AEROPUERTOS JUAN

Ametz Segovia

1.Rename

```
public class Conversor {
    2 usages
    public float conv (float c) {
        float x = c * 166.386f;
        return x;
    }
```

El conversor normal multiplica una variable por un número específico.

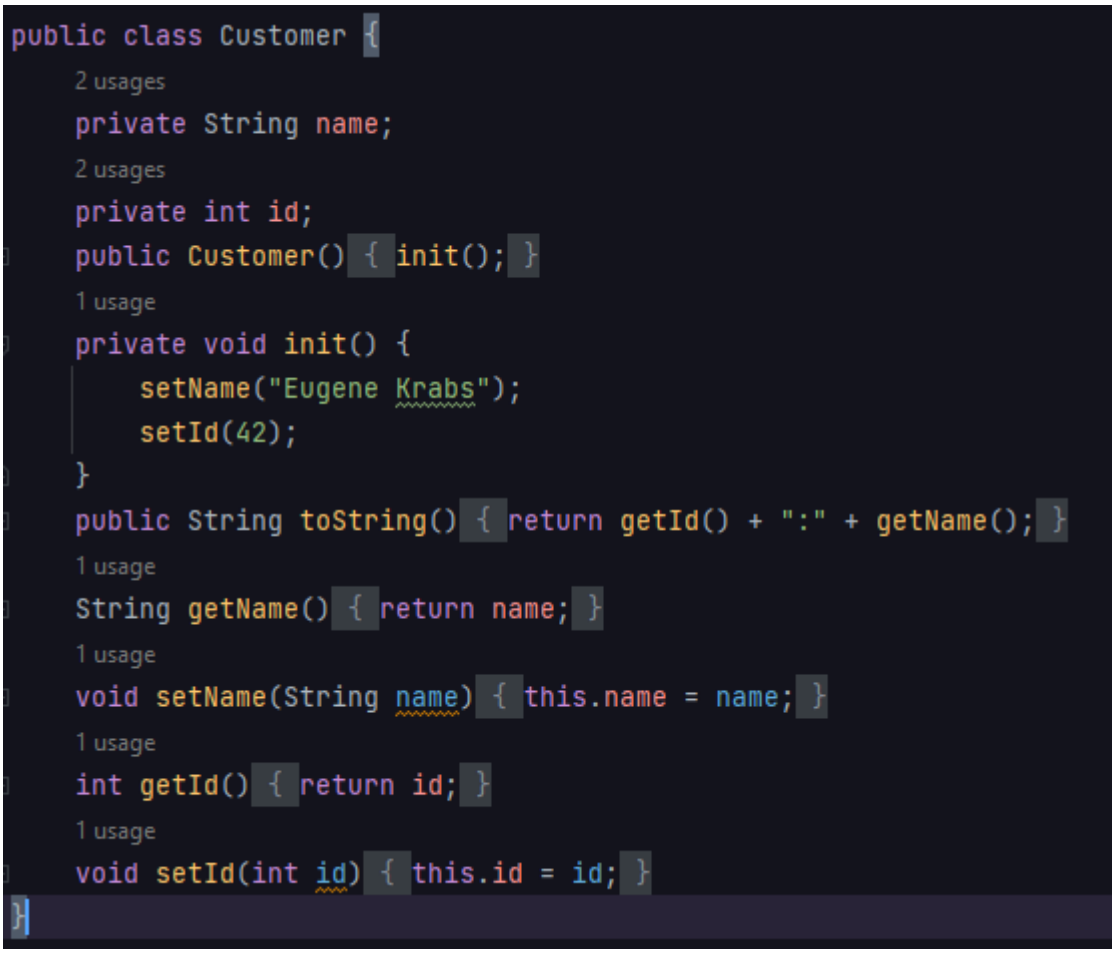
```
public class Conversor {
    1 usage
    private static final float EUROS_PESETAS_CHANGE_RATE = 166.386f;

    public float eurosToPesetas (float euros) {
        float pesetas = euros * EUROS_PESETAS_CHANGE_RATE;
        return pesetas;
    }
}
```

Al refactorizar tienes pesetas que son euros multiplicado por el porcentaje de diferencia. Al porcentaje estar en una variable es mucho más sencillo cambiarlo a futuro.

