Examen: Refactoriza, Documenta y crea repositorio remoto.(25/Abril/2016)

Fecha entrega: 24/04/2016

Autores: Isabel María Gómez Palomeque

Indice

INTRODUCCIÓN 3

1.1Refactorización:....3
1.1.1En la clase TestCCuenta....3
1.1.1.1TestCCuenta.java: Extrae los métodos recogerOpcion, mostrarMenu, ingresar, retirar....3
1.1.1.2TestCCuenta.java: Convierte dato en campo....9
1.1.1.3TestCCuenta.java: Incorpora saldoActual en la misma línea....10
1.1.1.4 TestCCuenta.java: Renombra la variable local cuenta1 con tu cuenta en el sistema (d13macom, por ejemplo)....12
1.1.2En la clase CCuenta.java:....12
1.1.2.1CCuenta.java: Cambia el nombre del método "estado" por "getCuenta"....12
1.1.2.2 CCuenta.java: Encapsula los campos y usa los métodos generados.....13
1.2Documentación:....17
1.2.1Deberás documentar todo el proyecto:....17
1.2.2Inserta todos los comentarios Javadoc posibles....17
1.2.2.1Comentario de varias líneas:....17
1.2.2.2Comentario de una línea:....17
1.2.2.3Comentario de javadoc:....17
1.2.3Introduce como autor tu nombre y apellidos....17
1.2.4Genera y entrega la documentación Javadoc para todo el proyecto....18
1.3Repositorio con git:....18

1 INTRODUCCIÓN

Examen: Refactoriza, Documenta y crea repositorio remoto. (25/Abril/2016)

Para realizar este examen deberás crear en Eclipse un proyecto con un paquete y dos clases: TestCCuenta y CCuenta.

1.1 Refactorización:

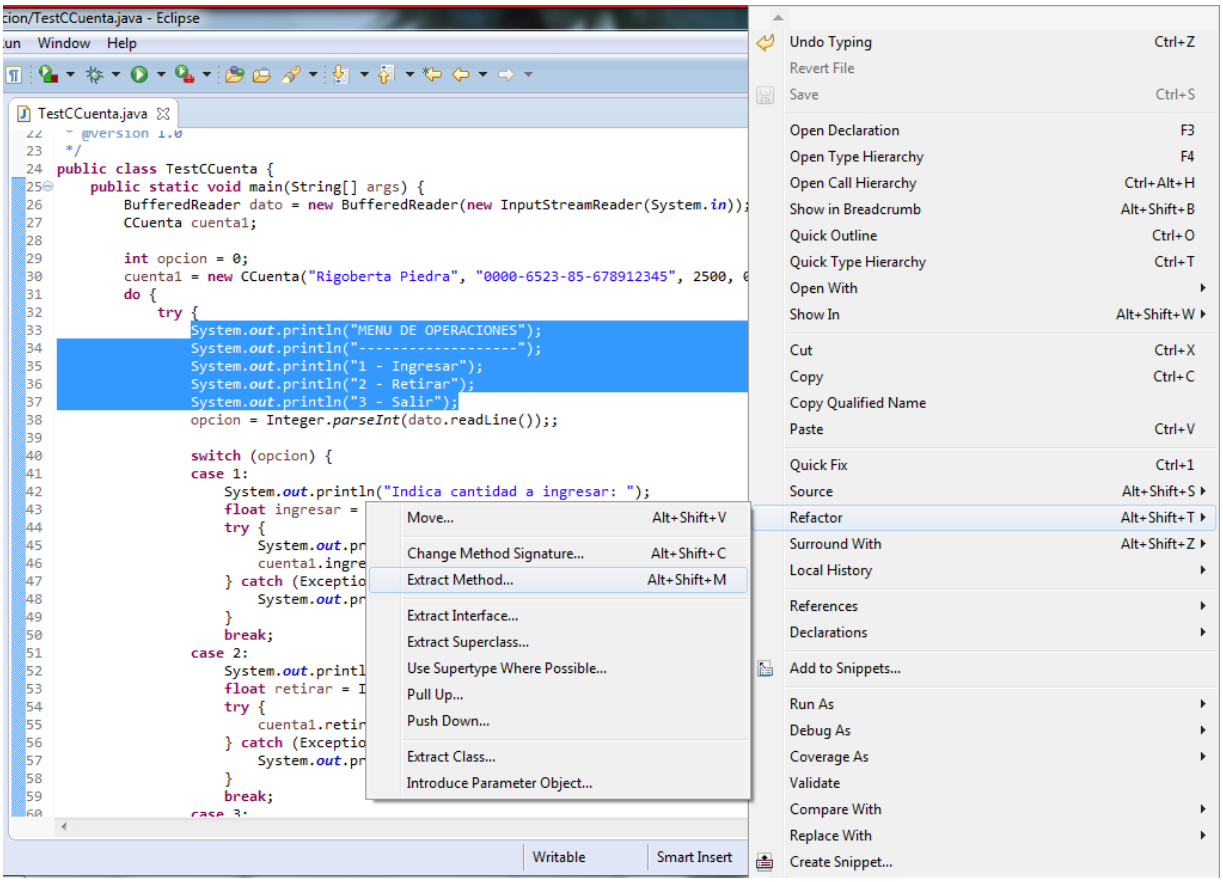
Deberás refactorizar ambas clases y entregar, aparte del resultado, un documento "ApellidosNombreRefactorizacion.pdf" con pantallazos del antes y después de cada uno de los pasos debidamente identificados, incluyendo el menú utilizado para ello.

Realiza las siguientes actividades:

1.1.1 En la clase TestCCuenta

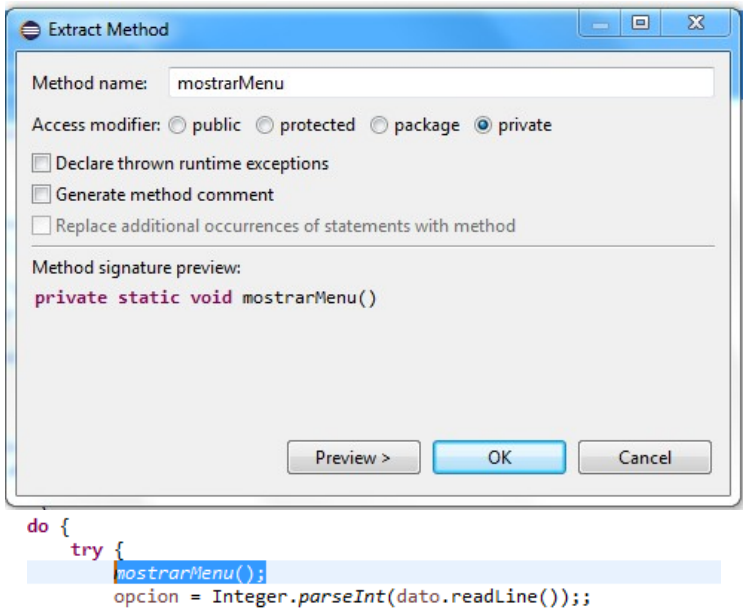
1.1.1.1 TestCCuenta.java: Extrae los métodos recogerOpcion, mostrarMenu, ingresar, retirar

Seleccionamos el trozo de código que queramos usar para crear el método, pulsamos botón derecho sobre el código código y clicamos “Refactor”>>”Extract Method”. También podríamos pulsar Alt + Shift + M.



