

**Tipo de Refactorización:** "Write simple units of code"

**Autor:** Luken Larrañaga

**Ubicación del issue:** DataAccess.java

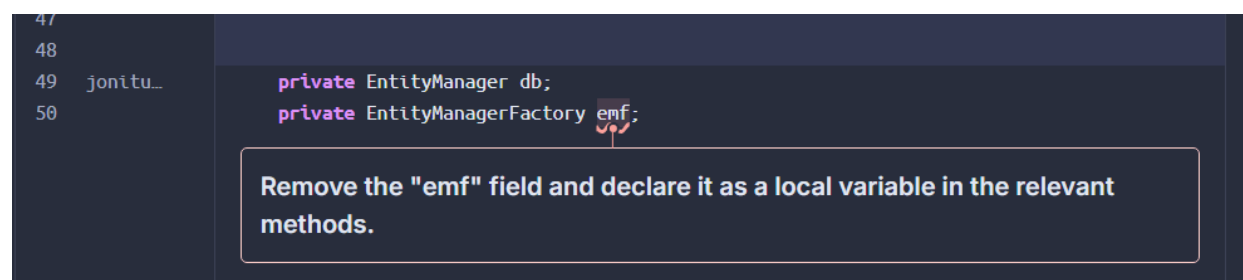
**Nombre del Issue:** Remove the "emf" field and declare it as a local variable in the relevant methods

### Descripción del Error

Cuando el valor de un campo privado se asigna siempre en los métodos de una clase antes de ser leído, no se está utilizando para almacenar información de la clase. Por lo tanto, debería convertirse en una variable local en los métodos relevantes para evitar cualquier malentendido.

**Cambios Realizados en el Código:** Se ha refactorizado la variable global "EntityManagerFactory emf" y se ha convertido en una variable local en el método que la usa, en el método open() en este caso.

### Código Inicial



### Código Refactorizado

```
public void open() {
    EntityManagerFactory emf;

    String fileName = c.getDbFilename();
    if (c.isDatabaseLocal()) {
        emf = Persistence.createEntityManagerFactory("objectdb:" + fileName);
        db = emf.createEntityManager();
    } else {
        Map<String, String> properties = new HashMap<>();
        properties.put(key:"javax.persistence.jdbc.user", c.getUser());
        properties.put(key:"javax.persistence.jdbc.password", c.getPassword());

        emf = Persistence.createEntityManagerFactory(
            "objectdb://" + c.getDatabaseNode() + ":" + c.getDatabasePort() +
            db = emf.createEntityManager();
    }
    System.out.println("DataAccess opened => isDatabaseLocal: " + c.isDatabaseLocal());
}
```

**Tipo de Refactorización:** "Keep unit interfaces small"

**Autor:** Luken Larrañaga

**Ubicación del issue:** DataAccess.java

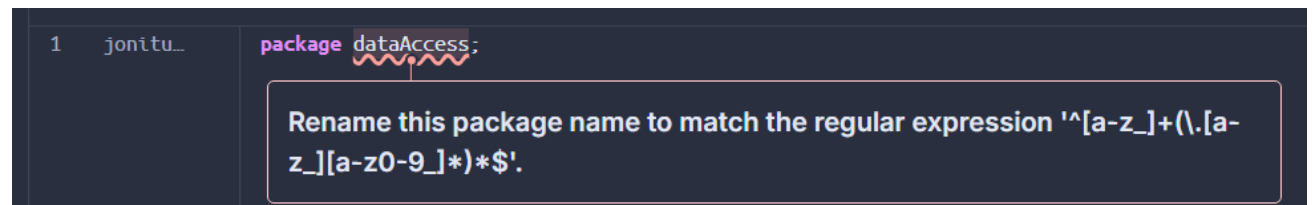
**Nombre del Issue:** Rename this package name to match the regular expression  
'^[a-z\_]+(\\.[a-z\_][a-z0-9\_]\*)\*\$'

### Descripción del Error


Las convenciones de nomenclatura compartidas mejoran la legibilidad y permiten que los equipos colaboren de manera eficiente. Esta regla verifica que todos los nombres de paquetes coincidan con una expresión regular proporcionada.