2.Encapsulate

```
public class Customer {
    2 usages
    String name;
    2 usages
    int id;
    public Customer() { init(); }
    1 usage
    public void init() {
        name = "Eugene Krabs";
        id = 42;
    }
    public String toString() { return id + ":" + name; }
}
```

Por defecto si no especificas en la variable son públicos entonces no es necesario los get y set.



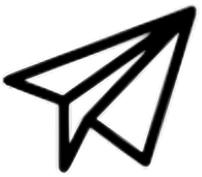
```java
public class Customer {
    2 usages
    private String name;
    2 usages
    private int id;
    public Customer() { init(); }
    1 usage
    private void init() {
        setName("Eugene Krabs");
        setId(42);
    }
    public String toString() { return getId() + ":" + getName(); }
    1 usage
    String getName() { return name; }
    1 usage
    void setName(String name) { this.name = name; }
    1 usage
    int getId() { return id; }
    1 usage
    void setId(int id) { this.id = id; }
}
```

Al poner las variables name y id en privado se tienen que crear los get y set. El toString al no tener acceso directo a las variables tiene que utilizar los get.

3.Extractmethod

```java
public class UrlNormalizer {
    1 usage
    public String normalize(String title) {
        String url = "";
        // First we trim whitespaces
        url = title.trim();
        // Remove special chars
        String specialRemoved = "";
        for (int i = 0; i < url.length(); i++) {
            if (url.charAt(i) != ',' && url.charAt(i) != ':'
                    && url.charAt(i) != '.' && url.charAt(i) != '?') {
                specialRemoved += url.charAt(i);
            }
        }
        url = specialRemoved;
        // Replace white spaces with hyphens
        String spacesReplaced = "";
        for (int i = 0; i < url.length(); i++) {
            if (url.charAt(i) == ' ') {
                spacesReplaced += "-";
            } else {
                spacesReplaced += url.charAt(i);
            }
        }
        url = spacesReplaced;
        // lowercase everything
        url = url.toLowerCase();
        return url;
    }
}
```

El código elimina caracteres especiales, espacios y pone todo en minuscula.



```java
public class UrlNormalizer {
    public String normalize(String title) {
        String url = trimSpaces(title);
        url = removeSpecialChars(url);
        url = replaceSpaces(url);
        url = url.toLowerCase();
        return url;
    }
    1 usage
    private String replaceSpaces(String url) {
        String spacesReplaced = "";
        for (int i = 0; i < url.length(); i++) {
            if (url.charAt(i) == ' ') {
                spacesReplaced += "-";
            } else {
                spacesReplaced += url.charAt(i);
            }
        }
        url = spacesReplaced;
        return url;
    }
    1 usage
    private String removeSpecialChars(String url) {
        String specialRemoved = "";
        for (int i = 0; i < url.length(); i++) {
            if (url.charAt(i) != ',' && url.charAt(i) != ':'
                    && url.charAt(i) != '.' && url.charAt(i) != '?') {
                specialRemoved += url.charAt(i);
            }
        }
        url = specialRemoved;
        return url;
    }
    1 usage
    private String trimSpaces(String title) {
        String url = "";
        url = title.trim();
        return url;
    }
}
```

Al refactorizar se tiene un método para cada cosa.

