



ARQUITETURAS DE SOFTWARE

Refactoring da Ferramenta de Market Trading
(Terceiro trabalho prático)

Relatório

Conteúdo

CODE SMELLS DA PRIMEIRA VERSÃO DO TRABALHO PRÁTICO.....	3
BLOATERS.....	3
DISPENSABLES.....	3
COUPLERS	3
CODE SMELLS DA SEGUNDA VERSÃO DO TRABALHO PRÁTICO.....	4
BLOATERS.....	4
DISPENSABLES.....	4
COUPLERS	4
REFACTORING DA PRIMEIRA VERSÃO DO TRABALHO PRÁTICO	5
LONG METHOD (abrirCompra).....	0
LONG METHOD (abrirVenda)	0
LONG METHOD (addAtivo).....	0
LONG METHOD (fechaVenda).....	0
LONG METHOD (fechaCompra)	0
LONG METHOD (getAtivos)	0
LARGE CLASS (AppESS).....	0
LAZY CLASS.....	0
DATA CLASS	0
SPECULATIVE GENERALITY.....	0
DEAD CODE.....	0
INNAPPROPRIATE INTIMACY	0
REFACTORING DA SEGUNDA VERSÃO DO TRABALHO PRÁTICO	0
LONG METHOD (abrirCompra).....	0
LONG METHOD (abrirVenda)	0
LONG METHOD (addAtivo).....	0
LONG METHOD (fechaVenda).....	0
LONG METHOD (fechaCompra)	0
LONG METHOD (getAtivos)	0
LARGE CLASS (AppESS).....	0
LAZY CLASS.....	0
DATA CLASS	0
SPECULATIVE GENERALITY.....	0

DEAD CODE.....	0
INNAPROPRIATE INTIMACY	0
ANTI-PADRÃO DE DESENHO.....	0
ANTI PADRÃO – DARK SINGLETON	0
IMPATO DO REFACTORING	0
TP1	0
TP2	0
CONCLUSÃO.....	1

CODE SMELLS DA PRIMEIRA VERSÃO DO TRABALHO PRÁTICO

Depois de uma cuidada análise, foram identificados no primeiro trabalho prático os seguintes *code smells*:

BLOATERS

- i. Long Method – Foram identificados inúmeros casos de Long Method no código da aplicação, i.e., métodos cuja implementação se estende por mais de dez, quinze, linhas de código.
- ii. Large Class - Foi identificado um caso de Large Class, onde uma classe continha demasiados métodos e consequentemente linhas de código.

DISPENSABLES

- i. Data Class – Foram identificados casos de Data Class, i.e., classes que apenas apresentam como conteúdo getters e setters, ou seja, métodos de acesso aos seus próprios campos.
- ii. Lazy Class – Foram identificados casos de Lazy Class, i.e., classes pouco importantes, e que acabaram por ser esquecidas e deixadas de fora da solução final.
- iii. Speculative Generality – Foram identificados casos de Speculative Generality, i.e., classes não usadas, e que não fazem falta na solução final.
- iv. Dead Code – Foram identificados casos de Dead Code, i.e., código e bibliotecas que deixaram de ser precisos, mas que foram sendo mantidos no código.

COUPLERS

- i. Inappropriate intimacy – Foram identificados casos onde várias classes usam um campo de uma outra classe.

CODE SMELLS DA SEGUNDA VERSÃO DO TRABALHO PRÁTICO

Depois de uma cuidada análise, foram identificados no segundo trabalho prático os seguintes *code smells*:

BLOATERS

- iii. Long Method – Foram identificados inúmeros casos de Long Method no código da aplicação, i.e., métodos cuja implementação se estende por mais de dez, quinze, linhas de código.
- iv. Large Class - Foi identificado um caso de Large Class, onde uma classe continha demasiados métodos e consequentemente linhas de código.

DISPENSABLES

- v. Data Class – Foram identificados casos de Data Class, i.e., classes que apenas apresentam como conteúdo getters e setters, ou seja, métodos de acesso aos seus próprios campos.
- vi. Lazy Class – Foram identificados casos de Lazy Class, i.e., classes pouco importantes, e que acabaram por ser esquecidas e deixadas de fora da solução final.
- vii. Speculative Generality – Foram identificados casos de Speculative Generality, i.e., classes não usadas, e que não fazem falta na solução final.
- viii. Dead Code – Foram identificados casos de Dead Code, i.e., código e bibliotecas que deixaram de ser precisos, mas que foram sendo mantidos no código.

COUPLERS

- i. Inappropriate intimacy – Foram identificados casos onde várias classes usam um campo de uma outra classe.

REFACTORING DA PRIMEIRA VERSÃO DO TRABALHO PRÁTICO

De seguida, passo a apresentar os excertos de código que mostram a presença dos code smells acima referidos, seguidos dos excertos de código resultantes da aplicação de uma técnica de resolução desse mesmo code smell.

LONG METHOD (abrirCompra)

```
public void abrirCompra(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        Statement st1 = con.createStatement();
        rs = st1.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            ResultSet rs1;
            rs1 = st.executeQuery("SELECT COUNT(idAtivo) FROM ativo");
            rs1.next();
            int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
            Statement st4 = con.createStatement();
            String fullname = stocks.get(nome).getName();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] tempQuote = valoresMercado.split(", ");
            String tempQuotes = tempQuote[1];
            String[] resultQuote = tempQuotes.split(" ");
            String bid = resultQuote[1];
            String[] tempQuote2 = valoresMercado.split(", ");
            String tempQuotes2 = tempQuote2[0];
            String[] resultQuote2 = tempQuotes2.split(" ");
            String ask = resultQuote2[1];
            String query2 = "INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")";
            st4.executeUpdate(query2);
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            Statement st6 = con.createStatement();
            st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA");
            rs = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
            rs.next();
            float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
            float plafondResultante = plafondAnterior - montante;
            st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
        } else {
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
        }
    }
}
```

Figura 1 Método abrirCompra

```

        int idAtivo = ((Number) resultset.getObject(1)).intValue();
        String valoresMercado = YahooFinance.get(nome).getQuote().toString();
        String[] tempQuote = valoresMercado.split(", ");
        String tempQuotes = tempQuote[1];
        String[] resultQuote = tempQuotes.split(" ");
        String bid = resultQuote[1];
        String[] tempQuote2 = valoresMercado.split(", ");
        String tempQuotes2 = tempQuote2[0];
        String[] resultQuote2 = tempQuotes2.split(" ");
        String ask = resultQuote2[1];
        Statement s = con.createStatement();
        s.executeUpdate("UPDATE ativo SET ValorCompra = '" + ask + "', ValorVenda = '" + bid + "' WHERE idAtivo = " + idAtivo);
        Statement st6 = con.createStatement();
        ResultSet rs3;
        st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA
        rs3 = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs3.next();
        float plafondAnterior = ((Number) rs3.getObject(1)).floatValue();
        float plafondResultante = plafondAnterior - montante;
        st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
    }
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}

```

Figura 2 Método abrirCompra

REFACTORING

- Extract Method – Maior parte do código foi movido para dois métodos auxiliares novos e criados para calcular resultados e devolvê-los à função abrirCompra. abrirNewCompra, no caso de não haver informação dessa compra na BD e abrirOldCompra, caso haja referência a essa compra na BD.
- Replace Temp with Query – O uso de variáveis temporárias foi resolvido, visto que agora todo o código foi movido para os dois novos métodos. A função abrirCompra apenas usa os valores retornados por esses dois métodos. Por sua vez, para reduzir o número de parâmetros temporários existentes no código de implementação, que passará a encontrar-se em abrirNewCompra e abrirOldCompra, foram usadas as funções getBD e alterBD.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 3 Métodos getBD e alterBD

```

public void abrirNewCompra(String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet rs1;
    rs1 = getBD("SELECT COUNT(idAtivo) FROM ativo");
    rs1.next();
    int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
    String fullname = stocks.get(nome).getName();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")");
    ResultSet rs2;
    rs2 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs2.next();
    int idAtivo = ((Number) rs2.getObject(1)).intValue();
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs.next();
    float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

public void abrirOldCompra (String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet rs3;
    rs3 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs3.next();
    int idAtivo = ((Number) rs3.getObject(1)).intValue();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("UPDATE ativo SET ValorCompra = '" + ask + "', ValorVenda = '" + bid + "' WHERE idAtivo = " + idAtivo);
    ResultSet rs4;
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs4 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs4.next();
    float plafondAnterior = ((Number) rs4.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

```

Figura 4 Métodos abrirNewCompra e abrirOldCompra

```

public void abrirCompra(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        rs = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            abrirNewCompra(nome, montante, ll, lp);
        } else {
            abrirOldCompra(nome, montante, ll, lp);
        }
    } catch(Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figura 5 Método abrirCompra após Refactoring

LONG METHOD (abrirVenda)

