para llegar a la opción se debe hacer click derecho sobre la variable, Refactor y luego Rename.

2. Introduce Method:

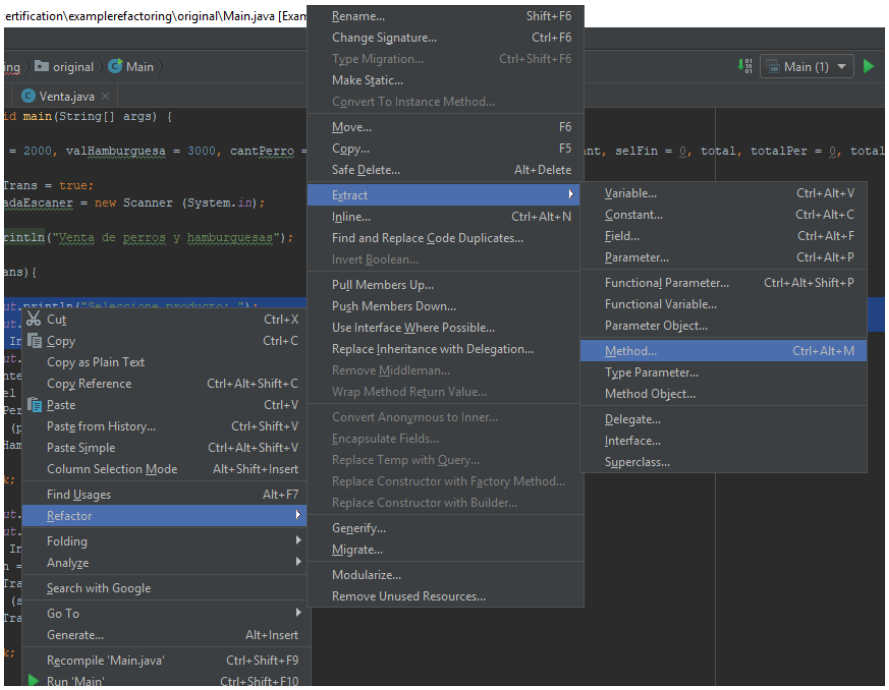
En las secciones de buenas prácticas de programación y principios S.O.L.I.D se habló sobre que las clases y los métodos deberían tener una única responsabilidad y que los métodos entre menos líneas de código se tengan nos darán razón de un buen diseño de la aplicación. Por tanto el siguiente paso de esta refactorización es realizar la separación de responsabilidades que se tiene en el desarrollo, para ellos se hará uso de “Introduce Method”.
De la solución se puede observar que solo se tiene un solo método que implementa el código. La idea, momentáneamente, es crear otros métodos para mejorar el código.

Después de realizar la refactorización, la clase main queda de la siguiente manera:

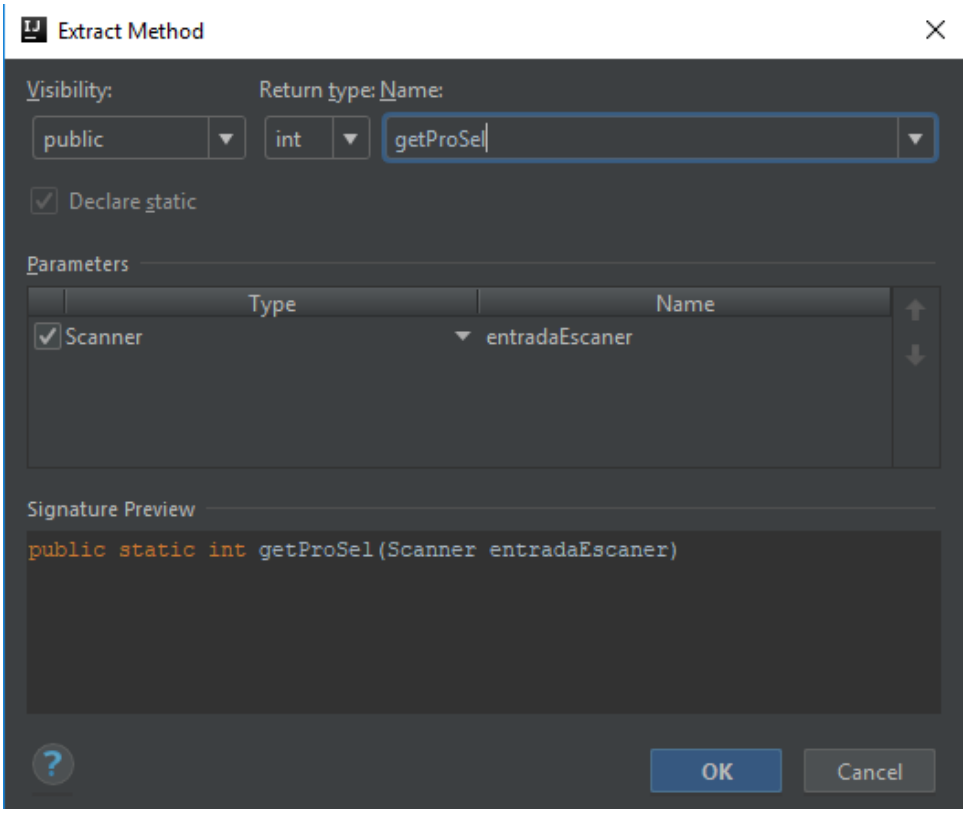
```
public static void main(String[] args) {

    iniciarVenta();
    finalizarVenta();
}
```

Se tiene un código mejor distribuido y mejor empaquetado, ya no es necesario revisar código que no se necesita conocer, se puede ir directamente al método donde se quiere realizar el cambio. El método main esta definido basicamente para lo que es, iniciar la aplicaicón. Sin embargo todavía se tienen métodos que no deberían estar en la clase principal, dicho problema se resolverá mas adelante.