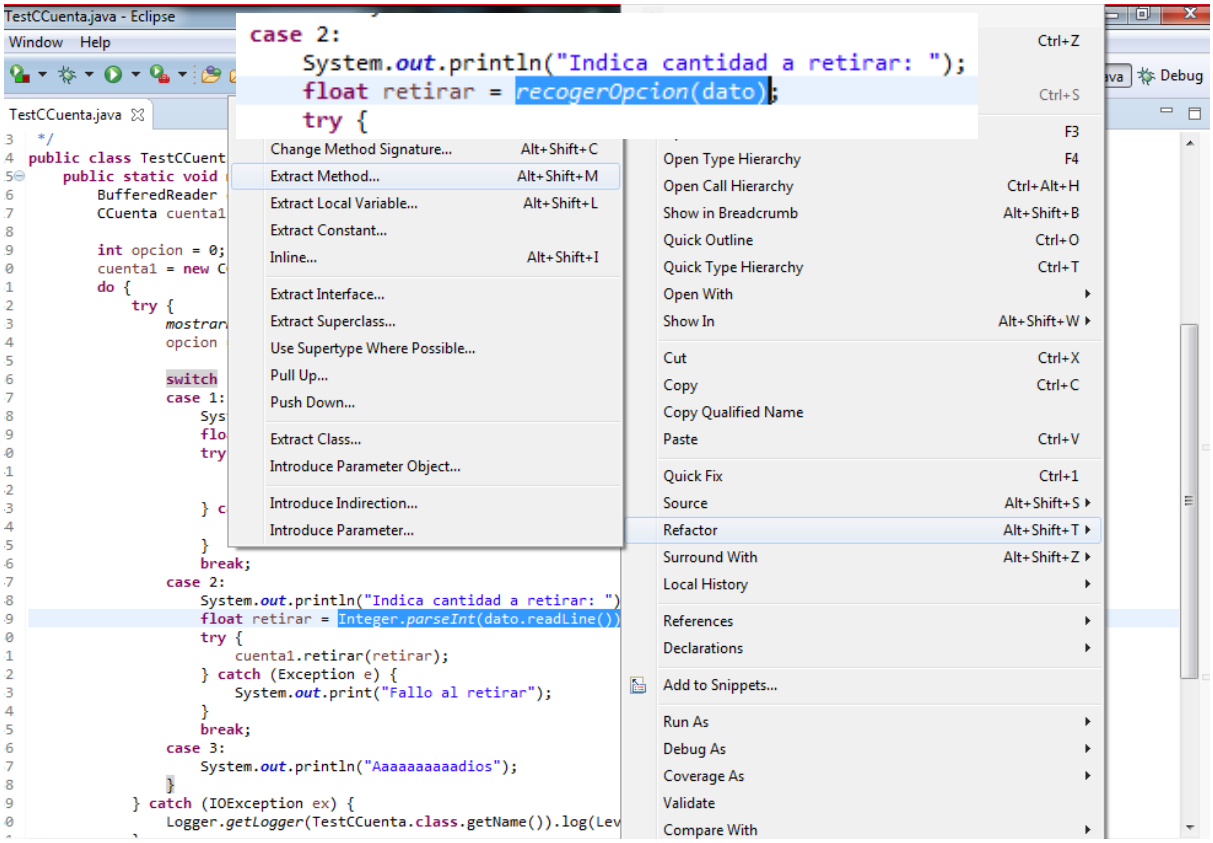
Nos saldrá esta interfaz en la que pondremos en "Method name" el nombre del método que queramos crear, en nuestro caso será mostrarMenu.

Opcionalmente podríamos marcar "Generate method comment" que nos generará los comentarios por defecto.



Y así implementaría los métodos que creamos.

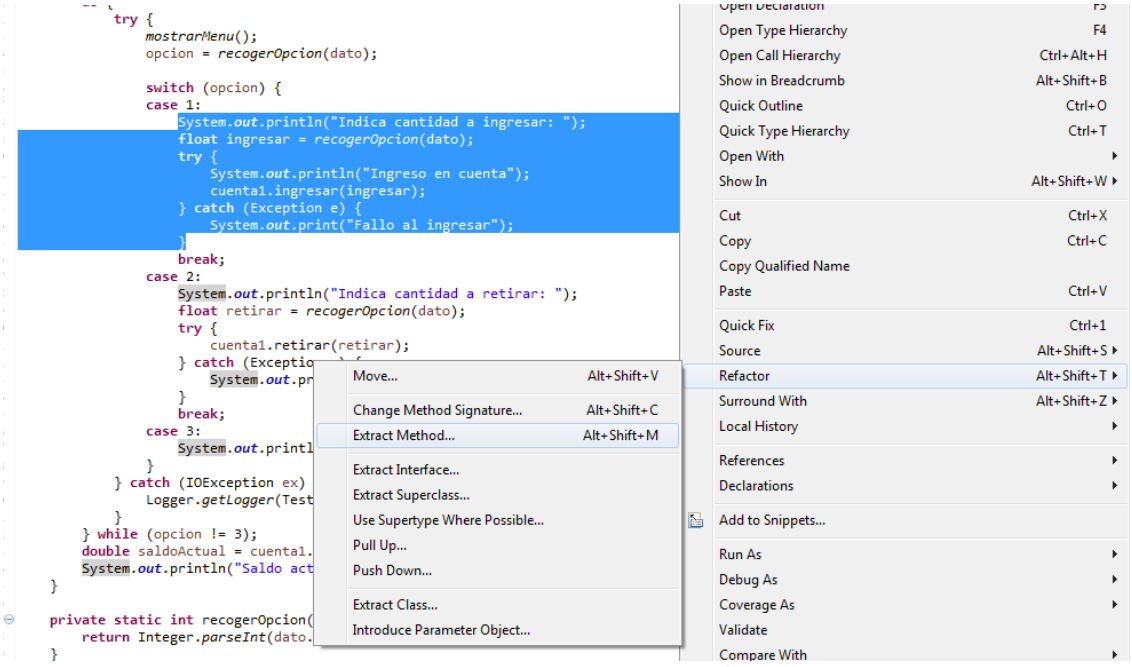
Ahora crearemos el método recogerOpción, siguiendo los pasos anteriores("Refactor">>"Extract Method">> "Method name" recogerOpcion)