**Cambios Realizados en el Código:** Se ha cambiado el nombre del paquete DataAccess para respetar el uso de expresiones regulares. En su lugar se ha establecido el nombre data\_access

### Código Inicial



### Código Refactorizado

```
src > main > java > data_access >  DataAccess.java > ...  
1 package data_access;  
2
```

**Tipo de Refactorización:** "Write short units of code"

**Autor:** Luken Larrañaga

**Ubicación del issue:** DataAccess.java

**Nombre del Issue:** Immediately return this expression instead of assigning it to the temporary variable "arrivingCities"

### Descripción del Error

Declarar una variable solo para devolverla o lanzarla inmediatamente se considera una mala práctica porque añade una complejidad innecesaria al código. Esta práctica puede dificultar la lectura y comprensión del código, ya que introduce un paso adicional que no aporta valor. En lugar de declarar una variable y luego devolverla o lanzarla de inmediato, generalmente es mejor devolver o lanzar el valor directamente. Esto hace que el código sea más limpio, simple y fácil de entender.

**Cambios Realizados en el Código:** Se ha eliminado la creación de una variable innecesaria que solo se utilizaba para guardar y devolver inmediatamente el valor devuelto por la query. En su lugar ahora se devuelve directamente el valor de la query.

### Código Inicial

```
229         public List<String> getArrivalCities(String from) {
230             TypedQuery<String> query = db.createQuery(
231                 "SELECT DISTINCT r.to FROM Ride r WHERE r.from=?1 ORDER BY r.to",
232                 String.class);
233             query.setParameter(1, from);
234             List<String> arrivingCities = query.getResultList();
235
236             return arrivingCities;
237         }
```

Immediately return this expression instead of assigning it to the temporary variable "arrivingCities".

### Código Refactorizado

```
227         public List<String> getArrivalCities(String from) {
228             TypedQuery<String> query = db.createQuery(qlString:"SELECT DISTINCT r.to FROM Ride r WHERE r.from=?1 ORDER BY r.to",
229                 resultClass:String.class);
230             query.setParameter(position:1, from);
231             return query.getResultList();
232         }
```

**Tipo de Refactorización:** "Duplicate code"

**Autor:** Luken Larrañaga

**Ubicación del issue:** DataAccess.java

**Nombre del Issue:** Define a constant instead of duplicating this literal "bookFreeze" 5 times

### Descripción del Error

Los Strings duplicados complican el proceso de refactorización y lo hacen propenso a errores, ya que cualquier cambio tendría que ser propagado en todas las ocurrencias.

**Cambios Realizados en el Código:** Se ha establecido una constante global para así evitar el uso repetitivo del string "bookFreeze". De esta manera, en el caso de que haya que cambiar el valor del string, se puede hacer una sola vez en la constante y no en cada uso del string.

### Código Inicial

```
160
161 llarra... Movement m1 = new Movement(traveler1, "bookFreeze", 20);
162
163 Movement m2 = new Movement(traveler1, "bookFreeze", 40);
164 Movement m3 = new Movement(traveler1, "bookFreeze", 5);
165 Movement m4 = new Movement(traveler2, "bookFreeze", 4);
166 jonitu... Movement m5 = new Movement(traveler1, "bookFreeze", 3);
167 Movement m6 = new Movement(driver1, "Deposit", 15);
Movement m7 = new Movement(traveler1, "Deposit", 168);
```

Define a constant instead of duplicating this literal "bookFreeze" 5 times.

### Código Refactorizado

```
38 public class DataAccess {
39     private static final String MADRID = "Madrid";
40     private static final String DONOSTIA = "Donostia";
41     private static final String ACCEPTED = "Accepted";
42     private static final String REJECTED = "Rejected";
43     private static final String BOOKFREEZE = "BookFreeze";
44     private static final String USERNAME = "username";
45 }
```

```
162 Movement m1 = new Movement(traveler1, BOOKFREEZE, kopurua:20);
163 Movement m2 = new Movement(traveler1, BOOKFREEZE, kopurua:40);
164 Movement m3 = new Movement(traveler1, BOOKFREEZE, kopurua:5);
165 Movement m4 = new Movement(traveler2, BOOKFREEZE, kopurua:4);
166 Movement m5 = new Movement(traveler1, BOOKFREEZE, kopurua:3);
167 Movement m6 = new Movement(driver1, eragiketa:"Deposit", kopurua:15);
168 Movement m7 = new Movement(traveler1, eragiketa:"Deposit", kopurua:168);
```