4.Parameterobject

```java
public class Order {
    3 usages
    private Hashtable<String, Float> items = new Hashtable<~>();
    3 usages
    public void addItem(Integer productID, String description, Integer quantity, Float price, Float discount) {
        items.put(productID + ": " + description, (quantity * price) - (quantity * price * discount));
    }
    1 usage
    public float calculateTotal() {
        float total = 0;
        Enumeration<String> keys = items.keys();

        while (keys.hasMoreElements()) {
            total = total + items.get(keys.nextElement());
        }
        return total;
    }
}
```

Al añadir al hashmap hace calculos.

```java
public class Order {
    3 usages
    private Hashtable<String, Float> items = new Hashtable<~>();

    public void addItem (OrderItem orderItem) {
        items.put(orderItem.getProductID() + ": " + orderItem.getDescription(), orderItem.totalItem());
    }

    public float calculateTotal () {
        float total = 0;
        Enumeration<String> keys = items.keys();

        while(keys.hasMoreElements()) {
            total  = total + items.get(keys.nextElement());
        }

        return total;
    }
}
```

Refactorizado llama a métodos que hay en orderItem.

```java
public class OrderItem {
    2 usages
    private Integer productID;
    2 usages
    private String description;
    4 usages
    private Integer quantity;
    4 usages
    private Float price;
    3 usages
    private Float discount;
    public OrderItem(Integer productID, String description, Integer quantity, Float price, Float discount)
        this.setProductID(productID);
        this.setDescription(description);
        this.setQuantity(quantity);
        this.setPrice(price);
        this.setDiscount(discount);
    }
    1 usage
    public float totalItem () { return (quantity*price) - (quantity*price*discount); }
    1 usage
    public Integer getProductID() { return productID; }
    1 usage
    public void setProductID(Integer productID) { this.productID = productID; }
    1 usage
    public String getDescription() { return description; }
    1 usage
    public void setDescription(String description) { this.description = description; }
    public Integer getQuantity() { return quantity; }
    1 usage
    public void setQuantity(Integer quantity) { this.quantity = quantity; }
    public Float getPrice() { return price; }
    public void setPrice(Float price) { this.price = price; }
    public Float getDiscount() { return discount; }
    public void setDiscount(Float discount) { this.discount = discount; }
}
```

Tiene métodos para los cálculos. Tiene su constructor con get y sets.

5.Splittemporaryvariable

```java
public class Invoice {
    1 usage
    public float totalPrice (float price, float vat, float discount) {
        float temp = 0;
        temp = (vat * price) / 100;
        System.out.println("Applied vat: " + temp);
        temp = price + temp;
        System.out.println("Total with vat: " + temp);
        return temp - discount;
    }
}
```

Crea temporal para guardar datos.

```java
public class Invoice {

    public float totalPrice (float price, float vat, float discount) {
        float appliedVat = (vat * 100) / price;
        System.out.println("Applied vat: " + appliedVat);
        float priceWithVat = price + appliedVat;
        System.out.println("Total: " + priceWithVat);
        return priceWithVat - discount;
    }
```

Elimina temporal y hace los cálculos directamente. Guardando el resultado en variables aparte para tener los datos de cada cálculo.

6.replaceconditionalwithpolymorphism

```
public class Vehicle {

    1 usage
    private static final int CAR = 0;
    1 usage
    private static final int BIKE = 1;
    1 usage
    private static final int PLANE = 2;
    2 usages
    private int vehicleType;
    3 usages
    private int speed;
    3 usages
    private int acceleration;

    public Vehicle(int vehicleType, int speed, int acceleration) {
        this.vehicleType = vehicleType;
        this.speed = speed;
        this.acceleration = acceleration;
    }

    public int move () {
        int result = 0;
        switch (vehicleType) {
            case CAR:
                result = speed * acceleration * 5;
                break;
            case BIKE:
                result = speed * 10;
                break;
            case PLANE:
                result = acceleration * 2;
                break;
        }
        return result;
    }
}
```

Tiene un menú y depende cual se elija calcula de un modo u otro.

```java
public abstract class Vehicle {
    1 usage
    protected int vehicleType;
    protected int speed;
    protected int acceleration;

    public Vehicle(int vehicleType, int speed, int acceleration) {
        this.vehicleType = vehicleType;
        this.speed = speed;
        this.acceleration = acceleration;
    }

     3 implementations
     public abstract int move ();
}
```

Crea clases con los tipos de vehículo.