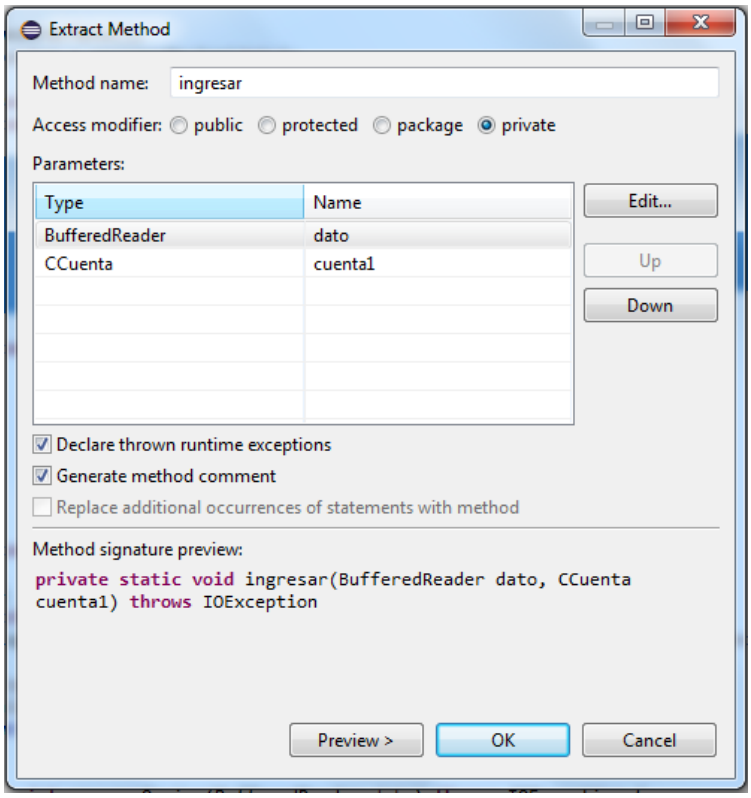


Y observamos que nos lo ha sustituído en todo el código (Marcando el método nos seleccionará todos los del documento):

```
        try {
            mostrarMenu();
            opcion = recogerOpcion(dato);

            switch (opcion) {
            case 1:
                System.out.println("Indica cantidad a ingresar: ");
                float ingresar = recogerOpcion(dato);
                try {
                    System.out.println("Ingreso en cuenta");
                    cuenta1.ingresar(ingresar);
                } catch (Exception e) {
                    System.out.print("Fallo al ingresar");
                }
                break;
            case 2:
                System.out.println("Indica cantidad a retirar: ");
                float retirar = recogerOpcion(dato);
                try {
                    cuenta1.retirar(retirar);
                } catch (Exception e) {
                    System.out.print("Fallo al retirar");
                }
                break;
            case 3:
                System.out.println("Aaaaaaaaaadios");
            }
        } catch (IOException ex) {
            Logger.getLogger(TestCCuenta.class.getName()).log(Level.SEVERE, null, ex);
        }
    } while (opcion != 3);
    double saldoActual = cuenta1.estado();
    System.out.println("Saldo actual: " + saldoActual);
}

private static int recogerOpcion(BufferedReader dato) throws IOException {
    return Integer.parseInt(dato.readLine());
}
```

Ahora crearemos el método ingresar, siguiendo los pasos anteriores("Refactor">>"Extract Method">> "Method name" ingresar) También podríamos pulsar Alt + Shift + M.





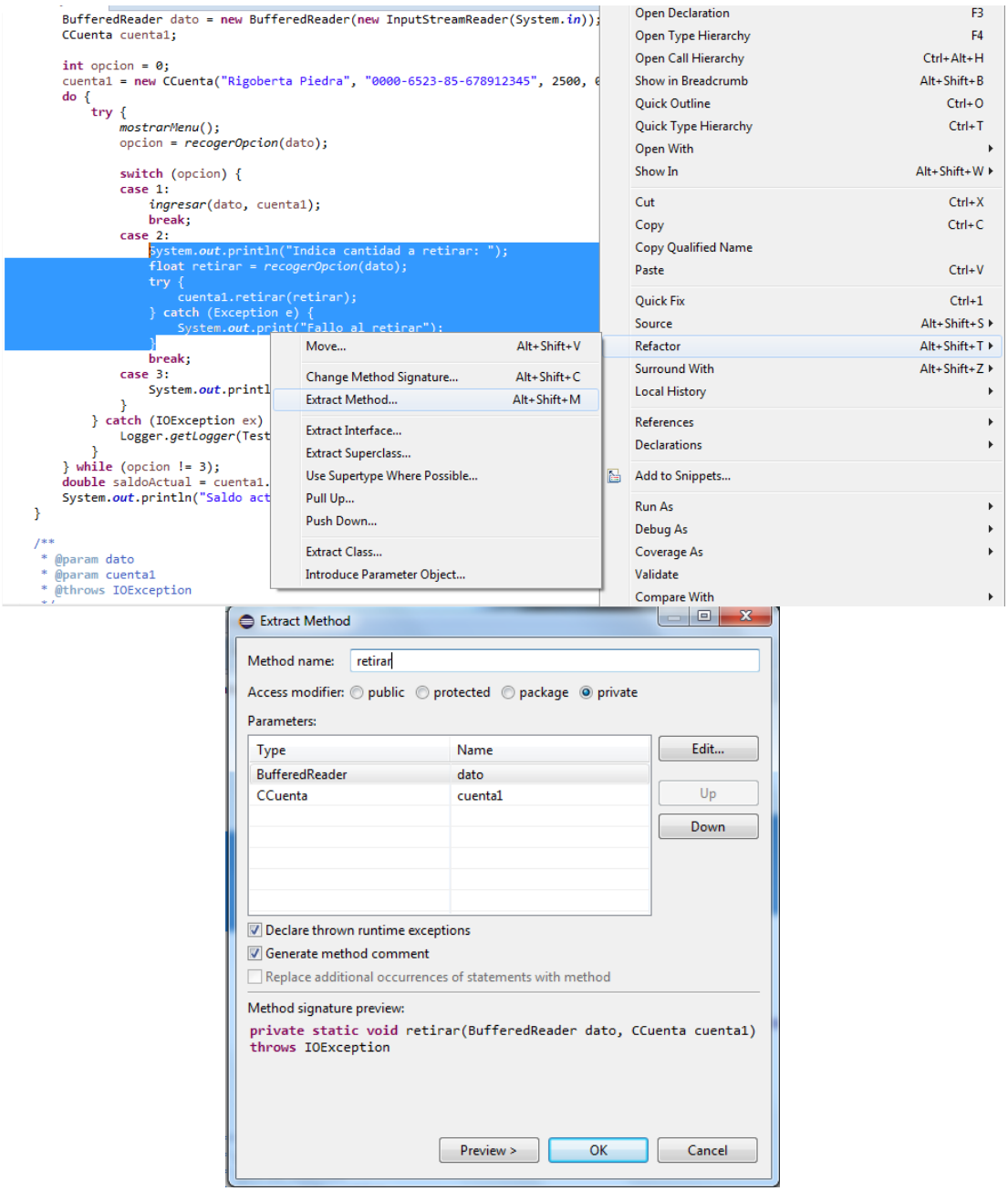
Marcaríamos "Declare thrown runtime exceptions" que añadiría las excepciones de java en tiempo de ejecución y nos generaría los comentarios automáticamente.

El resultado sería:

```
switch (opcion) {
case 1:
    ingresar(dato, cuenta1);
    break;
```

```
/**
 * @param dato
 * @param cuenta1
 * @throws IOException
 */
private static void ingresar(BufferedReader dato, CCuenta cuenta1) throws IOException {
    System.out.println("Indica cantidad a ingresar: ");
    float ingresar = recogerOpcion(dato);
    try {
        System.out.println("Ingreso en cuenta");
        cuenta1.ingresar(ingresar);
    } catch (Exception e) {
        System.out.print("Fallo al ingresar");
    }
}
```

Ahora crearemos el método retirar, siguiendo los pasos anteriores("Refactor">>"Extract Method">> "Method name" retirar) También podríamos pulsar Alt + Shift + M.