You, hac

**Tipo de Refactorización:** “Write Simple Units of Code”

**Autor:** Telmo Goicoechea

**Ubicación del issue:** DataAccess.java

**Nombre del Issue:** This block of commented-out lines of code should be removed.

#### Descripción del Error:

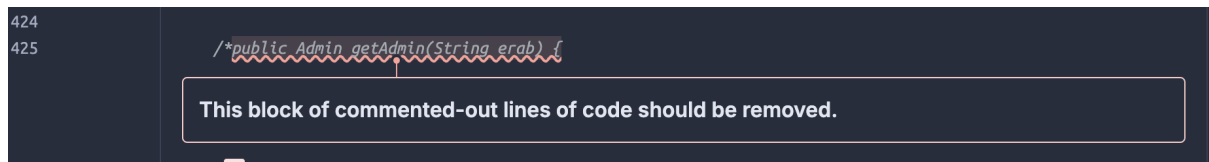
El código comentado distrae la atención del código que realmente se ejecuta. Crea un ruido que aumenta la complejidad del mantenimiento del código. Y como nunca se ejecuta, rápidamente se vuelve obsoleto e inválido.

El código comentado debería eliminarse y puede recuperarse del historial de control de versiones si es necesario.

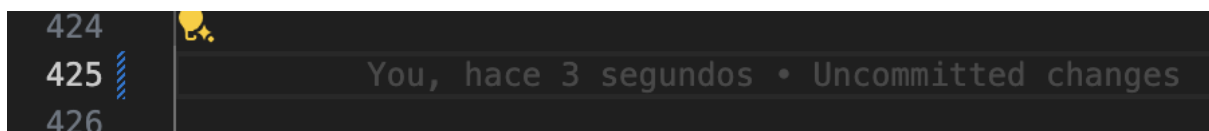
#### Cambios Realizados en el Código:

Se han eliminado las líneas de código no utilizadas.

#### Código Inicial



#### Código Refactorizado



**Tipo de Refactorización:** "Duplicate Code"

**Autor:** Telmo Goicoechea

**Ubicación del issue:** DataAccess.java

**Nombre del Issue:** Define a constant instead of duplicating this literal "SELECT d FROM Driver d WHERE d.username = :username" 3 times.

**Descripción del Error:** Los strings duplicados complican el proceso de refactorización y lo hacen propenso a errores, ya que cualquier cambio tendría que propagarse en todas las ocurrencias.

**Cambios Realizados en el Código:** Se ha creado una variable DRIVER\_QUERY con el valor para utilizarlo en la función.

### Código Inicial

```
402     public Driver getDriver(String erab) {
403         TypedQuery<Driver> query = db.createQuery("SELECT d FROM Driver d WHERE d.username = :username",
            Driver.class);
```

Define a constant instead of duplicating this literal "SELECT d FROM Driver d WHERE d.username = :username" 3 times.

### Código Refactorizado

```
1
2     private static final String DRIVER_QUERY = "SELECT d FROM Driver d WHERE d.username = :username";
3
4     public Driver getDriver(String erab) {
5         TypedQuery<Driver> query = db.createQuery(DRIVER_QUERY, resultClass:Driver.class);
6         query.setParameter(USERNAME, erab);
7         List<Driver> resultList = query.getResultList();
8         if (resultList.isEmpty()) {
```

**Tipo de Refactorización:** “Write short units of code”

**Autor:** Telmo Goicoechea

**Ubicación del issue:** BusinessLogicServer.java

**Nombre del Issue:** Extract this nested code block into a method.

### Descripción del Error:

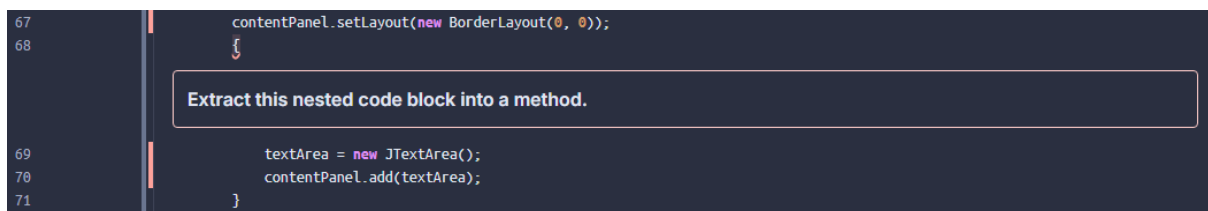
Los bloques de código anidados crean nuevos ámbitos donde las variables declaradas dentro son inaccesibles desde el exterior y su vida útil finaliza con el bloque.