```
public void abrirVenda(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        Statement st1 = con.createStatement();
        rs = st1.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            ResultSet rs1;
            rs1 = st.executeQuery("SELECT COUNT(idAtivo) FROM ativo");
            rs1.next();
            int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
            Statement st4 = con.createStatement();
            String fullname = stocks.get(nome).getName();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] temps = valoresMercado.split(", ");
            String temp = temps[0];
            String[] result = temp.split(" ");
            String ask = result[1];
            String[] temps2 = valoresMercado.split(", ");
            String temp2 = temps2[1];
            String[] result2 = temp2.split(" ");
            String bid = result2[1];
            String query2 = "INSERT INTO ativo VALUES (" + nAtivos + ", '" + fullname + "', '" + nome + "', '" + ask + "', '" + bid + "'");
            st4.executeUpdate(query2);
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            Statement st6 = con.createStatement();
            st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA");
            rs = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
            rs.next();
            float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
            float plafondResultante = plafondAnterior - montante;
            st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
        } else {
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
        }
    }
}
```

Figura 6 Método abrirVenda

```

        int idAtivo = ((Number) resultSet.getObject(1)).intValue();
        String valoresMercado = YahooFinance.get(nome).getQuote().toString();
        String[] temps = valoresMercado.split(", ");
        String temp = temps[0];
        String[] result = temp.split(" ");
        String ask = result[1];
        String[] temps2 = valoresMercado.split(", ");
        String temp2 = temps2[1];
        String[] result2 = temp2.split(" ");
        String bid = result2[1];
        Statement s = con.createStatement();
        s.executeUpdate("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
        Statement st6 = con.createStatement();
        ResultSet rs3;
        st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA
        rs3 = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs3.next();
        float plafondAnterior = ((Number) rs3.getObject(1)).floatValue();
        float plafondResultante = plafondAnterior - montante;
        st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
    }
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}

```

Figura 7 Método abrirVenda

REFACTORING

- Extract Method – Maior parte do código foi movido para dois métodos auxiliares novos e criados para calcular resultados e devolvê-los à função abrirVenda. abrirNewVenda, no caso de não haver informação dessa venda na BD e abrirOldVenda, caso haja referência a essa compra na BD.
- Replace Temp with Query – O uso de variáveis temporárias foi resolvido, visto que agora todo o código foi movido para os dois novos métodos. A função abrirVenda apenas usa os valores retornados por esses dois métodos. Por sua vez, para reduzir o número de parâmetros temporários existentes no código de implementação, que passará a encontrar-se em abrirNewVenda e abrirOldVenda, foram usadas as funções getBD e alterBD.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 8 Métodos geBD e alterBD

```

public void abrirNewVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet rs1;
    rs1 = getBD("SELECT COUNT(idAtivo) FROM ativo");
    rs1.next();
    int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
    String fullname = stocks.get(nome).getName();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")");
    ResultSet rs2;
    rs2 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs2.next();
    int idAtivo = ((Number) rs2.getObject(1)).intValue();
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs.next();
    float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

public void abrirOldVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet resultset;
    resultset = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    resultset.next();
    int idAtivo = ((Number) resultset.getObject(1)).intValue();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
    ResultSet rs3;
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs3 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs3.next();
    float plafondAnterior = ((Number) rs3.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

```

Figura 9 Métodos abrirNewVenda e abrirOldVenda

```

public void abrirVenda(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        rs = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            abrirNewVenda(nome, montante, ll, lp);
        } else {
            abrirOldVenda(nome, montante, ll, lp);
        }
    } catch(Exception e) {
        System.out.println("Error: " + e);
    }
}
}

```

Figura 10 Método abrirVenda após Refactoring

LONG METHOD (addAtivo)

```
public void addAtivo(String nome) {
    try {
        Statement st = con.createStatement();
        String query = "SELECT Tipo FROM ativo WHERE Tipo = '" + nome + "'";
        ResultSet rst = st.executeQuery(query);
        if(rst.next() == false) {
            ResultSet rs;
            rs = st.executeQuery("SELECT COUNT(idAtivo) FROM ativo");
            rs.next();
            int nAtivos = ((Number) rs.getObject(1)).intValue() + 1;
            Statement st4 = con.createStatement();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] tempQuote = valoresMercado.split(", ");
            String tempQuotes = tempQuote[1];
            String[] resultQuote = tempQuotes.split(" ");
            String bid = resultQuote[1];
            String[] tempQuote2 = valoresMercado.split(", ");
            String tempQuotes2 = tempQuote2[0];
            String[] resultQuote2 = tempQuotes2.split(" ");
            String ask = resultQuote2[1];
            String fullname = stocks.get(nome).getName();
            String query2 = "INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")";
            st4.executeUpdate(query2);
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            Statement st6 = con.createStatement();
            st6.executeUpdate("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin + ")");
        } else {
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] tempQuote = valoresMercado.split(", ");
            String tempQuotes = tempQuote[1];
            String[] resultQuote = tempQuotes.split(" ");
```

Figura 11 Método addAtivo

```

        String bid = resultQuote[1];
        String[] tempQuote2 = valoresMercado.split(", ");
        String tempQuotes2 = tempQuote2[0];
        String[] resultQuote2 = tempQuotes2.split(" ");
        String ask = resultQuote2[1];
        Statement s = con.createStatement();
        s.executeUpdate("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
        Statement st6 = con.createStatement();
        st6.executeUpdate("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin + ")");
    }
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}

```

Figura 12 Método addAtivo

REFACTORING

- Extract Method – Maior parte do código foi movido para dois métodos auxiliares novos e criados para calcular resultados e devolvê-los à função addAtivo. addNewAtivo, no caso de não haver informação desse ativo na BD e addNewAtivo, caso haja referência a essa compra na BD.
- Replace Temp with Query – O uso de variáveis temporárias foi resolvido, visto que agora todo o código foi movido para os dois novos métodos. A função addAtivo apenas usa os valores retornados por esses dois métodos. Por sua vez, para reduzir o número de parâmetros temporários existentes no código de implementação, que passará a encontrar-se em addNewAtivo e addOldAtivo, foram usadas as funções getBD e alterBD.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 13 Métodos getBD e alterBD

```

public void addNewAtivo(String nome) throws SQLException, IOException {
    ResultSet rs = getBD("SELECT COUNT(idAtivo) FROM ativo");
    rs.next();
    int nAtivos = ((Number) rs.getObject(1)).intValue() + 1;
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    String fullname = stocks.get(nome).getName();
    alterBD("INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")");
    ResultSet rst2 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rst2.next();
    int idAtivo = ((Number) rst2.getObject(1)).intValue();
    alterBD("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin +
}

public void addOldAtivo(String nome) throws SQLException, IOException {
    ResultSet rs = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs.next();
    int idAtivo = ((Number) rs.getObject(1)).intValue();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
    alterBD("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin +
}

```

Figura 14 Métodos addNewAtivo e addOldAtivo

```
public void addAtivo(String nome) {  
    try {  
        ResultSet rst = getBD("SELECT Tipo FROM ativo WHERE Tipo = '" + nome + "'");  
        if(rst.next() == false) {  
            addNewAtivo(nome);  
        } else {  
            addOldAtivo(nome);  
        }  
    } catch(Exception e) {  
        System.out.println("Error: " + e);  
    }  
}
```

Figura 15 Método addAtivo após Refactoring

LONG METHOD (fechaVenda)

```
public void fechaVenda(int idAtivo) {
    try {
        Statement st = con.createStatement();
        ResultSet rs;
        rs = st.executeQuery("SELECT ValorVenda FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorVenda = rs.getFloat(1);
        Statement st2 = con.createStatement();
        ResultSet rs2;
        rs2 = st2.executeQuery("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + getUserLogin());
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        Statement st3 = con.createStatement();
        ResultSet rs3;
        rs3 = st3.executeQuery("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String quote = YahooFinance.get(tipo).getQuote().toString();
        String[] tempQuote = quote.split(", ");
        String tempQuotes = tempQuote[1];
        String[] resultQuote = tempQuotes.split(" ");
        String tempResultQuote = resultQuote[1];
        float priceAtual = Float.parseFloat(tempResultQuote);
        float quantiaInicial = montanteInvestido * valorVenda;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        Statement st4 = con.createStatement();
        ResultSet rs4;
        rs4 = st4.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float lucros = addPlafond + montanteInvestido;
        float setPlafond = plafondAtual + lucros;
        Statement st5 = con.createStatement();
        st5.executeUpdate("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        Statement st6 = con.createStatement();
        st6.executeUpdate("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    }
}
```

Figura 16 Método fechaVenda

REFACTORING

- Extract Method – As variáveis temporárias de onde se extraíam os preços e outros dados acerca do ativo, foram substituídas. Passa-se agora a usar uma variável apenas para guardar uma dada informação, e uma chamada da função própria para o efeito.
- Replace Temp with Query – O uso de variáveis temporárias (sobretudo Statements e ResultSet, para acesso à BD) foi resolvido, visto que agora todo o código foi movido para os dois novos métodos:
 - i. getBD – Que de cada vez que é chamada cria um Statement e um ResultSet para executar a query e guardar os seus resultados, respetivamente.
 - ii. alterBD – Que cria o Statement necessário para que seja executada a Query de alteração da informação na BD.

Com a adição destes dois novos métodos é resolvido o problema de haver muitas, se calhar até demasiadas, declarações de Statements e ResultSet que se mostrarão inúteis, numa só implementação de um método.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 17 Métodos getBD e alterBD