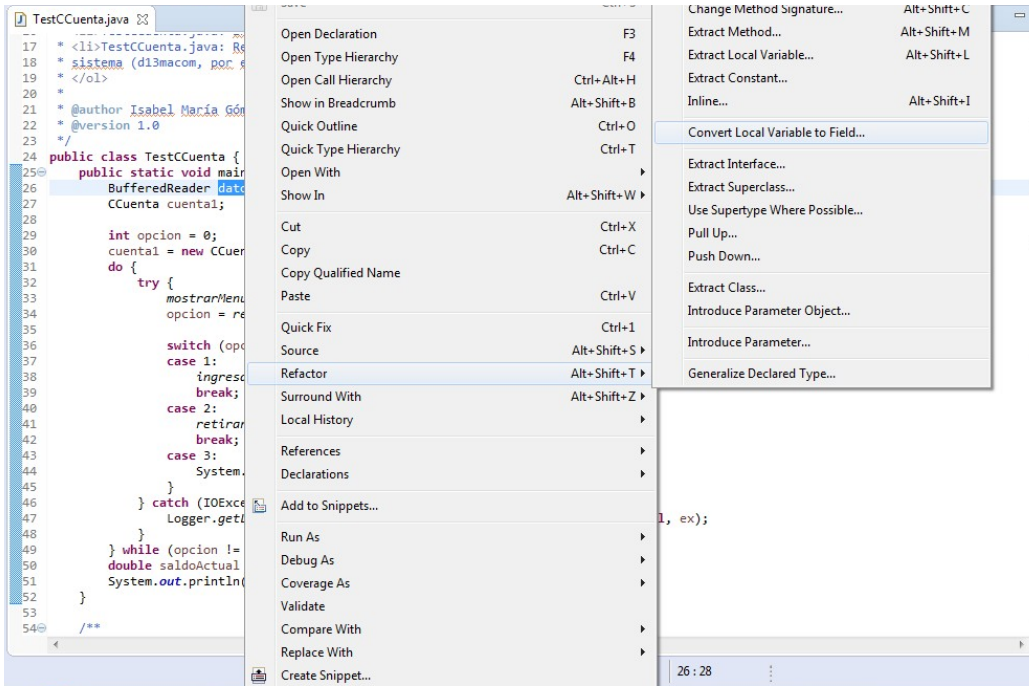
Marcaríamos "Declare thrown runtime exceptions" que añadiría las excepciones de java en tiempo de ejecución y nos generaría los comentarios automáticamente.

El resultado sería:

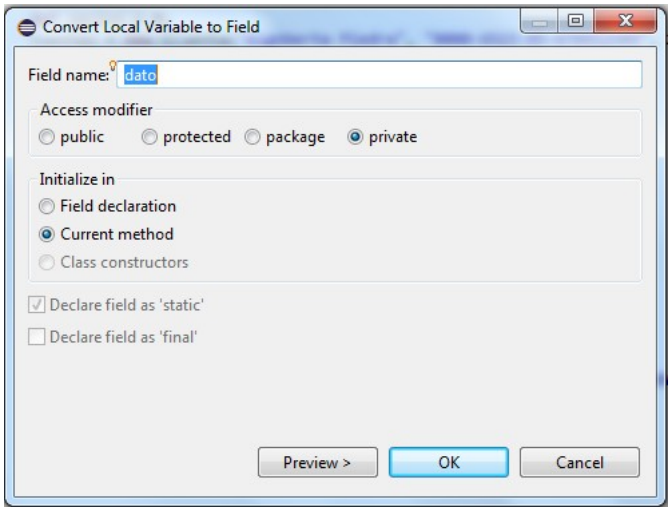
```
            case 2:
                retirar(dato, cuenta1);
                break;

    /**
     * @param dato
     * @param cuenta1
     * @throws IOException
     */
    private static void retirar(BufferedReader dato, CCuenta cuenta1) throws IOException {
        System.out.println("Indica cantidad a retirar: ");
        float retirar = recogerOpcion(dato);
        try {
            cuenta1.retirar(retirar);
        } catch (Exception e) {
            System.out.print("Fallo al retirar");
        }
    }
```

1.1.1.2 TestCCuenta.java: Convierte dato en campo

Ahora seleccionaríamos la variable "dato"(`BufferedReader dato = new BufferedReader(new InputStreamReader(System.in))`) hacemos clic derecho "Refactor">>"Convert Local Variable to Field", que lo convertirá en un campo estático que nos ahorrará tener que pasárselo por parámetros.



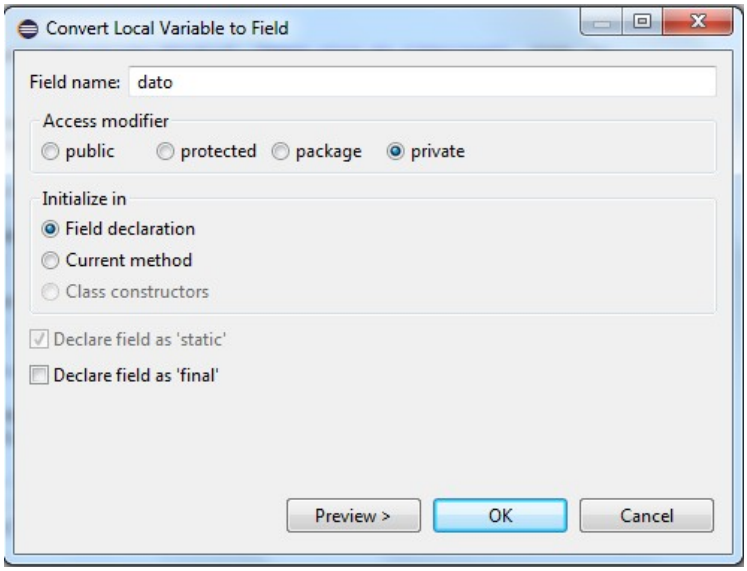
Nos saldría la interfaz en la que podemos seleccionar el modificador de visilidad(public, protected, package, private) y podemos seleccionar dónde lo inicializa si en el main o en la declaración del mismo.



El resultado sería:

Según "Field declaration" ó inicialización dónde se declara el campo.

Nos daría también la opción de colocarla como campo final, pero en nuestro caso no es recomendable porque nuestro campo debe cambiar de valor para sernos útil.




```
public class TestCCuenta {
    private static BufferedReader dato = new BufferedReader(new InputStreamReader(System.in));

    public static void main(String[] args) {
        CCuenta cuenta1;
```

O bien, la más correcta "Current method" que sólo declararía el campo pero no lo inicializaría.