Aunque esto puede parecer beneficioso, su uso dentro de una función a menudo sugiere que la función está sobrecargada. Por lo tanto, puede violar el principio de responsabilidad única y la función debe dividirse en funciones más pequeñas.

La presencia de bloques anidados que no afectan el flujo de control puede sugerir posibles errores en el código.

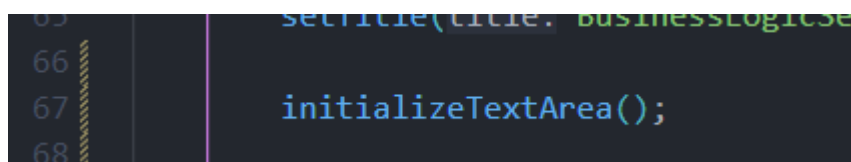
**Cambios Realizados en el Código:** Los bloques de código anidados se han extraído a métodos separados para mejorar la legibilidad y el mantenimiento del código.

### Código Inicial:



```
67 contentPanel.setLayout(new BorderLayout(0, 0));
68 {
69     textArea = new JTextArea();
70     contentPanel.add(textArea);
71 }
```

### Código Refactorizado:



```
65 setTitle(title, BusinessLogicServer
66
67 initializeTextArea();
68
```



```
Run main | Debug main | Run | Debug
39 public static void main(String[] args) {
40     try {
41         BusinessLogicServer dialog = new BusinessLogicServer();
42         dialog.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
43         dialog.setVisible(b:true);
44     } catch (Exception e) {
45         e.printStackTrace();
46     }
47
48 }
49 private void initializeTextArea() {
50     textArea = new JTextArea();
51     contentPanel.add(textArea);
52 }
53
54
```

**Tipo de Refactorización:** “Keep Unit Interfaces Small”

**Autor:** Telmo Goicoechea

**Ubicación del issue:** UtilDate.java

**Nombre del Issue:** Add a private constructor to hide the implicit public one.

**Descripción del Error:** Siempre que haya porciones de código que estén duplicadas y no dependan del estado de su clase contenedora, pueden ser centralizadas dentro de una "clase de utilidad". Una clase de utilidad es una clase que solo tiene miembros estáticos, por lo tanto, no debería ser instanciada.

**Cambios Realizados en el Código:** Se ha creado una constructora privada.

### Código Inicial

```
6
7 public class UtilDate {
8
9
```

Add a private constructor to hide the implicit public one.

### Código Refactorizado

```
public class UtilDate {

    private UtilDate() {}

    // private constructor to hide the implicit public one
}
```



**Tipo de Refactorización:** "Write simple units of code"

**Autor:** Unai Matas

**Ubicación del issue:** DataAccess.java

**Nombre del Issue** Refactor this method to reduce its Cognitive Complexity from 23 to the 15 allowed

### Descripción del Error

La complejidad cognitiva es una medida de la dificultad para comprender el flujo de control de una unidad de código. El código con alta complejidad cognitiva es difícil de leer, entender, probar y modificar.

Como regla general, una complejidad cognitiva alta es señal de que el código debe refactorizarse en partes más pequeñas y fáciles de gestionar.

**Cambios Realizados en el Código:** Se ha refactorizado el método deleteUser creando los nuevos métodos deleteDriverRelatedData y deleteTravelerRelatedData para reducir el número de líneas de código de deleteUser.