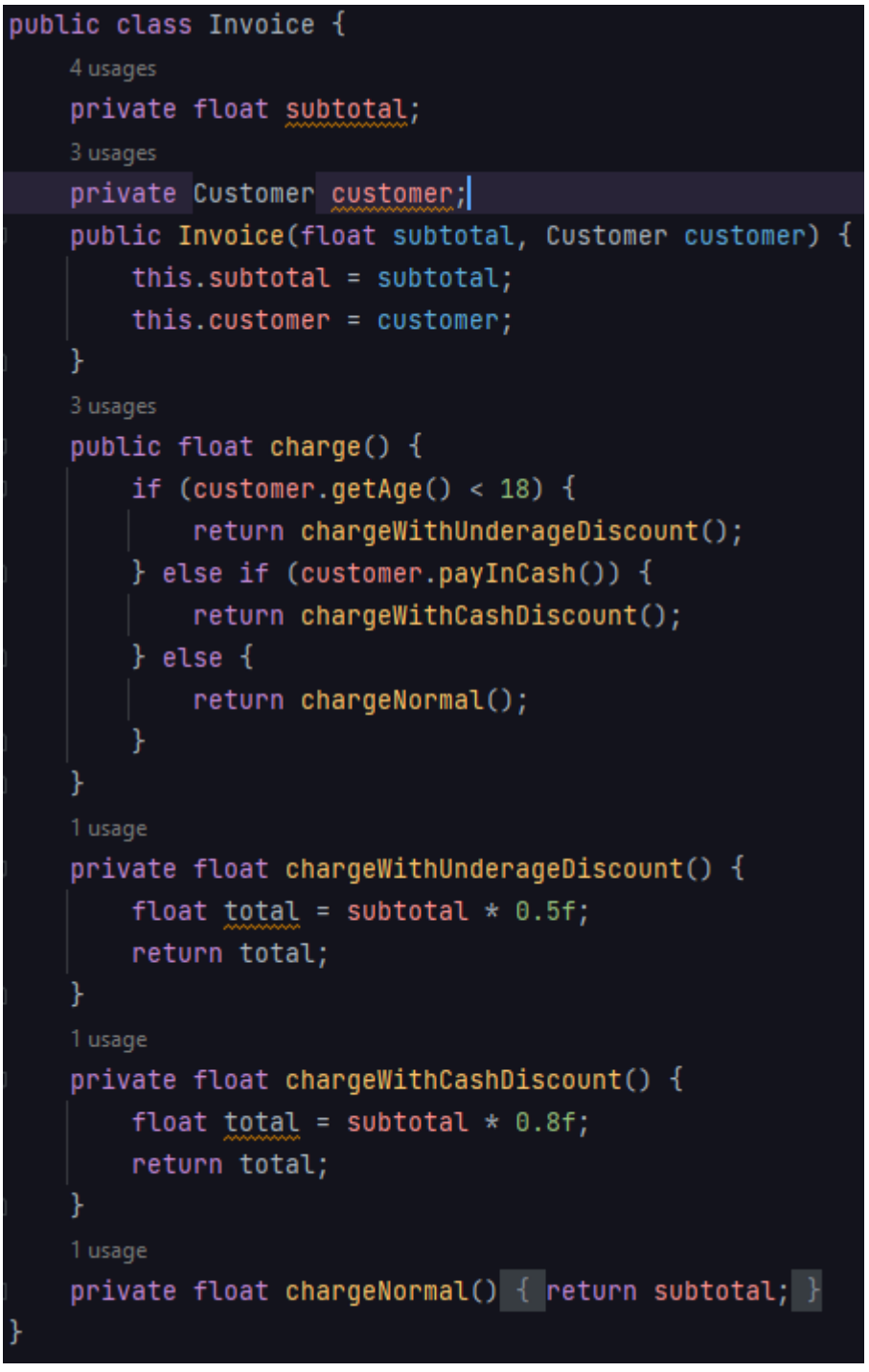
```java
public class Bike extends Vehicle {
    1 usage
    public Bike(int vehicleType, int speed, int acceleration) {
        super(vehicleType, speed, acceleration);
    }

     @Override
     public int move () {
            return speed * 10;
    }
}
```

Cada tipo de vehículo reemplaza el método de move.