```

public void fechaVenda(int idAtivo) {
    try {
        ResultSet rs;
        rs = getBD("SELECT ValorVenda FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorVenda = rs.getFloat(1);
        ResultSet rs2;
        rs2 = getBD("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + idUtilizador);
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        ResultSet rs3;
        rs3 = getBD("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String bid = YahooFinance.get(tipo).getQuote().getBid().toString();
        float priceAtual = Float.parseFloat(bid);
        float quantiaInicial = montanteInvestido * valorVenda;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        ResultSet rs4;
        rs4 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float lucros = addPlafond + montanteInvestido;
        float setPlafond = plafondAtual + lucros;
        alterBD("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        alterBD("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figura 18 Método fechaVenda após Refactoring

LONG METHOD (fechaCompra)

```
public void fechaCompra(int idAtivo) {
    try {
        Statement st = con.createStatement();
        ResultSet rs;
        rs = st.executeQuery("SELECT ValorCompra FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorCompra = rs.getFloat(1);
        Statement st2 = con.createStatement();
        ResultSet rs2;
        rs2 = st2.executeQuery("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + getUserLogin());
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        Statement st3 = con.createStatement();
        ResultSet rs3;
        rs3 = st3.executeQuery("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String quote = YahooFinance.get(tipo).getQuote().toString();
        String[] tempQuote = quote.split(", ");
        String tempQuotes = tempQuote[0];
        String[] resultQuote = tempQuotes.split(" ");
        String tempResultQuote = resultQuote[1];
        float priceAtual = Float.parseFloat(tempResultQuote);
        float quantiaInicial = montanteInvestido * valorCompra;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        float lucros = addPlafond + montanteInvestido;
        Statement st4 = con.createStatement();
        ResultSet rs4;
        rs4 = st4.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float setPlafond = plafondAtual + lucros;
        Statement st5 = con.createStatement();
        st5.executeUpdate("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        Statement st6 = con.createStatement();
        st6.executeUpdate("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    }
}
```

Figura 19 Método fechaCompra

REFACTORING

- Extract Method – As variáveis temporárias de onde se extraíam os preços e outros dados acerca do ativo, foram substituídas. Passa-se agora a usar uma variável apenas para guardar uma dada informação, e uma chamada da função própria para o efeito.
- Replace Temp with Query – O uso de variáveis temporárias (sobretudo Statements e ResultSet, para acesso à BD) foi resolvido, visto que agora todo o código foi movido para os dois novos métodos:
 - iii. getBD – Que de cada vez que é chamada cria um Statement e um ResultSet para executar a query e guardar os seus resultados, respetivamente.
 - iv. alterBD – Que cria o Statement necessário para que seja executada a Query de alteração da informação na BD.

Com a adição destes dois novos métodos é resolvido o problema de haver muitas, se calhar até demasiadas, declarações de Statements e ResultSet que se mostrarão inúteis, numa só implementação de um método.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 20 Métodos getBD e alterBD

```

public void fechaVenda(int idAtivo) {
    try {
        ResultSet rs;
        rs = getBD("SELECT ValorVenda FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorVenda = rs.getFloat(1);
        ResultSet rs2;
        rs2 = getBD("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + idUtilizador);
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        ResultSet rs3;
        rs3 = getBD("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String bid = YahooFinance.get(tipo).getQuote().getBid().toString();
        float priceAtual = Float.parseFloat(bid);
        float quantiaInicial = montanteInvestido * valorVenda;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        ResultSet rs4;
        rs4 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float lucros = addPlafond + montanteInvestido;
        float setPlafond = plafondAtual + lucros;
        alterBD("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        alterBD("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figura 21 Método fechaCompra após Refactoring

LONG METHOD (getAtivos)

```
public String getAtivos() throws SQLException, IOException {
    StringBuilder sb = new StringBuilder();
    for(String s : stocks.keySet()) {
        String key = s;
        String name = stocks.get(s).getName();
        Stock st = YahooFinance.get(s);
        String quote = st.getQuote().toString();
        String[] tempQuote = quote.split(", ");
        String tempQuotes = tempQuote[1];
        String[] resultQuote = tempQuotes.split(" ");
        String bid = resultQuote[1];
        String[] tempQuote2 = quote.split(", ");
        String tempQuotes2 = tempQuote2[0];
        String[] resultQuote2 = tempQuotes2.split(" ");
        String ask = resultQuote2[1];
        String[] tempQuote3 = quote.split(", ");
        String tempQuotes3 = tempQuote3[2];
        String[] resultQuote3 = tempQuotes3.split(" ");
        String price = resultQuote3[1];
        String prev = st.getQuote().getChangeInPercent().toString();
        sb.append(key + "::" + name + "::" + ask + "::" + bid + "::" + price + "::" + prev + "\n");
    }
    return sb.toString();
}
```

Figura 22 Método getAtivos

REFACTORING

- Result Temp with Query – Em vez de se usarem inúmeras variáveis temporárias para fazer o parsing de cada um dos campos significativos, como por exemplo, valor de venda ou valor de compra, apenas se usa uma variável para guardar cada um desses dados, e recorre-se a uma função criada para o efeito que devolve de forma imediata e direto a mesma.

```
public String getAtivos() throws SQLException, IOException {
    StringBuilder sb = new StringBuilder();
    for(String s : stocks.keySet()) {
        String key = s;
        String name = stocks.get(s).getName();
        Stock st = YahooFinance.get(s);
        String ask = st.getQuote().getAsk().toString();
        String bid = st.getQuote().getBid().toString();
        String price = st.getQuote().getPrice().toString();
        String prev = st.getQuote().getChangeInPercent().toString();
        sb.append(key + "::" + name + "::" + ask + "::" + bid + "::" + price + "::" + prev + "\n");
    }
    return sb.toString();
}
```

Figura 23 Método getAtivos após Refactoring

LARGE CLASS (AppESS)

Por motivos de apresentação o conteúdo da classe AppESS irá ser colocado como Anexo deste relatório. É importante apenas dizer que esta foi uma classe identificada com um Code Smell de Large Class pois excedia em muito o número de linhas de códigos aconselháveis.

REFACTORING

- Extract Class – A classe AppESS continha todos os métodos usados para implementar os requisitos da aplicação, algo que tornava esta classe demasiado extensa. A solução encontrada foi criar duas novas classes: Ativo e Negócio, onde são colocados os métodos que dizem respeito a transações relativas aos ativos e aos negócios, como abertura de compra, fecho de venda, respetivamente.
- Extract Interface – Foi criada uma interface para cada uma das classes com os nomes Mercado e Negociação, para tornar mais legível para alguém novo à implementação, quais os métodos envolvidos.

ANTES

Nome	Nº de métodos	Nº de linhas
AppESS	35	670

DEPOIS

Nome	Nº de métodos	Nº de linhas
AppESS	12	221
Negocio	14	250
Ativo	9	103

As classes Negocio e Ativo irão ser apresentadas nos anexos.

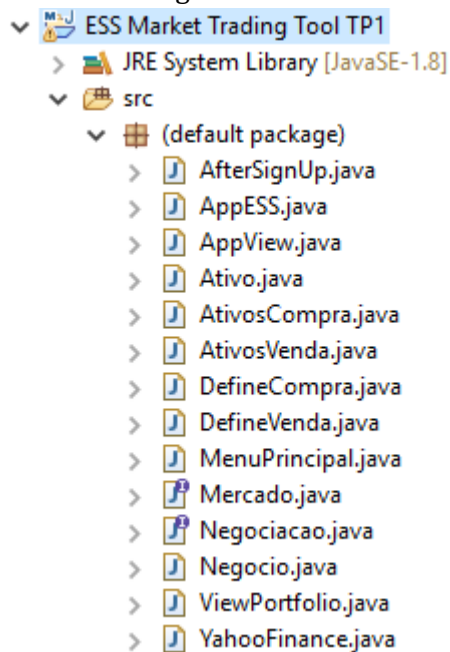


Figura 24 Esquema de classes da Aplicação

```

public interface Negociacao {
    public float getPlafond();
    public void abrirNewCompra (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirOldCompra (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirCompra(String nome, float montante, float ll, float lp);
    public void abrirNewVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirOldVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirVenda(String nome, float montante, float ll, float lp);
    public void fechaCompra(int idAtivo);
    public void fechaVenda(int idAtivo);
    public void addFundos(float fundos);
    public float getPlafondInvestido();
    public float getPlafondInvestidoModalidade(String modalidade);
}

```

Figura 25 Interface Negociacao