### Código Inicial:

```
public void deleteUser(User us) {    Refactor this method to reduce its Cognitive Complexity
    try {
        if (us.getMota().equals("Driver")) {
            List<Ride> r1 = getRidesByDriver(us.getUsername());
            if (r1 != null) {
                for (Ride ri : r1) {
                    cancelRide(ri);
                }
            }
            Driver d = getDriver(us.getUsername());
            List<Car> c1 = d.getCars();
            if (c1 != null) {
                for (int i = c1.size() - 1; i >= 0; i--) {
                    Car ci = c1.get(i);
                    deleteCar(ci);
                }
            }
        } else {
            List<Booking> lb = getBookedRides(us.getUsername());
            if (lb != null) {
                for (Booking li : lb) {
                    li.setStatus(REJECTED);
                    li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
                }
            }
            List<Alert> la = getAlertsByUsername(us.getUsername());
            if (la != null) {
                for (Alert lx : la) {
                    deleteAlert(lx.getAlertNumber());
                }
            }
        }
        db.getTransaction().begin();
        us = db.merge(us);
        db.remove(us);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();    Print Stack Trace
    }
}
```

## Código Final:

```
public void deleteUser(User us) {
    if (us.getMota().equals("Driver")) {
        deleteDriverRelatedData(us);
    } else {
        deleteTravelerRelatedData(us);
    }
    db.getTransaction().begin();
    us = db.merge(us);
    db.remove(us);
    db.getTransaction().commit();
} catch (Exception e) {
    e.printStackTrace();    Print Stack Trace
}

private void deleteDriverRelatedData(User us) {
    List<Ride> r1 = getRidesByDriver(us.getUsername());
    if (r1 != null) {
        for (Ride ri : r1) {
            cancelRide(ri);
        }
    }
    Driver d = getDriver(us.getUsername());
    List<Car> c1 = d.getCars();
    if (c1 != null) {
        for (int i = c1.size() - 1; i >= 0; i--) {
            Car ci = c1.get(i);
            deleteCar(ci);
        }
    }
}

private void deleteTravelerRelatedData(User us) {
    List<Booking> lb = getBookedRides(us.getUsername());
    if (lb != null) {
        for (Booking li : lb) {
            li.setStatus(REJECTED);
            li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
        }
    }
    List<Alert> la = getAlertsByUsername(us.getUsername());
    if (la != null) {
        for (Alert lx : la) {
            deleteAlert(lx.getAlertNumber());
        }
    }
}
```

**Tipo de Refactorización:** "Avoid duplicate code"

**Autor:** Unai Matas

**Ubicación del issue:** DataAccess.java

**Nombre del Issue** This block of commented-out lines of code should be removed.

### Descripción del Error

El código comentado distrae la atención del código realmente ejecutado. Crea un ruido que aumenta el código de mantenimiento. Y como nunca se ejecuta, queda rápidamente obsoleto e inválido.

El código comentado debe eliminarse y puede recuperarse del historial del control de código fuente si es necesario.

**Cambios Realizados en el Código:** Se ha eliminado el bloque comentado para mas claridad en el código.

### Código inicial:

```
TypedQuery<Long> driverQuery = db.createQuery(  
    "SELECT COUNT(d) FROM Driver d WHERE d.username = :username AND d.passwd = :passwd", Long.class);  
driverQuery.setParameter(USERNAME, erab);  
driverQuery.setParameter("passwd", passwd);  
Long driverCount = driverQuery.getSingleResult();  
  
/*TypedQuery<Long> adminQuery = db.createQuery(  
    "SELECT COUNT(a) FROM Admin a WHERE a.username = :username AND a.passwd = :passwd", Long.class);  
adminQuery.setParameter(USERNAME, erab);  
adminQuery.setParameter("passwd", passwd);  
Long adminCount = adminQuery.getSingleResult();*/  
  
boolean isAdmin=((erab.compareTo("admin")==0) && (passwd.compareTo(adminPass)==0));  
return travelerCount > 0 || driverCount > 0 || isAdmin;
```

### Código final:

```
TypedQuery<Long> driverQuery = db.createQuery(  
    "SELECT COUNT(d) FROM Driver d WHERE d.username = :username AND d.passwd = :passwd", Long.class);  
driverQuery.setParameter(USERNAME, erab);  
driverQuery.setParameter("passwd", passwd);  
Long driverCount = driverQuery.getSingleResult();  
  
boolean isAdmin=((erab.compareTo("admin")==0) && (passwd.compareTo(adminPass)==0));  
return travelerCount > 0 || driverCount > 0 || isAdmin;  
  
public Driver getDriver(String erab) {  
    TypedQuery<Driver> query = db.createQuery("SELECT d FROM Driver d WHERE d.username = :username", Driver.class);  
    query.setParameter(USERNAME, erab);  
    List<Driver> resultList = query.getResultList();  
    if (resultList.isEmpty()) {  
        return null;  
    } else {
```