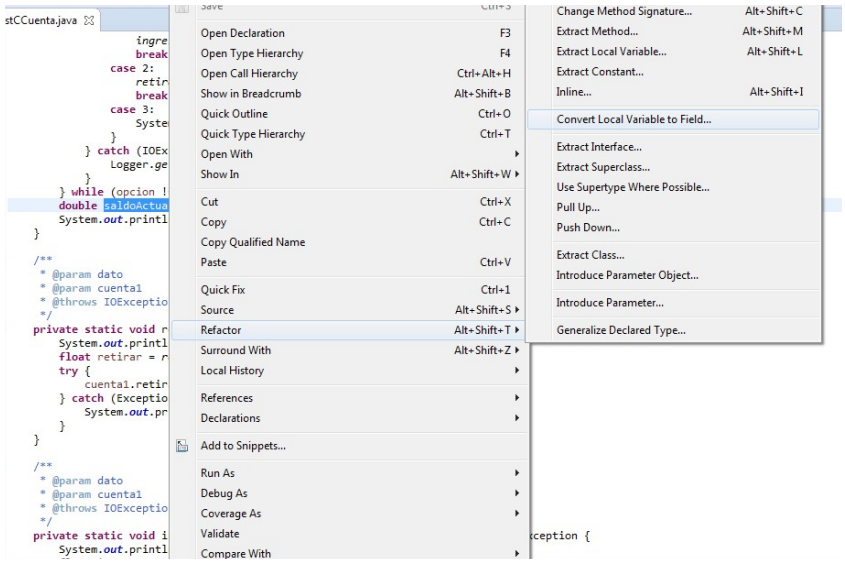
```
public class TestCCuenta {
    private static BufferedReader dato;

    public static void main(String[] args) {
        dato = new BufferedReader(new InputStreamReader(System.in));
        CCuenta cuenta1;
```

1.1.1.3 TestCCuenta.java: Incorpora saldoActual en la misma línea.

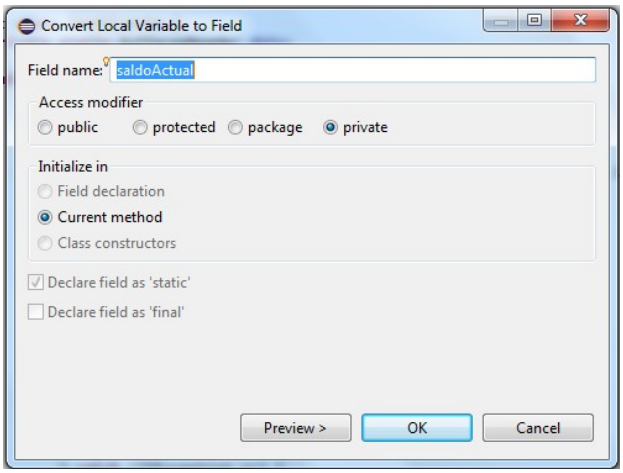
Ahora lo implementaríamos como en la primera forma que hemos visto. "Field Declaration".

Recordando los pasos anteriores tendríamos que:

Seleccionaríar la variable "saldoActual"(`double saldoActual = cuenta1.estado();`) hacemos clic derecho "Refactor">>"Convert Local Variable to Field", que lo convertirá en un campo estático que nos ahorrará tener que pasárselo por parámetros.



Nos saldría la interfaz en la que podemos seleccionar el modificador de visilidad(public, protected, package, private) y podemos seleccionar dónde lo inicializa si en el main o en la declaración del mismo.



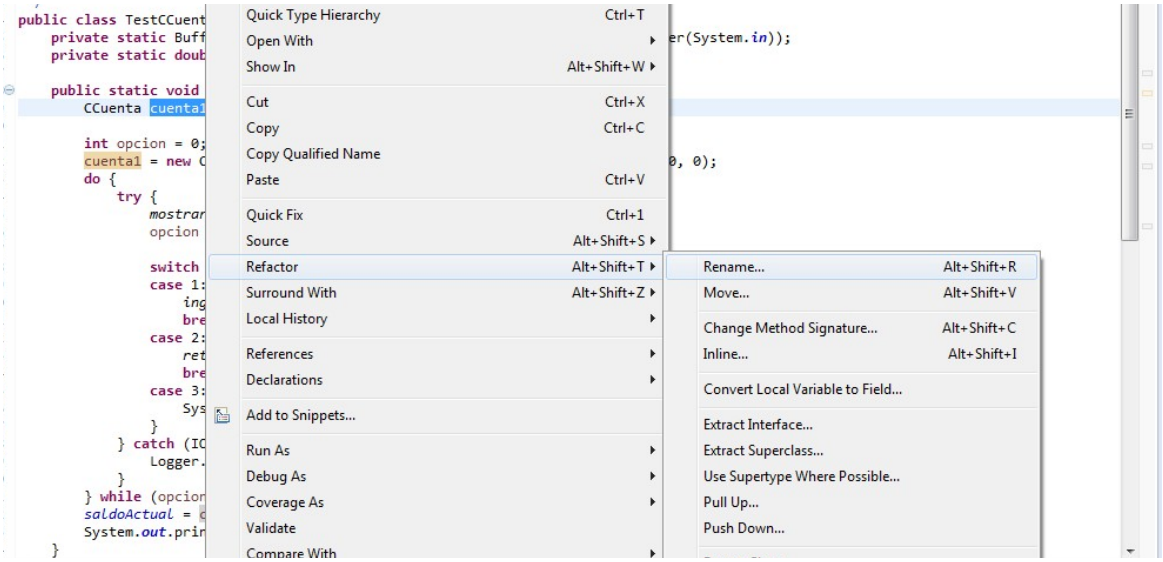
Ya está seleccionada por defecto la de "Current method" porque a diferencia de la variable anterior no la inicializa mediante un constructor si no con una invocación a un método, por lo que no nos dejará realizar las otras opciones. Y pulsamos sobre "Ok"

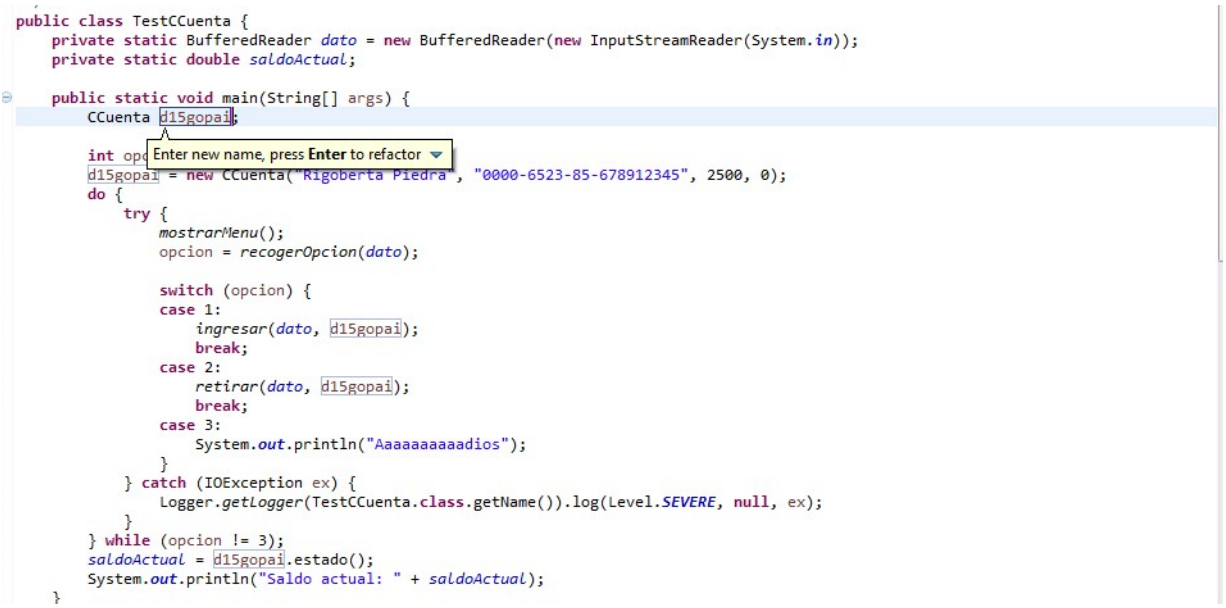
"Current method" que sólo declararía el campo en la clase pero no lo inicializaría.

```
public class TestCCuenta {
    private static BufferedReader dato;
    private static double saldoActual;

    public static void main(String[] args) {
        dato = new BufferedReader(new InputStreamReader(System.in)
        CCuenta cuenta1;

        int opcion = 0;
        cuenta1 = new CCuenta("Rigoberta Piedra", "0000-6523-85-67
        do {
            try {
                mostrarMenu();
                opcion = recogerOpcion(dato);

                switch (opcion) {
                case 1:
                    ingresar(dato, cuenta1);
                    break;
                case 2:
                    retirar(dato, cuenta1);
                    break;
                case 3:
                    System.out.println("Aaaaaaaaadios");
                }
            } catch (IOException ex) {
                Logger.getLogger(TestCCuenta.class.getName()).log(
            }
        } while (opcion != 3);
        saldoActual = cuenta1.estado();
        System.out.println("Saldo actual: " + saldoActual);
    }
```