```

public interface Mercado {
    public void addNewAtivo(String nome) throws SQLException, IOException;
    public void addOldAtivo(String nome) throws SQLException, IOException;
    public void addAtivo(String nome);
    public void removeAtivo(int id);
    public ResultSet getPortfolio();
    public ResultSet getAtivosCompra();
    public ResultSet getAtivosVenda();
}

```

Figura 26 Interface Mercado

LAZY CLASS

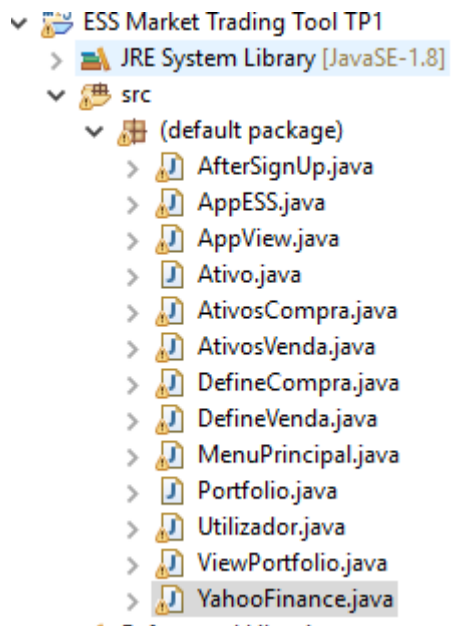


Figura 27 Esquema de classes

REFACTORING

- Inline Class – As classes Portfolio, Utilizador e Ativo, não adicionavam nada de útil à aplicação, deste modo a solução encontrada, foi eliminá-las por inteiro.

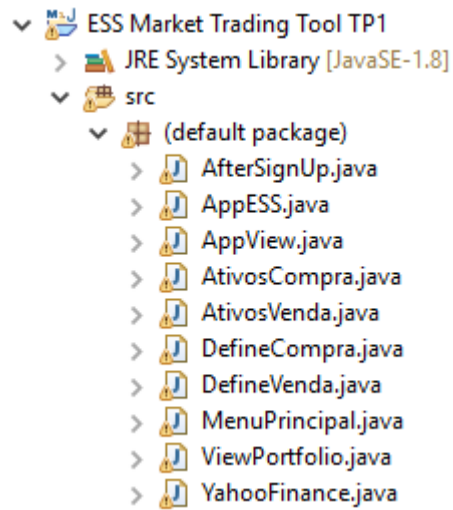


Figura 28 Esquema de classes após Refactoring

DATA CLASS

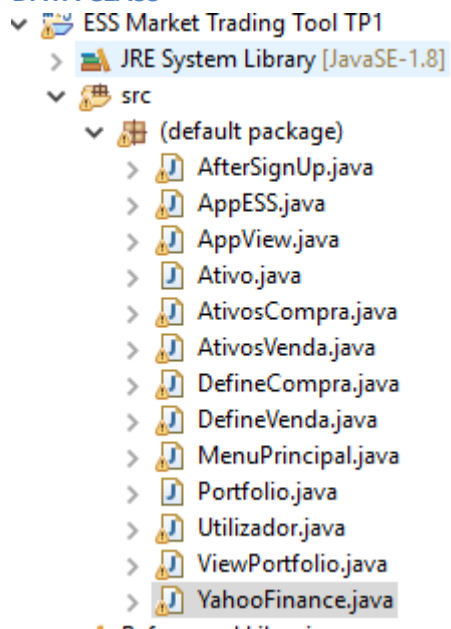


Figura 29 Esquema de classes

REFACTORING

- Inline Class – As classes Ativo, Utilizador e Portfolio, apenas tinham como conteúdo os métodos getters e setters, para acesso aos seus próprios campos. Deste modo a solução passou pela sua eliminação por inteiro.

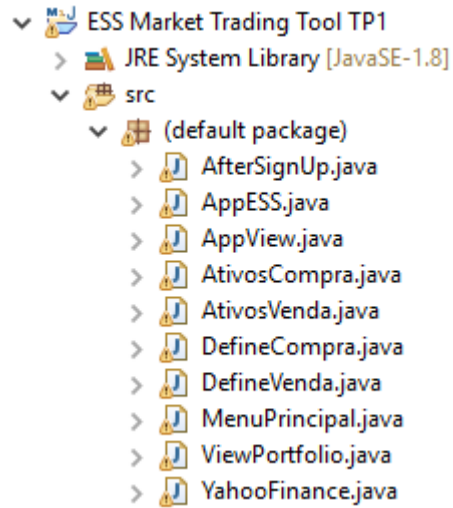


Figura 30 Esquema de classes após Refactoring

SPECULATIVE GENERALITY

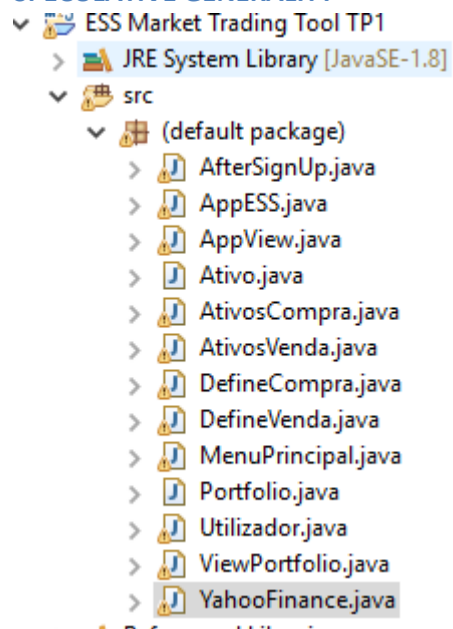


Figura 31 Esquema de classes

REFACTORING

- Inline Class – Inicialmente estaria previsto o recurso às classes Portfolio, Utilizador e Ativo, no entanto, à medida que o projeto foi prosseguindo, devido a tomadas de decisão diferentes às inicialmente previstas estas classes passaram a ser inúteis. A solução passou pela sua eliminação por inteiro.

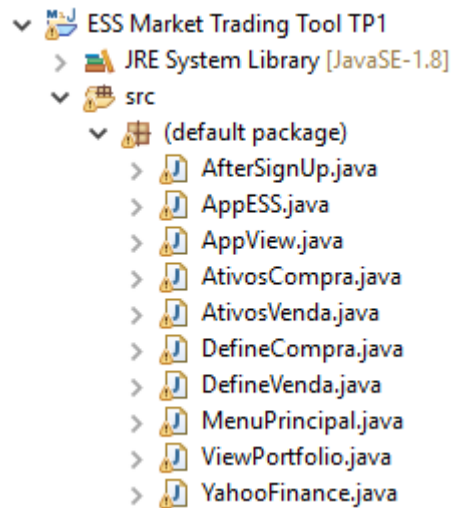


Figura 32 Esquema de classes após Refactoring

DEAD CODE

```
1  import java.awt.EventQueue;
2
3  import javax.swing.JFrame;
4  import javax.swing.JOptionPane;
5  import java.math.BigDecimal;
6  import java.sql.SQLException;
7
8  import yahoofinance.Stock;
9  import java.awt.FlowLayout;
10 import javax.swing.JButton;
11 import java.awt.event.ActionListener;
12 import java.io.IOException;
13 import java.awt.event.ActionEvent;
14 import javax.swing.JLabel;
15 import javax.swing.SwingConstants;
16
17 import java.awt.Font;
18 import javax.swing.ImageIcon;
19 import javax.swing.JTextField;
20 import javax.swing.JPasswordField;
21 import javax.swing.JSeparator;
22 import java.awt.Color;
23
```

Figura 33 Warnings da classe AppView

REFACTORING

- IDE – Inúmeros casos de Dead Code foram encontrados ao longo da fase de desenvolvimento da aplicação, não só na implementação dos métodos, mas também no que diz respeito a bibliotecas importadas, que depois passaram a um estado obsoleto. A solução acabou por passar pelo uso de um IDE como o Eclipse, que avisa o programador de tais Code Smells e ajuda o mesmo a corrigi-los. O conjunto completo de Warnings encontrar-se-á em anexo, o exemplo seguinte trata-se de uma resolução dos Code Smells da classe AppView.

```
1 import java.awt.EventQueue;
2
3 import javax.swing.JFrame;
4 import javax.swing.JOptionPane;
5
6 import javax.swing.JButton;
7 import java.awt.event.ActionListener;
8 import java.io.IOException;
9 import java.awt.event.ActionEvent;
10 import javax.swing.JLabel;
11 import javax.swing.SwingConstants;
12
13 import java.awt.Font;
14 import javax.swing.ImageIcon;
15 import javax.swing.JTextField;
16 import javax.swing.JPasswordField;
17 import javax.swing.JSeparator;
18 import java.awt.Color;
19
20 public class AppView {
21
22     private JFrame frame;
23     private static JTextField textField;
24     private static JPasswordField passwordField;
25 }
```

Figura 34 Classe AppView após Refactoring

INNAPPROPRIATE INTIMACY

```
JLabel lblNewLabel_1 = new JLabel("New label");
lblNewLabel_1.setFont(new Font("Tahoma", Font.PLAIN, 18));
lblNewLabel_1.setBounds(10, 30, 290, 42);
contentPane.add(lblNewLabel_1);
lblNewLabel_1.setText("Montante Investido: " + n.getPlafondInvestidoModalidade(connect.getUserLogin(), "COMPRA") + "€");

JButton btnNewButton = new JButton("Fechar Compra");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        int id = Integer.parseInt(textField.getText());
        n.fechaCompra(id, connect.getUserLogin());
        table.setModel(connect.resultSetToTableModel(a.getAtivosCompra(connect.getUserLogin())));
        textField.setText(null);
        lblNewLabel_1.setText("Montante Investido: " + n.getPlafondInvestidoModalidade(connect.getUserLogin(), "COMPRA")
    }
}
```

Figura 35 Innappropriate Intimacy em AtivosCompra

```
table = new JTable();
scrollPane.setViewportViewView(table);
table.setModel(connect.resultSetToTableModel(a.getPortfolio(connect.getUserLogin())));
```

Figura 36 Innappropriate Intimacy em ViewPortfolio

REFACTORING

- Move Field – Foram identificados inúmeros casos de Innapropriate Intimacy. Todos pelo mesmo motivo, a necessidade de aceder a um campo que identifica o id do utilizador autenticado no momento da execução dos métodos que dele necessitam e que se encontra na classe AppESS. Ora, os métodos que deste campo necessitam, estão todos eles em classes que não AppESS. A solução foi criar um campo userLogin em cada um dessas classes (Negocio e Ativo) e aceder ao mesmo localmente.