Al igual que en rename, algunos IDE también tienen la opción de crear métodos a partir de porciones de código. A continuación se muestra el ejemplo con IntelliJ:



Para realizar la creación de métodos se debe seleccionar la porción de código que define la función, posteriormente click derecho sobre el área seleccionada, luego se selecciona la opción refactor y finalmente la opción extract method. Finalmente se abre el siguiente cuadro de dialogo que sirve como asistente para la creación del método.



3. **Move Class:**

Como se mencionó anteriormente, la clase principal solamente debería lanzar la aplicación, es por esto que debemos volver a refactorizar e implementar los métodos en sus respectivas clases.

Además de crear las clases es necesario dividir el código por capas para seguir separando las funciones de cada segmento de código, por eso se crearon las capas de utils y domain, dónde en utils se almacenan métodos transversales a toda la aplicación, mientras que en domain se debe almacenar lo referente a la lógica del negocio.

Después de realizar la refactorización tenemos las siguientes clases para la solución del problema:

```
public class Prueba {

    public static void main(String[] args) {

        Venta.iniciarVenta();
        Venta.finalizarVenta();

    }

}

public class Venta {

    public static int cantidadDePerroSeleccinada = 0;
    public static int cantidadDeHamburguesaSeleccionada = 0;

    public static void iniciarVenta() {

        Scanner scanner = new Scanner (System.in);
        boolean ventaActivada = true;
        int productoSeleccionado;
        int cantidadDeProductoSeleccionado;
        int opcionFinVenta;
        System.out.println("Venta de perros y hamburguesas");

        while(ventaActivada){

            productoSeleccionado =
Consola.obtenerProductoSeleccionado(scanner);
            cantidadDeProductoSeleccionado =
Consola.obtenerCantidadDeProductoSeleccionado(scanner);
            aumentarProducto(productoSeleccionado,
cantidadDeProductoSeleccionado);
            opcionFinVenta = Consola.obtenerOpcionFinVenta(scanner);

            ventaActivada = ventaEstaActivada(opcionFinVenta);
        }
    }

    public static void aumentarProducto(int productoSeleccionado, int
cantidadDeProductoSeleccionado) {
        if (productoSeleccionado == 1) {
            cantidadDePerroSeleccinada = cantidadDePerroSeleccinada +
cantidadDeProductoSeleccionado;
        }else if(productoSeleccionado == 2){
            cantidadDeHamburguesaSeleccionada =
cantidadDeHamburguesaSeleccionada + cantidadDeProductoSeleccionado;
        }
    }
```

```
public static boolean ventaEstaActivada(int opcionFinVenta) {
        boolean ventaActivada;
        if(opcionFinVenta == 1) {
            ventaActivada = true;
        }else if(opcionFinVenta == 2) {
            ventaActivada = false;
        }else {
            ventaActivada = false;
        }
        return ventaActivada;
    }

    public static void finalizarVenta() {
        int precioTotalHamburguesa;
        int precioTotalPerro;
        int precioTotalVenta;
        int PRECIO_PERRO = 2000;
        int PRECIO_HAMBURGUESA = 3000;
        precioTotalHamburguesa = PRECIO_HAMBURGUESA *
cantidadDeHamburguesaSeleccionada;
        precioTotalPerro = PRECIO_PERRO * cantidadDePerroSeleccinada;
        precioTotalVenta = precioTotalHamburguesa + precioTotalPerro;
        System.out.println("Total: "+ precioTotalVenta);
    }
}

public class Consola {

    public static int obtenerOpcionFinVenta(Scanner scanner) {
        int opcionFinVenta;
        System.out.println("Desea finalizar o agregar otro producto ?");
        System.out.println("1. Agregar \n2. Finalizar");
        opcionFinVenta = Integer.parseInt(scanner.nextLine());
        return opcionFinVenta;
    }

    public static int obtenerCantidadDeProductoSeleccionado(Scanner
scanner) {
        int cantidadDeProductoSeleccionado;
        System.out.println("Indique cantidad: ");
        cantidadDeProductoSeleccionado =
Integer.parseInt(scanner.nextLine());
        return cantidadDeProductoSeleccionado;
    }

    public static int obtenerProductoSeleccionado(Scanner scanner) {
        int productoSeleccionado;
        System.out.println("Seleccione producto: ");
        System.out.println("1. Perro \n2. Hamburguesa");
        productoSeleccionado = Integer.parseInt(scanner.nextLine());
        return productoSeleccionado;
    }

}
```

4. **Inline:**

Para realizar un ejemplo de Inline se va a modificar el método de ventaEstaActivada, actualmente, se tiene la siguiente función:

```
public static boolean ventaEstaActivada(int opcionFinVenta) {
        boolean ventaActivada;
        if(opcionFinVenta == 1) {
            ventaActivada = true;
        }else if(opcionFinVenta == 2) {
            ventaActivada = false;
        }else {
            ventaActivada = false;
        }
        return ventaActivada;
    }
```

Dicha función se podría cambiar de la siguiente manera, para tener todo en una sola línea:

```
public static boolean ventaEstaActivada(int opcionFinVenta) {
    return opcionFinVenta == 1;
}
```

Recuerdde que si la aplicación cambia su funcionamiento es porque se esta refactorizando mal.

Se puede seguir realizando refactorización del código pero en este documento solo se presenta un ejemplo por cada caso.