1.1.1.4 TestCCuenta.java: Renombra la variable local cuenta1 con tu cuenta en el sistema (d13macom, por ejemplo).

Para renombrar una variable sólo tendríamos que pulsar botón derecho sobre la variable que queramos modificar. O bien con el comando Alt + Shift + R





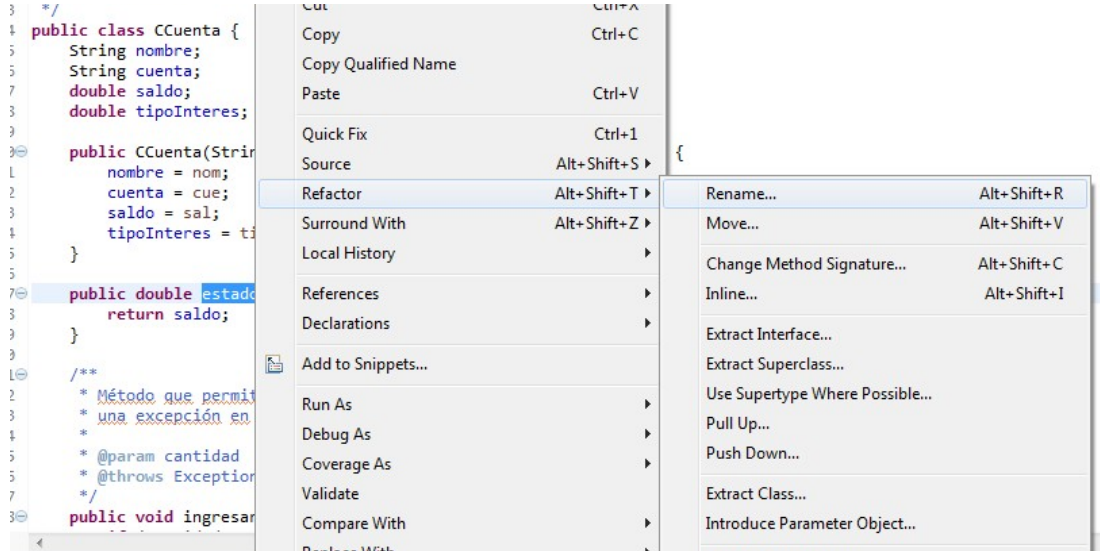

```
public class TestCCuenta {
    private static BufferedReader dato = new BufferedReader(new InputStreamReader(System.in));
    private static double saldoActual;

    public static void main(String[] args) {
        CCuenta d15gopai;

        int opc
        d15gopai = new CCuenta("Rigoberta Piedra", "0000-6523-85-678912345", 2500, 0);
        do {
            try {
                mostrarMenu();
                opcion = recogerOpcion(dato);

                switch (opcion) {
                case 1:
                    ingresar(dato, d15gopai);
                    break;
                case 2:
                    retirar(dato, d15gopai);
                    break;
                case 3:
                    System.out.println("Aaaaaaaaadios");
                }
            } catch (IOException ex) {
                Logger.getLogger(TestCCuenta.class.getName()).log(Level.SEVERE, null, ex);
            }
        } while (opcion != 3);
        saldoActual = d15gopai.estado();
        System.out.println("Saldo actual: " + saldoActual);
    }
```

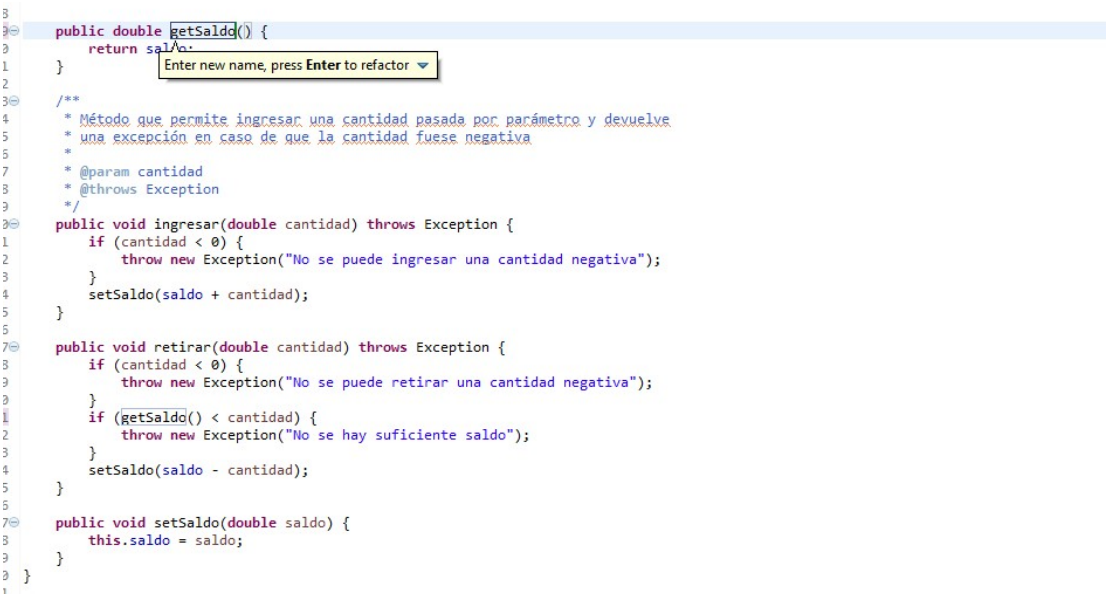
Y observamos cómo lo cambiará en todo el código. Y pulsamos enter para guardar los cambios.