```
public class Negocio implements Negociacao {  
  
    private ResultSet rs;  
    private Connection con;  
    private int userLogin;
```

Figura 37 Campo userLogin em Negocio

```
public class Ativo implements Mercado {  
  
    private ResultSet rs;  
    private Connection con;  
    private int userLogin;
```

Figura 38 Campo userLogin em Ativo

```
public ViewPortfolio() throws ClassNotFoundException, SQLException {  
    AppESS connect = new AppESS();  
    Ativo a = new Ativo();  
    a.setLogin(connect.getUserLogin());
```

Figura 39 Definir userLogin

```
JLabel lblNewLabel_1 = new JLabel("New label");  
lblNewLabel_1.setFont(new Font("Tahoma", Font.PLAIN, 18));  
lblNewLabel_1.setBounds(10, 30, 290, 42);  
contentPane.add(lblNewLabel_1);  
lblNewLabel_1.setText("Montante Investido: " + n.getPlafondInvestidoModalidade("COMPRA") + "€");  
  
JButton btnNewButton = new JButton("Fechar Compra");  
btnNewButton.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        int id = Integer.parseInt(textField.getText());  
        n.fechaCompra(id);  
        table.setModel(connect.resultSetToTableModel(a.getAtivosCompra()));  
        textField.setText(null);  
        lblNewLabel_1.setText("Montante Investido: " + n.getPlafondInvestidoModalidade("COMPRA") + "€");  
    }  
});
```

Figura 40 Métodos sem acesso a um campo de outra classe

REFACTORING DA SEGUNDA VERSÃO DO TRABALHO PRÁTICO

De seguida, passo a apresentar os excertos de código que mostram a presença dos code smells acima referidos, seguidos dos excertos de código resultantes da aplicação de uma técnica de resolução desse mesmo code smell.

LONG METHOD (abrirCompra)

```
public void abrirCompra(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        Statement st1 = con.createStatement();
        rs = st1.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            ResultSet rs1;
            rs1 = st.executeQuery("SELECT COUNT(idAtivo) FROM ativo");
            rs1.next();
            int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
            Statement st4 = con.createStatement();
            String fullname = stocks.get(nome).getName();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] tempQuote = valoresMercado.split(", ");
            String tempQuotes = tempQuote[1];
            String[] resultQuote = tempQuotes.split(" ");
            String bid = resultQuote[1];
            String[] tempQuote2 = valoresMercado.split(", ");
            String tempQuotes2 = tempQuote2[0];
            String[] resultQuote2 = tempQuotes2.split(" ");
            String ask = resultQuote2[1];
            String query2 = "INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")";
            st4.executeUpdate(query2);
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            Statement st6 = con.createStatement();
            st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA");
            rs = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
            rs.next();
            float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
            float plafondResultante = plafondAnterior - montante;
            st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
        } else {
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
        }
    }
}
```

Figura 41 Método abrirCompra

```

        int idAtivo = ((Number) resultset.getObject(1)).intValue();
        String valoresMercado = YahooFinance.get(nome).getQuote().toString();
        String[] tempQuote = valoresMercado.split(", ");
        String tempQuotes = tempQuote[1];
        String[] resultQuote = tempQuotes.split(" ");
        String bid = resultQuote[1];
        String[] tempQuote2 = valoresMercado.split(", ");
        String tempQuotes2 = tempQuote2[0];
        String[] resultQuote2 = tempQuotes2.split(" ");
        String ask = resultQuote2[1];
        Statement s = con.createStatement();
        s.executeUpdate("UPDATE ativo SET ValorCompra = '" + ask + "', ValorVenda = '" + bid + "' WHERE idAtivo = " + idAtivo);
        Statement st6 = con.createStatement();
        ResultSet rs3;
        st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA
        rs3 = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs3.next();
        float plafondAnterior = ((Number) rs3.getObject(1)).floatValue();
        float plafondResultante = plafondAnterior - montante;
        st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
    }
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}

```

Figura 42 Método abrirCompra

REFACTORING

- Extract Method – Maior parte do código foi movido para dois métodos auxiliares novos e criados para calcular resultados e devolvê-los à função abrirCompra. abrirNewCompra, no caso de não haver informação dessa compra na BD e abrirOldCompra, caso haja referência a essa compra na BD.
- Replace Temp with Query – O uso de variáveis temporárias foi resolvido, visto que agora todo o código foi movido para os dois novos métodos. A função abrirCompra apenas usa os valores retornados por esses dois métodos. Por sua vez, para reduzir o número de parâmetros temporários existentes no código de implementação, que passará a encontrar-se em abrirNewCompra e abrirOldCompra, foram usadas as funções getBD e alterBD.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 43 Métodos getBD e alterBD

```

public void abrirNewCompra(String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet rs1;
    rs1 = getBD("SELECT COUNT(idAtivo) FROM ativo");
    rs1.next();
    int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
    String fullname = stocks.get(nome).getName();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")");
    ResultSet rs2;
    rs2 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs2.next();
    int idAtivo = ((Number) rs2.getObject(1)).intValue();
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs.next();
    float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

public void abrirOldCompra (String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet rs3;
    rs3 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs3.next();
    int idAtivo = ((Number) rs3.getObject(1)).intValue();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("UPDATE ativo SET ValorCompra = '" + ask + "', ValorVenda = '" + bid + "' WHERE idAtivo = " + idAtivo);
    ResultSet rs4;
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs4 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs4.next();
    float plafondAnterior = ((Number) rs4.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

```

Figura 44 Métodos abrirNewCompra e abrirOldCompra

```

public void abrirCompra(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        rs = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            abrirNewCompra(nome, montante, ll, lp);
        } else {
            abrirOldCompra(nome, montante, ll, lp);
        }
    } catch(Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figura 45 Método abrirCompra após Refactoring

LONG METHOD (abrirVenda)

```
public void abrirVenda(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        Statement st1 = con.createStatement();
        rs = st1.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            ResultSet rs1;
            rs1 = st.executeQuery("SELECT COUNT(idAtivo) FROM ativo");
            rs1.next();
            int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
            Statement st4 = con.createStatement();
            String fullname = stocks.get(nome).getName();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] temps = valoresMercado.split(", ");
            String temp = temps[0];
            String[] result = temp.split(" ");
            String ask = result[1];
            String[] temps2 = valoresMercado.split(", ");
            String temp2 = temps2[1];
            String[] result2 = temp2.split(" ");
            String bid = result2[1];
            String query2 = "INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "','" + nome + "','" + ask + "','" + bid + "')";
            st4.executeUpdate(query2);
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            Statement st6 = con.createStatement();
            st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA");
            rs = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
            rs.next();
            float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
            float plafondResultante = plafondAnterior - montante;
            st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
        } else {
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
        }
    }
}
```

Figura 46 Método abrirVenda

```

        int idAtivo = ((Number) resultSet.getObject(1)).intValue();
        String valoresMercado = YahooFinance.get(nome).getQuote().toString();
        String[] temps = valoresMercado.split(", ");
        String temp = temps[0];
        String[] result = temp.split(" ");
        String ask = result[1];
        String[] temps2 = valoresMercado.split(", ");
        String temp2 = temps2[1];
        String[] result2 = temp2.split(" ");
        String bid = result2[1];
        Statement s = con.createStatement();
        s.executeUpdate("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
        Statement st6 = con.createStatement();
        ResultSet rs3;
        st6.executeUpdate("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, LimiteLucro, LimitePerda) VA
        rs3 = st6.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs3.next();
        float plafondAnterior = ((Number) rs3.getObject(1)).floatValue();
        float plafondResultante = plafondAnterior - montante;
        st6.executeUpdate("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
    }
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}

```

Figura 47 Método abrirVenda

REFACTORING

- Extract Method – Maior parte do código foi movido para dois métodos auxiliares novos e criados para calcular resultados e devolvê-los à função abrirVenda. abrirNewVenda, no caso de não haver informação dessa venda na BD e abrirOldVenda, caso haja referência a essa compra na BD.
- Replace Temp with Query – O uso de variáveis temporárias foi resolvido, visto que agora todo o código foi movido para os dois novos métodos. A função abrirVenda apenas usa os valores retornados por esses dois métodos. Por sua vez, para reduzir o número de parâmetros temporários existentes no código de implementação, que passará a encontrar-se em abrirNewVenda e abrirOldVenda, foram usadas as funções getBD e alterBD.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 48 Métodos getBD e alterBD

```

public void abrirNewVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet rs1;
    rs1 = getBD("SELECT COUNT(idAtivo) FROM ativo");
    rs1.next();
    int nAtivos = ((Number) rs1.getObject(1)).intValue() + 1;
    String fullname = stocks.get(nome).getName();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")");
    ResultSet rs2;
    rs2 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs2.next();
    int idAtivo = ((Number) rs2.getObject(1)).intValue();
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs.next();
    float plafondAnterior = ((Number) rs.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

public void abrirOldVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException {
    ResultSet resultset;
    resultset = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    resultset.next();
    int idAtivo = ((Number) resultset.getObject(1)).intValue();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
    ResultSet rs3;
    alterBD("INSERT INTO utilizador_has_ativo(Utilizador_idUtilizador, Ativo_idAtivo, MontanteResultante, Estado, Montante, Lin
rs3 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
    rs3.next();
    float plafondAnterior = ((Number) rs3.getObject(1)).floatValue();
    float plafondResultante = plafondAnterior - montante;
    alterBD("UPDATE utilizador SET Plafond = " + plafondResultante + " WHERE idUtilizador = " + getUserLogin());
}

