ENTORNOS DE DESARROLLO

Jon urrutia

[FECHA]

[NOMBRE DE LA COMPAÑÍA]
[Dirección de la compañía]

Ejercicio 1

```java
package io.pello.refactorings.rename.refactored;

/**
* Refactored version
*/
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;
       }
}
```

En estos dos la diferencia es que en el de arriba el numero esta guardado en una variable y en la de abajo no. Esto es más útil ya que sería más fácil cambiarle el valor

```java
package io.pello.refactorings.rename;

/**
 * Rename variables and methods
 * This class is intended to be used as a refactoring playground
 * @author PELLO_ALTADILL
 */
public class Conversor {
    2 usages
    public float conv (float c) {
        float x = c * 166.386f;
        return x;
       }
}
```

Ejercicio2

```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; }
```

En estos dos la variable name en uno esta público y en otro privado, a parte que en uno están los getters y setters y en otro no. A parte a la hora de cambiar los valores en uno utiliza los setters y en otro no

```java
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;
    }
}
```

Ejercicio 3

```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;
    }
```

En este ejemplo vemos que en el refactorizad tenemos tres métodos diferentes. En cambio, en el otro tenemos todo en un método desordenado.

```java
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;
}
```

Ejercicio 4

```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;
    }
}
```

En el constructor del refactorizado simplemente pone una variable tipo orderItem y en el otro pone todas las variables de la clase por parámetro.

```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;
    }
}
```

Ejercicio5

```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;
    }
}
```

La diferencia es que en el de arriba lo hace todo con una variable y en el de abajo se reparte en dos

```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;
    }

    /*
     * Another Step
    public float totalPrice (float price, float vat, float discount) {
        return price + appliedVat(price, vat) - discount;
    }

    private float appliedVat (float price, float vat) {
        return (vat * price) / 100;
    }*/
}
```

Ejercicio6

```java
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;
        }
}
```

En la clase de vehículo tiene un método muve que tiene un case para cada situación. Sin envargo en las otras clases cada una tiene su método move con su contenido

```java
package io.pello.refactorings.replaceconditionalwithpolymorphism.refactored.copy;

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;
    }

}
```

Ejercicio 7

```java
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;
        }


}
```

El refactorizado tiene solo un método de carga que le puedes pasar el valor que quieras por parámetro y en el no refactorizado tiene tres métodos diferentes de carga en el que no puedes pasarle nada por parametro

```java
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;
    }
}
```

Ejercicio 8

```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(); }
}
```

La diferencia es que el refactorizado tiene dos métodos más el addplayer y el remove player

```java
public class Team {
    2 usages
    private String name;
    2 usages
    private Date creation;
    2 usages
    private ArrayList<Player> players = new ArrayList<~>();

    1 usage
    public Team(String name, Date creation) {
        this.name = name;
        this.creation = creation;
    }

    public String getName() {
        return name;
    }

    public Date getCreation() {
        return creation;
    }

    public ArrayList<Player> getPlayers() {
        return players;
    }

    1 usage
    public int totalPlayers() {
        return players.size();
    }
}
```

Ejercicio 9

```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] + "]";
    }
}
```

La diferencia es que en el refactorizado tiene un objeto y el otro tiene un array. La diferencia es en el constructor

```java
public class Airplane {

    2 usages
    private String model;
    2 usages
    private Pilot pilot;

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

    public void initPilot(String name, String license, int flightHours) {
        pilot = new Pilot(name, license, flightHours);
    }

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

Ejercicio 10

```java
public class MotorBike extends Vehicle {
    private String plate;
    private String helmet;

    public void start() {
    }
}
```

En una está la variable plate que es para poder diferenciar el vehiculo y en el otro no esta

```java
public class MotorBike extends Vehicle {
    private String helmet;
}
```

Ejercicio 11

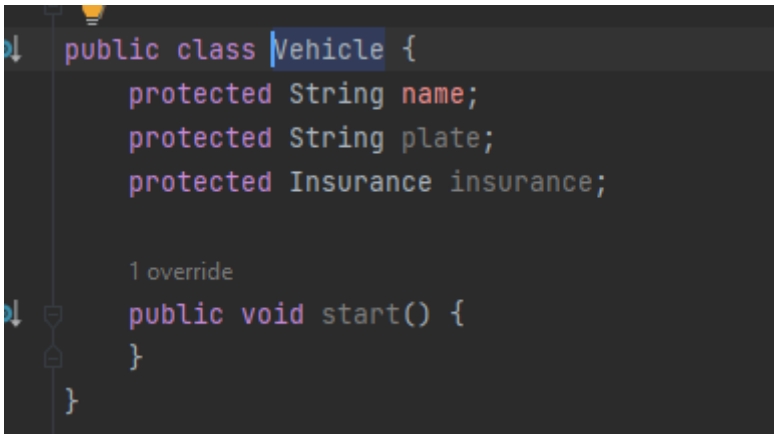
```java
public class Car extends Vehicle {
    private String trunk;
    1 usage
    private boolean isTrunkOpened;
    protected String plate;
    protected Insurance insurance;


    public boolean isTrunkOpen() { return isTrunkOpened; }


    public void start() {
    }

}
```

En el refactorizado está el insurance en cada vehículo y en el otro está en la clase vehículo


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