**Tipo de Refactorización:** "Keep unit interfaces small"

**Autor:** Unai Matas

**Ubicación del issue:** BLFacadeImplementation.java

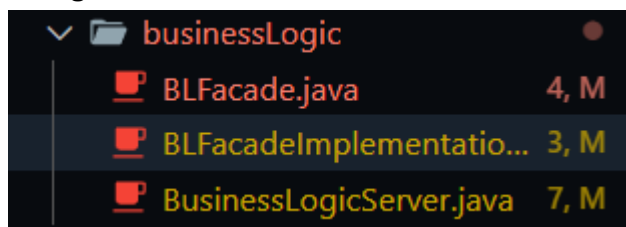
**Nombre del Issue** Rename this package name to match the regular expression `^[a-z_]+(\.[a-z_][a-z0-9_]*)*$`.

#### Descripción del Error:

Las convenciones de nomenclatura compartidas mejoran la legibilidad y permiten a los equipos colaborar con eficacia. Esta regla comprueba que todos los nombres de paquetes coincidan con una expresión regular proporcionada.

**Cambios Realizados en el Código:** Se ha cambiado el nombre del paquete `businessLogic` para respetar el uso de expresiones regulares. En su lugar se ha establecido el nombre `business_logic`. También se han hecho los cambios pertinentes en las demás clases para evitar errores.

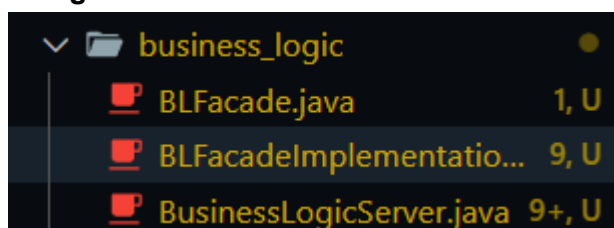
#### Código inicial:



```
package businessLogic;

import java.util.Date;
import java.util.List;
import java.util.logging.Logger;
```

#### Código final:



```
src > main > java > business_logic > BLFacadeImplementation.java > {} business_logic
1 package business_logic;
2
3 import java.util.Date;
4 import java.util.List;
5 import java.util.logging.Logger;
```

**Tipo de Refactorización:** "Write short units of code"

**Autor:** Unai Matas

**Ubicación del issue:** BusinessLogicServer.java

**Nombre del Issue:** Make this anonymous inner class a lambda

### Descripción del Error:

Antes de Java 8, la única manera de soportar parcialmente los cierres en Java era mediante el uso de clases internas anónimas. Java 8 introdujo las lambdas, que son significativamente más legibles y deben ser utilizadas en su lugar.

Esta regla se desactiva automáticamente cuando el sonar.java.source del proyecto es inferior a 8, ya que las expresiones lambda se introdujeron en Java 8.

**Cambios Realizados en el Código:** Se ha extraído el evento addActionListener para tomar la forma de una expresión lambda además, se ha eliminado el bloque comentado para más claridad en el código.

### Código inicial:

```
72      getContentPane().add(buttonPane, BorderLayout.SOUTH);
73      { Extract this nested code block into a method.
74          JButton okButton = new JButton("OK");
75          okButton.addActionListener(new ActionListener() { Make
76              public void actionPerformed(ActionEvent e) { Add @
77                  textArea.append("\n\n\nClosing the server... ");
78              }
79              //server.close(); This block of commented-
80              System.exit(1);
81          }
82      });
83      okButton.setActionCommand("OK");
84  }
```

### Código final:

```
70      getContentPane().add(buttonPane, BorderLayout.SOUTH);
71      { Extract this nested code block into a method.
72          JButton okButton = new JButton("OK");
73          okButton.addActionListener(e -> {
74              textArea.append("\n\n\nClosing the server... ");
75          });
76          System.exit(1);
77      });
78      okButton.setActionCommand("OK");
79  }
```