```

Figura 49 Métodos abrirNewVenda e abrirOldVenda

```

public void abrirVenda(String nome, float montante, float ll, float lp) {
    try {
        ResultSet rs;
        rs = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
        if(rs.next() == false) {
            abrirNewVenda(nome, montante, ll, lp);
        } else {
            abrirOldVenda(nome, montante, ll, lp);
        }
    } catch(Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figura 50 Método abrirVenda após Refactoring

LONG METHOD (addAtivo)

```
public void addAtivo(String nome) {
    try {
        Statement st = con.createStatement();
        String query = "SELECT Tipo FROM ativo WHERE Tipo = '" + nome + "'";
        ResultSet rst = st.executeQuery(query);
        if(rst.next() == false) {
            ResultSet rs;
            rs = st.executeQuery("SELECT COUNT(idAtivo) FROM ativo");
            rs.next();
            int nAtivos = ((Number) rs.getObject(1)).intValue() + 1;
            Statement st4 = con.createStatement();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] tempQuote = valoresMercado.split(", ");
            String tempQuotes = tempQuote[1];
            String[] resultQuote = tempQuotes.split(" ");
            String bid = resultQuote[1];
            String[] tempQuote2 = valoresMercado.split(", ");
            String tempQuotes2 = tempQuote2[0];
            String[] resultQuote2 = tempQuotes2.split(" ");
            String ask = resultQuote2[1];
            String fullname = stocks.get(nome).getName();
            String query2 = "INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")";
            st4.executeUpdate(query2);
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            Statement st6 = con.createStatement();
            st6.executeUpdate("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin + ")");
        } else {
            ResultSet resultset;
            resultset = st.executeQuery("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
            resultset.next();
            int idAtivo = ((Number) resultset.getObject(1)).intValue();
            String valoresMercado = YahooFinance.get(nome).getQuote().toString();
            String[] tempQuote = valoresMercado.split(", ");
            String tempQuotes = tempQuote[1];
            String[] resultQuote = tempQuotes.split(" ");
```

Figura 51 Método addAtivo

```

        String bid = resultQuote[1];
        String[] tempQuote2 = valoresMercado.split(", ");
        String tempQuotes2 = tempQuote2[0];
        String[] resultQuote2 = tempQuotes2.split(" ");
        String ask = resultQuote2[1];
        Statement s = con.createStatement();
        s.executeUpdate("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
        Statement st6 = con.createStatement();
        st6.executeUpdate("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin + ")");
    }
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}

```

Figura 52 Método addAtivo

REFACTORING

- Extract Method – Maior parte do código foi movido para dois métodos auxiliares novos e criados para calcular resultados e devolvê-los à função addAtivo. addNewAtivo, no caso de não haver informação desse ativo na BD e addNewAtivo, caso haja referência a essa compra na BD.
- Replace Temp with Query – O uso de variáveis temporárias foi resolvido, visto que agora todo o código foi movido para os dois novos métodos. A função addAtivo apenas usa os valores retornados por esses dois métodos. Por sua vez, para reduzir o número de parâmetros temporários existentes no código de implementação, que passará a encontrar-se em addNewAtivo e addOldAtivo, foram usadas as funções getBD e alterBD.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 53 Métodos getBD e alterBD


```

public void addNewAtivo(String nome) throws SQLException, IOException {
    ResultSet rs = getBD("SELECT COUNT(idAtivo) FROM ativo");
    rs.next();
    int nAtivos = ((Number) rs.getObject(1)).intValue() + 1;
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    String fullname = stocks.get(nome).getName();
    alterBD("INSERT INTO ativo VALUES (" + nAtivos + "," + fullname + "," + nome + "," + ask + "," + bid + ")");
    ResultSet rst2 = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rst2.next();
    int idAtivo = ((Number) rst2.getObject(1)).intValue();
    alterBD("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin +
}

public void addOldAtivo(String nome) throws SQLException, IOException {
    ResultSet rs = getBD("SELECT idAtivo FROM ativo WHERE Tipo = '" + nome + "'");
    rs.next();
    int idAtivo = ((Number) rs.getObject(1)).intValue();
    String bid = YahooFinance.get(nome).getQuote().getBid().toString();
    String ask = YahooFinance.get(nome).getQuote().getAsk().toString();
    alterBD("UPDATE ativo SET ValorVenda = '" + bid + "', ValorCompra = '" + ask + "' WHERE idAtivo = " + idAtivo);
    alterBD("INSERT INTO ativo_has_portfolio(Ativo_idAtivo,Portfolio_idPortfolio) " + "VALUES (" + idAtivo + "," + userLogin +
}

```

Figura 54 Métodos addNewAtivo e addOldAtivo

```
public void addAtivo(String nome) {  
    try {  
        ResultSet rst = getBD("SELECT Tipo FROM ativo WHERE Tipo = '" + nome + "'");  
        if(rst.next() == false) {  
            addNewAtivo(nome);  
        } else {  
            addOldAtivo(nome);  
        }  
    } catch(Exception e) {  
        System.out.println("Error: " + e);  
    }  
}
```

Figura 55 Método addAtivo após Refactoring

LONG METHOD (fechaVenda)

```
public void fechaVenda(int idAtivo) {
    try {
        Statement st = con.createStatement();
        ResultSet rs;
        rs = st.executeQuery("SELECT ValorVenda FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorVenda = rs.getFloat(1);
        Statement st2 = con.createStatement();
        ResultSet rs2;
        rs2 = st2.executeQuery("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + getUserLogin());
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        Statement st3 = con.createStatement();
        ResultSet rs3;
        rs3 = st3.executeQuery("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String quote = YahooFinance.get(tipo).getQuote().toString();
        String[] tempQuote = quote.split(", ");
        String tempQuotes = tempQuote[1];
        String[] resultQuote = tempQuotes.split(" ");
        String tempResultQuote = resultQuote[1];
        float priceAtual = Float.parseFloat(tempResultQuote);
        float quantiaInicial = montanteInvestido * valorVenda;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        Statement st4 = con.createStatement();
        ResultSet rs4;
        rs4 = st4.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float lucros = addPlafond + montanteInvestido;
        float setPlafond = plafondAtual + lucros;
        Statement st5 = con.createStatement();
        st5.executeUpdate("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        Statement st6 = con.createStatement();
        st6.executeUpdate("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    }
}
```

Figura 56 Método fechaVenda

REFACTORING

- Extract Method – As variáveis temporárias de onde se extraíam os preços e outros dados acerca do ativo, foram substituídas. Passa-se agora a usar uma variável apenas para guardar uma dada informação, e uma chamada da função própria para o efeito.
- Replace Temp with Query – O uso de variáveis temporárias (sobretudo Statements e ResultSet, para acesso à BD) foi resolvido, visto que agora todo o código foi movido para os dois novos métodos:
 - v. getBD – Que de cada vez que é chamada cria um Statement e um ResultSet para executar a query e guardar os seus resultados, respetivamente.
 - vi. alterBD – Que cria o Statement necessário para que seja executada a Query de alteração da informação na BD.

Com a adição destes dois novos métodos é resolvido o problema de haver muitas, se calhar até demasiadas, declarações de Statements e ResultSet que se mostrarão inúteis, numa só implementação de um método.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 57 Métodos getBD e alterBD

```

public void fechaVenda(int idAtivo) {
    try {
        ResultSet rs;
        rs = getBD("SELECT ValorVenda FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorVenda = rs.getFloat(1);
        ResultSet rs2;
        rs2 = getBD("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + idUtilizador);
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        ResultSet rs3;
        rs3 = getBD("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String bid = YahooFinance.get(tipo).getQuote().getBid().toString();
        float priceAtual = Float.parseFloat(bid);
        float quantiaInicial = montanteInvestido * valorVenda;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        ResultSet rs4;
        rs4 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float lucros = addPlafond + montanteInvestido;
        float setPlafond = plafondAtual + lucros;
        alterBD("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        alterBD("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figura 58 Método fechaVenda após Refactoring

LONG METHOD (fechaCompra)

```
public void fechaCompra(int idAtivo) {
    try {
        Statement st = con.createStatement();
        ResultSet rs;
        rs = st.executeQuery("SELECT ValorCompra FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorCompra = rs.getFloat(1);
        Statement st2 = con.createStatement();
        ResultSet rs2;
        rs2 = st2.executeQuery("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + getUserLogin());
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        Statement st3 = con.createStatement();
        ResultSet rs3;
        rs3 = st3.executeQuery("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String quote = YahooFinance.get(tipo).getQuote().toString();
        String[] tempQuote = quote.split(", ");
        String tempQuotes = tempQuote[0];
        String[] resultQuote = tempQuotes.split(" ");
        String tempResultQuote = resultQuote[1];
        float priceAtual = Float.parseFloat(tempResultQuote);
        float quantiaInicial = montanteInvestido * valorCompra;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        float lucros = addPlafond + montanteInvestido;
        Statement st4 = con.createStatement();
        ResultSet rs4;
        rs4 = st4.executeQuery("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float setPlafond = plafondAtual + lucros;
        Statement st5 = con.createStatement();
        st5.executeUpdate("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        Statement st6 = con.createStatement();
        st6.executeUpdate("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    }
}
```

Figura 59 Método fechaCompra

REFACTORING

- Extract Method – As variáveis temporárias de onde se extraíam os preços e outros dados acerca do ativo, foram substituídas. Passa-se agora a usar uma variável apenas para guardar uma dada informação, e uma chamada da função própria para o efeito.
- Replace Temp with Query – O uso de variáveis temporárias (sobretudo Statements e ResultSet, para acesso à BD) foi resolvido, visto que agora todo o código foi movido para os dois novos métodos:
 - vii. getBD – Que de cada vez que é chamada cria um Statement e um ResultSet para executar a query e guardar os seus resultados, respetivamente.
 - viii. alterBD – Que cria o Statement necessário para que seja executada a Query de alteração da informação na BD.