7.parametrizedmethod

```
public class Invoice {
    2 usages
    private float subtotal;
    3 usages
    private Customer customer;
    public Invoice(float subtotal, Customer customer) {
        this.subtotal = subtotal;
        this.customer = customer;
    }
    1 usage
    public float charge() {

        if (customer.getAge() < 18) {
            return charge( discount: 0.5f);
        } else if (customer.payInCash()) {
            return charge( discount: 0.8f);
        } else {
            return charge();
        }
    }
    2 usages
    public float charge (float discount) { return subtotal * discount; }
}
```

En el if depende de la condición llama a charge con un valor distinto como parámetro.



```
public class Invoice {
    4 usages
    private float subtotal;
    3 usages
    private Customer customer;
    public Invoice(float subtotal, Customer customer) {
        this.subtotal = subtotal;
        this.customer = customer;
    }
    3 usages
    public float charge() {
        if (customer.getAge() < 18) {
            return chargeWithUnderageDiscount();
        } else if (customer.payInCash()) {
            return chargeWithCashDiscount();
        } else {
            return chargeNormal();
        }
    }
    1 usage
    private float chargeWithUnderageDiscount() {
        float total = subtotal * 0.5f;
        return total;
    }
    1 usage
    private float chargeWithCashDiscount() {
        float total = subtotal * 0.8f;
        return total;
    }
    1 usage
    private float chargeNormal() { return subtotal; }
}
```

Crea distintos charge para cada condición.

8.encapsultacollection

```java
public class Team {
    2 usages
    private String name;
    2 usages
    private Date creation;
    4 usages
    private ArrayList<Player> players = new ArrayList<~>();
    public Team(String name, Date creation) {
        this.name = name;
        this.creation = creation;
    }
    public String getName() { return name; }
    public Date getCreation() { return creation; }
    public Player getPlayer (int index) { return players.get(index); }
    public void addPlayer (Player player) { players.add(player); }
    public void  removePlayer (int index) { players.remove(index); }
    public int totalPlayers() { return players.size(); }
}
```

Añade addPlayer y removePlayer.

9.replacearraywithobject

```java
public class Airplane {

    2 usages
    private String model;
    4 usages
    private String pilotData[] = new String[3];

    1 usage
    public Airplane(String model) { this.model = model; }

    1 usage
    public void initPilot(String name, String license, int flightHours) {
        pilotData[0] = name;
        pilotData[1] = license;
        pilotData[2] = Integer.toString(flightHours);
    }

    @Override
    public String toString() { return "Airplane [model=" + model + ", pilot=" + pilotData[0] + "]"; }
}
```

Tiene un array donde guarda información del piloto.

```java
public class Pilot {
    2 usages
    private String name;
    1 usage
    private String license;
    1 usage
    private int flightHours;

    1 usage
    public Pilot (String name, String license, int flightHours) {
        this.name = name;
        this.license = license;
        this.flightHours = flightHours;
    }

    public String toString() { return name; }
}
```

Reemplaza el array por una clase piloto con la misma información.

10.pullup

```
public class Vehicle {
    protected String name;
}
```

Tiene nombre para diferenciar el vehículo.

```
public class Vehicle {
    protected String name;
    protected String plate;
    public void start() {
    }
}
```

Añade matrícula para que si se repite el nombre se pueda diferenciar los vehículos.

11.pushdown

```
public class Vehicle {
    protected String name;
    protected String plate;
    protected Insurance insurance;
    1 override
    public void start() {
    }
}
```

El método está en vehiculo.

```
public class Vehicle {
    protected String name;
}
```

Quita el método de vehiculo y lo pone en cada clase para que tenga más sentido y sea intuitivo.

```
public class MotorBike extends Vehicle {
    private String type;
    protected String plate;
    protected Insurance insurance;

    public void start() {
    }
}
```