1.1.2 En la clase CCuenta.java:

1.1.2.1 CCuenta.java: Cambia el nombre del método "estado" por "getCuenta"

Nos vamos a la clase del objeto: Ccuenta. Seleccionaríamos el nombre del método, pulsaríamos clic derecho >> "Refactor">>"Rename" >> "getSaldo" >> pulsamos Enter.

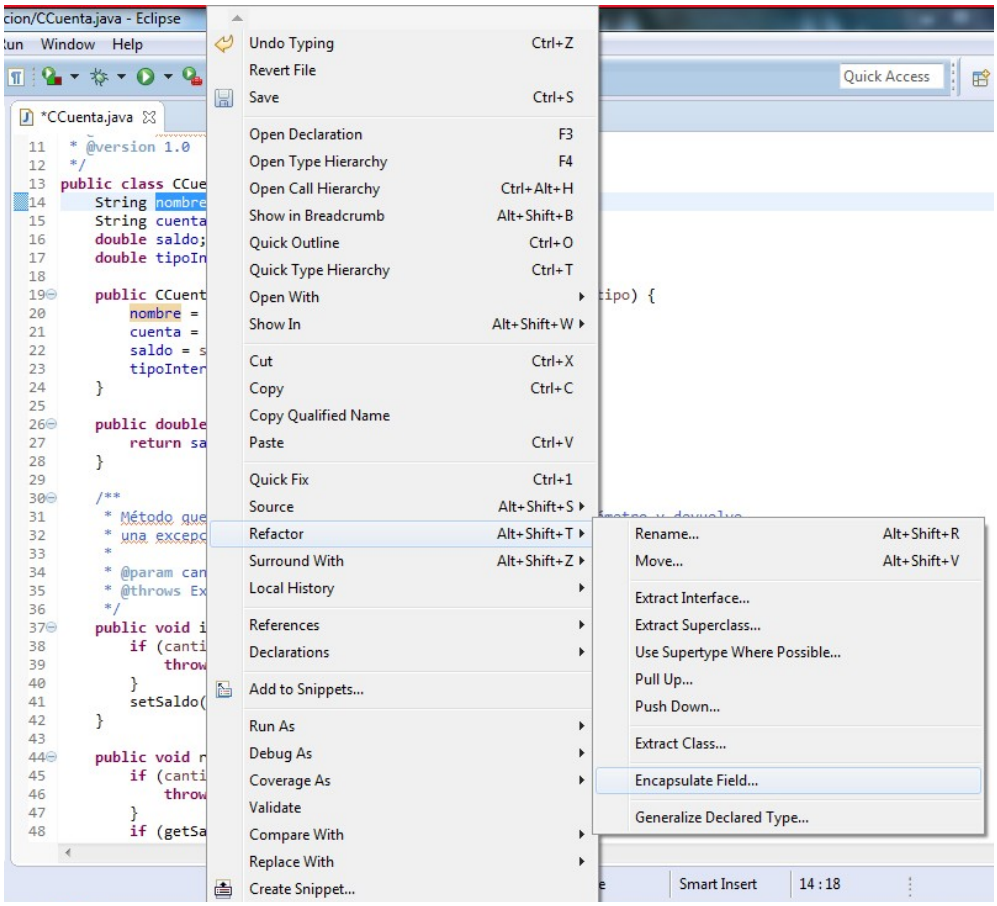




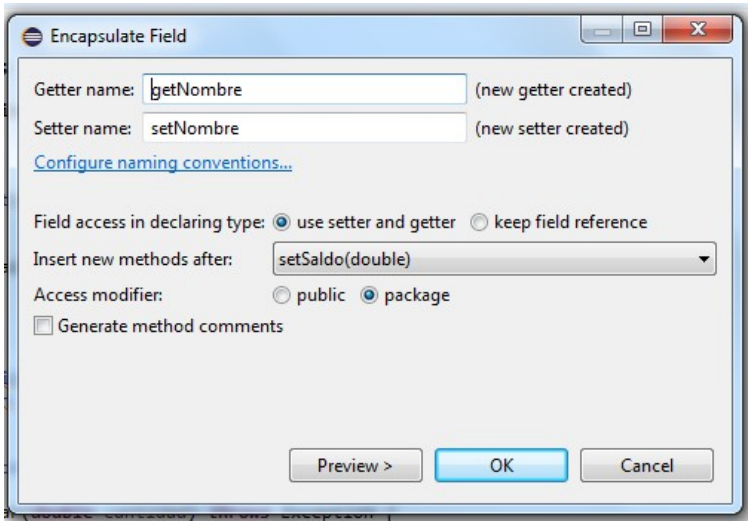

```
public class CCuenta {
    String nombre;
    String cuenta;
    double saldo;
    double tipoInteres;

    public CCuenta(String nom, String cue, double sal, double ti) {
        nombre = nom;
        cuenta = cue;
        saldo = sal;
        tipoInteres = ti;
    }

    public double getSaldo() {
        return saldo;
    }

    /**
     * Método que permite ingresar una cantidad pasada por parámetro y devuelve
     * una excepción en caso de que la cantidad fuese negativa
     *
     * @param cantidad
     * @throws Exception
     */
    public void ingresar(double cantidad) throws Exception {
        if (cantidad < 0) {
            throw new Exception("No se puede ingresar una cantidad negativa");
        }
        setSaldo(saldo + cantidad);
    }

    public void retirar(double cantidad) throws Exception {
        if (cantidad < 0) {
            throw new Exception("No se puede retirar una cantidad negativa");
        }
        if (getSaldo() < cantidad) {
            throw new Exception("No se hay suficiente saldo");
        }
        setSaldo(saldo - cantidad);
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }
}
```

1.1.2.2 CCuenta.java: Encapsula los campos y usa los métodos generados.

Seleccionamos el campo "nombre" pulsamos botón derecho>>"Refactor">>"Encapsulate Field"