Com a adição destes dois novos métodos é resolvido o problema de haver muitas, se calhar até demasiadas, declarações de Statements e ResultSet que se mostrarão inúteis, numa só implementação de um método.

```
public ResultSet getBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery(query);  
    con.close();  
    return rs;  
}  
  
public void alterBD(String query) throws SQLException {  
    Statement st = con.createStatement();  
    st.executeUpdate(query);  
    con.close();  
}
```

Figura 60 Métodos getBD e alterBD

```

public void fechaVenda(int idAtivo) {
    try {
        ResultSet rs;
        rs = getBD("SELECT ValorVenda FROM ativo WHERE idAtivo = " + idAtivo);
        rs.next();
        float valorVenda = rs.getFloat(1);
        ResultSet rs2;
        rs2 = getBD("SELECT Montante FROM utilizador_has_ativo WHERE Ativo_idAtivo = " + idAtivo + " AND Utilizador_idUtilizador = " + idUtilizador);
        rs2.next();
        float montanteInvestido = ((Number) rs2.getObject(1)).floatValue();
        ResultSet rs3;
        rs3 = getBD("SELECT Tipo FROM ativo WHERE idAtivo = " + idAtivo);
        rs3.next();
        String tipo = ((String) rs3.getObject(1)).toString();
        String bid = YahooFinance.get(tipo).getQuote().getBid().toString();
        float priceAtual = Float.parseFloat(bid);
        float quantiaInicial = montanteInvestido * valorVenda;
        float quantiaAtual = montanteInvestido * priceAtual;
        float addPlafond = quantiaAtual - quantiaInicial;
        ResultSet rs4;
        rs4 = getBD("SELECT Plafond FROM utilizador WHERE idUtilizador = " + getUserLogin());
        rs4.next();
        float plafondAtual = ((Number) rs4.getObject(1)).floatValue();
        float lucros = addPlafond + montanteInvestido;
        float setPlafond = plafondAtual + lucros;
        alterBD("UPDATE utilizador SET Plafond = " + setPlafond + " WHERE idUtilizador = " + getUserLogin());
        alterBD("DELETE FROM utilizador_has_ativo WHERE Utilizador_idUtilizador = " + getUserLogin() + " AND Ativo_idAtivo = " + idAtivo);
    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figura 61 Método fechaVenda após Refactoring

LONG METHOD (getAtivos)

```
public String getAtivos() throws SQLException, IOException {
    StringBuilder sb = new StringBuilder();
    for(String s : stocks.keySet()) {
        String key = s;
        String name = stocks.get(s).getName();
        Stock st = YahooFinance.get(s);
        String quote = st.getQuote().toString();
        String[] tempQuote = quote.split(", ");
        String tempQuotes = tempQuote[1];
        String[] resultQuote = tempQuotes.split(" ");
        String bid = resultQuote[1];
        String[] tempQuote2 = quote.split(", ");
        String tempQuotes2 = tempQuote2[0];
        String[] resultQuote2 = tempQuotes2.split(" ");
        String ask = resultQuote2[1];
        String[] tempQuote3 = quote.split(", ");
        String tempQuotes3 = tempQuote3[2];
        String[] resultQuote3 = tempQuotes3.split(" ");
        String price = resultQuote3[1];
        String prev = st.getQuote().getChangeInPercent().toString();
        sb.append(key + "::" + name + "::" + ask + "::" + bid + "::" + price + "::" + prev + "\n");
    }
    return sb.toString();
}
```

Figura 62 Método getAtivos

REFACTORING

- Result Temp with Query – Em vez de se usarem inúmeras variáveis temporárias para fazer o parsing de cada um dos campos significativos, como por exemplo, valor de venda ou valor de compra, apenas se usa uma variável para guardar cada um desses dados, e recorre-se a uma função criada para o efeito que devolve de forma imediata e direto a mesma.

```
public String getAtivos() throws SQLException, IOException {
    StringBuilder sb = new StringBuilder();
    for(String s : stocks.keySet()) {
        String key = s;
        String name = stocks.get(s).getName();
        Stock st = YahooFinance.get(s);
        String ask = st.getQuote().getAsk().toString();
        String bid = st.getQuote().getBid().toString();
        String price = st.getQuote().getPrice().toString();
        String prev = st.getQuote().getChangeInPercent().toString();
        sb.append(key + "::" + name + "::" + ask + "::" + bid + "::" + price + "::" + prev + "\n");
    }
    return sb.toString();
}
```

Figura 63 Método getAtivos após Refactoring

LARGE CLASS (AppESS)

Tal como na primeira versão do trabalho, por motivos de apresentação o conteúdo da classe AppESS irá ser colocado como Anexo deste relatório. É importante apenas dizer que esta foi uma classe identificada com um Code Smell de Large Class pois excedia em muito o número de linhas de códigos aconselháveis. Linhas estas ainda superiores às da primeira versão visto que os métodos implementativos das novas funcionalidades da segunda versão foram sendo acumulados nesta classe.

REFACTORING

- Extract Class – A classe AppESS continha todos os métodos usados para implementar os requisitos da aplicação, algo que tornava esta classe demasiado extensa. A solução encontrada foi criar duas novas classes: Ativo e Negócio, onde são colocados os métodos que dizem respeito a transações relativas aos ativos e aos negócios, como abertura de compra, fecho de venda, respetivamente.
- Extract Interface – Foi criada uma interface para cada uma das classes com os nomes Mercado e Negociação, para tornar mais legível para alguém novo à implementação, quais os métodos envolvidos. Foi ainda criada uma interface, designada BD, com os dois métodos de acesso e alteração à BD (getBD e alterBD).
- ANTES

Nome	Nº de métodos	Nº de linhas
AppESS	38	975

- DEPOIS

Nome	Nº de métodos	Nº de linhas
AppESS	12 + 8	320
Negocio	16	304
Ativo	10	124

As classes Negocio e Ativo irão ser apresentadas nos anexos.

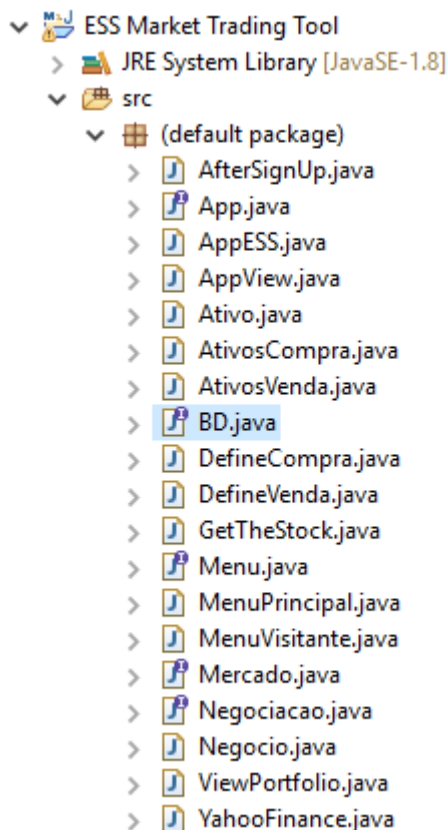


Figura 64 Esquema de classes após Refactoring

```

import java.io.IOException;

public interface Negociacao {
    public float getPlafond();
    public void abrirNewCompra (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirOldCompra (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirCompra(String nome, float montante, float ll, float lp);
    public void abrirNewVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirOldVenda (String nome, float montante, float ll, float lp) throws SQLException, IOException;
    public void abrirVenda(String nome, float montante, float ll, float lp);
    public void fechaCompra(int idAtivo);
    public void fechaVenda(int idAtivo);
    public void addFundos(float fundos);
    public float getPlafondInvestido();
    public float getPlafondInvestidoModalidade(String modalidade);
    public void updateCompra() throws SQLException, IOException;
    public void updateVenda() throws SQLException, IOException;
}

```

Figura 65 Interface Negociação

```

import java.io.IOException;

public interface Mercado {
    public void addNewAtivo(String nome) throws SQLException, IOException;
    public void addOldAtivo(String nome) throws SQLException, IOException;
    public void addAtivo(String nome);
    public void removeAtivo(int id);
    public ResultSet getPortfolio();
    public ResultSet getAtivosCompra();
    public ResultSet getAtivosVenda();
    public void addAlerta(String codigo, String modalidade, float valor);
}

```

Figura 66 Interface Mercado