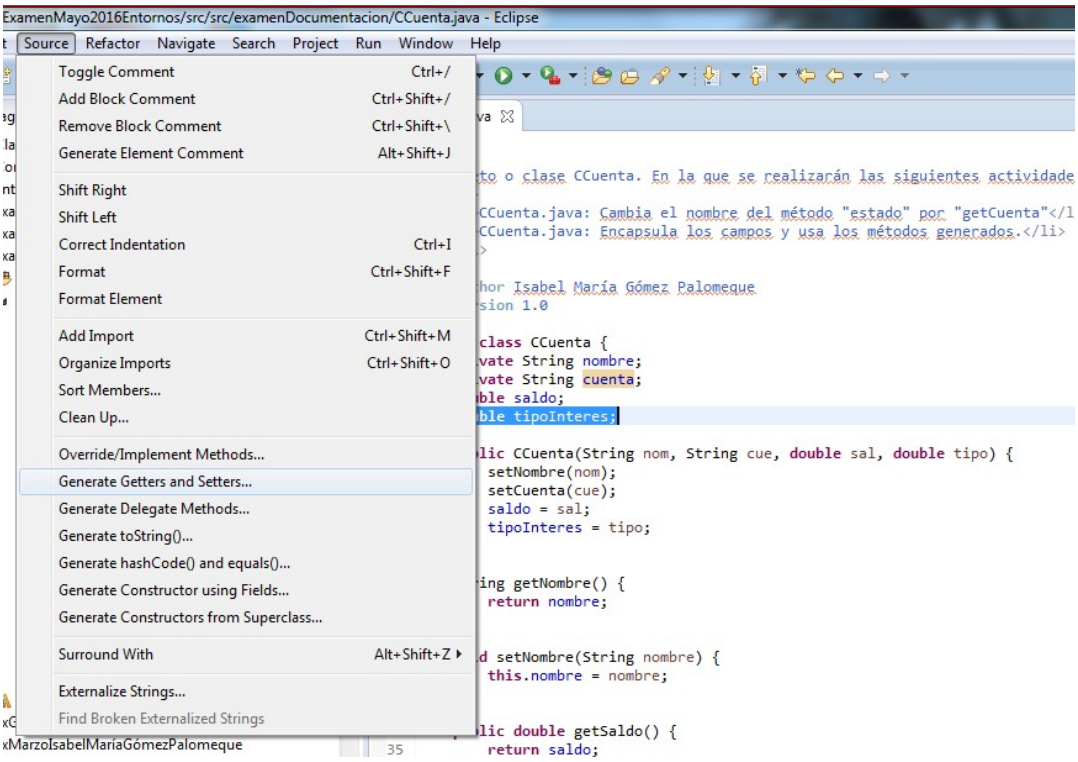
Nos saldrá esta interfaz, nos dejará cambiar el nombre de los setters o getters, si queremos implementarlos o no en el constructor "use setter and getter" o no implementarlos "keep field reference". El orden en el que se colocarán los métodos, en este caso después que setSaldo(double). La visibilidad de los métodos creados "public" "package". Y por último si queremos que nos cree automáticamente los comentarios.



```java
public class CCuenta {
    private String nombre;
    String cuenta;
    double saldo;
    double tipoInteres;

    public CCuenta(String nom, String cue, double sal, double tipo) {
        setNombre(nom);
        cuenta = cue;
        saldo = sal;
        tipoInteres = tipo;
    }

    String getNombre() {
        return nombre;
    }

    void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public double getSaldo() {
        return saldo;
    }
```

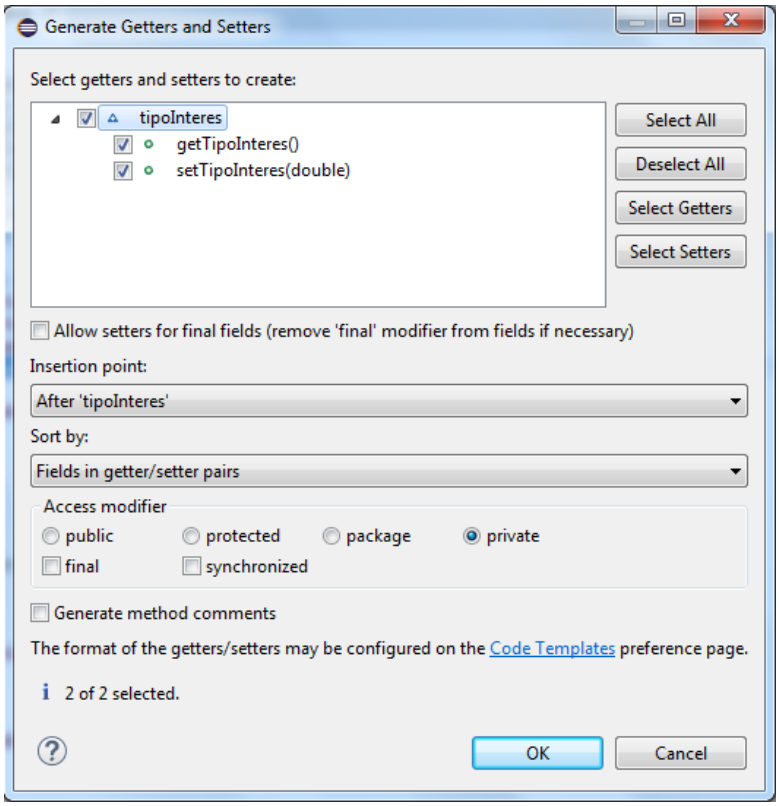
Y ya observaríamos cómo nos encapsularía el campo “nombre”.

Otra forma de hacerlo sería irnos a la barra de herramientas, seleccionar “Source”>>”Generate Getters and Setters”



Seleccionamos qué queremos que nos genere, setters o getters. E igual que lo anterior sólo que aquí tenemos más opciones de visibilidad como “protected” y “private” el inconveniente es que no te lo sustituye en el constructor.

También posee “final” y “synchronized”




```java
public class CCuenta {
    private String nombre;
    private String cuenta;
    double saldo;
    double tipoInteres;

    private double getTipoInteres() {
        return tipoInteres;
    }

    private void setTipoInteres(double tipoInteres) {
        this.tipoInteres = tipoInteres;
    }

    public CCuenta(String nom, String cue, double sal, double tipo) {
        setNombre(nom);
        setCuenta(cue);
        saldo = sal;
        tipoInteres = tipo;
    }
```

Ahora deberíamos modificar el constructor y el modificador de visibilidad del campo.

```java
public class CCuenta {
    private String nombre;
    private String cuenta;
    double saldo;
    private double tipoInteres;

    double getTipoInteres() {
        return tipoInteres;
    }

    private void setTipoInteres(double tipoInteres) {
        this.tipoInteres = tipoInteres;
    }

    public CCuenta(String nom, String cue, double sal, double tipo) {
        setNombre(nom);
        setCuenta(cue);
        saldo = sal;
        setTipoInteres(tipo);
    }
```

1.2 Documentación:

1.2.1 Deberás documentar todo el proyecto:

1.2.2 Inserta todos los comentarios Javadoc posibles.

1.2.2.1 Comentario de varias líneas:

```java
/*
* Podría ponerse sólo una excepción de que
* fuese nulo mediante NuloException e y crear
* la excepción. O bien, Exception = Captura
* todas las excepciones
*/
```

1.2.2.2 Comentario de una línea:

```java
//Devuelve un número entero introducido por teclado
```

1.2.2.3 Comentario de javadoc:

```java
/**
 * Asigna el valor saldo a su campo.
 *
 * @param saldo cantidad de dinero en la cuenta.
 */
```

1.2.3 Introduce como autor tu nombre y apellidos.

```java
/**
 * Objeto o clase CCuenta. En la que se realizarán las siguientes activida-
des:
 * <ol>
 * <li>CCuenta.java: Cambia el nombre del método "estado" por "getSaldo"</li>
 * <li>CCuenta.java: Encapsula los campos y usa los métodos generados.</li>
 * </ol>
 *
 * @author Isabel María Gómez Palomeque
 * @version 1.0
 */
```

1.2.4 Genera y entrega la documentación Javadoc para todo el proyecto

1.3 Repositorio con git:

Mediante comandos git, crea un repositorio local con el resultado. Una vez creado súbelo a tu cuenta de github y envía la dirección para poder visualizarlo. Incluirá lo siguiente:

- Proyecto resultante de la refactorización y de la documentación
- "ApellidosNombreRefactorizacion.pdf" resultante de la refactorización.
- "ApellidosNombreRepositorio.pdf" resultante de este apartado. Aparecerán pantallazos con los comandos utilizados en la consola git.