```
import java.sql.ResultSet;

public interface BD {
    public ResultSet getBD (String query) throws SQLException;
    public void alterBD (String query) throws SQLException;
}
```

Figura 67 Interface BD

LAZY CLASS






















- ▼  ESS Market Trading Tool
 - >  JRE System Library [JavaSE-1.8]
 - ▼  src
 - ▼  (default package)
 - >  AfterSignUp.java
 - >  App.java
 - >  AppESS.java
 - >  AppView.java
 - >  Ativo.java
 - >  AtivosCompra.java
 - >  AtivosVenda.java
 - >  DefineCompra.java
 - >  DefineVenda.java
 - >  GetTheStock.java
 - >  Menu.java
 - >  MenuPrincipal.java
 - >  MenuVisitante.java
 - >  Portfolio.java
 - >  Utilizador.java
 - >  ViewPortfolio.java
 - >  YahooFinance.java

Figura 68 Esquema de classes TP2

REFACTORING

- Inline Class – As classes Portfolio, Utilizador e Ativo, não adicionavam nada de útil à aplicação, deste modo a solução encontrada, foi eliminá-las por inteiro.

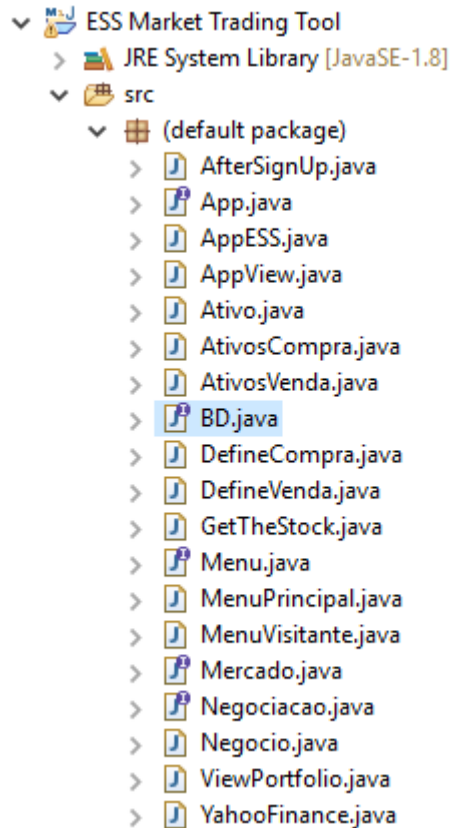


Figura 69 Esquema de classes TP2 após Refactoring

DATA CLASS

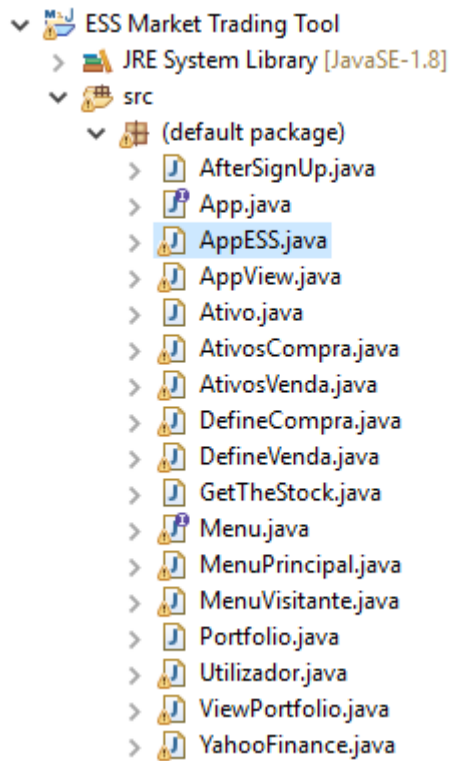


Figura 70 Esquema de classes

REFACTORING

- Inline Class – As classes Ativo, Utilizador e Portfolio, apenas tinham como conteúdo os métodos getters e setters, para acesso aos seus próprios campos. Deste modo a solução passou pela sua eliminação por inteiro.

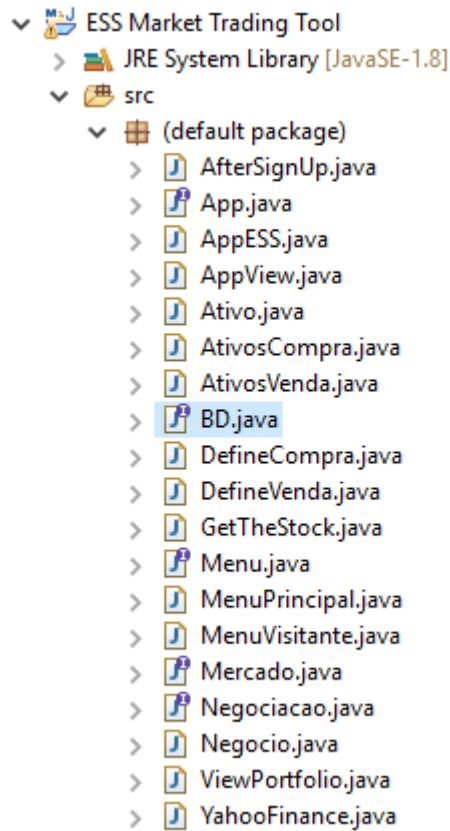


Figura 71 Esquema de classes

SPECULATIVE GENERALITY






















- ▼  ESS Market Trading Tool
 - >  JRE System Library [JavaSE-1.8]
 - ▼  src
 - ▼  (default package)
 - >  AfterSignUp.java
 - >  App.java
 - >  AppESS.java
 - >  AppView.java
 - >  Ativo.java
 - >  AtivosCompra.java
 - >  AtivosVenda.java
 - >  DefineCompra.java
 - >  DefineVenda.java
 - >  GetTheStock.java
 - >  Menu.java
 - >  MenuPrincipal.java
 - >  MenuVisitante.java
 - >  Portfolio.java
 - >  Utilizador.java
 - >  ViewPortfolio.java
 - >  YahooFinance.java

Figura 72 Esquema de classes

REFACTORING

- Inline Class – Inicialmente estaria previsto o recurso às classes Portfolio, Utilizador e Ativo, no entanto, à medida que o projeto foi prosseguindo, devido a tomadas de decisão diferentes às inicialmente previstas estas classes passaram a ser inúteis. A solução passou pela sua eliminação por inteiro.

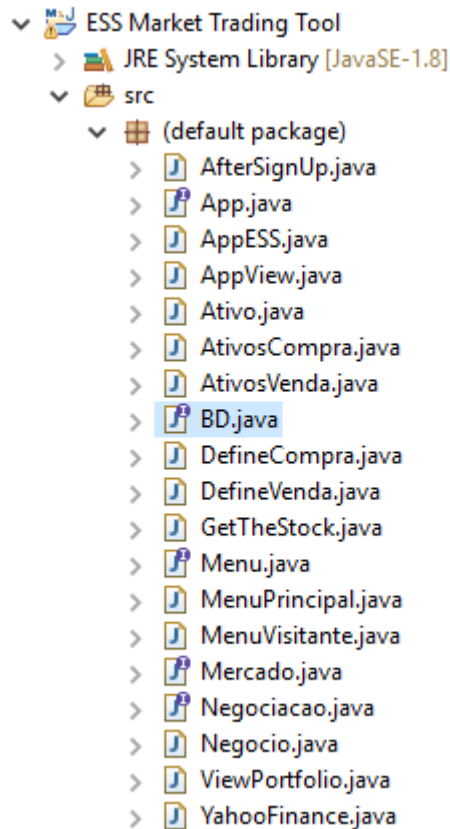


Figura 73 Esquema de classes TP2 após Refactoring

DEAD CODE

```
1 import java.awt.EventQueue;
2
3 import javax.swing.JFrame;
4 import javax.swing.JOptionPane;
5 import java.math.BigDecimal;
6 import java.sql.SQLException;
7
8 import yahoofinance.Stock;
9 import java.awt.FlowLayout;
10 import javax.swing.JButton;
11 import java.awt.event.ActionListener;
12 import java.io.IOException;
13 import java.awt.event.ActionEvent;
14 import javax.swing.JLabel;
15 import javax.swing.SwingConstants;
16
17 import java.awt.Font;
18 import javax.swing.ImageIcon;
19 import javax.swing.JTextField;
20 import javax.swing.JPasswordField;
21 import javax.swing.JSeparator;
22 import java.awt.Color;
23
```

Figura 74 Exemplo de Dead Code em TP2

REFACTORING

- IDE – Inúmeros casos de Dead Code foram encontrados ao longo da fase de desenvolvimento da aplicação, não só na implementação dos métodos, mas também no que diz respeito a bibliotecas importadas, que depois passaram a um estado obsoleto. A solução acabou por passar pelo uso de um IDE como o Eclipse, que avisa o programador de tais Code Smells e ajuda o mesmo a corrigi-los. O conjunto completo de Warnings encontrar-se-á em anexo, o exemplo seguinte trata-se de uma resolução dos Code Smells da classe AppView.

```
1 import java.awt.EventQueue;
2
3 import javax.swing.JFrame;
4 import javax.swing.JOptionPane;
5
6 import javax.swing.JButton;
7 import java.awt.event.ActionListener;
8 import java.io.IOException;
9 import java.awt.event.ActionEvent;
10 import javax.swing.JLabel;
11 import javax.swing.SwingConstants;
12
13 import java.awt.Font;
14 import javax.swing.ImageIcon;
15 import javax.swing.JTextField;
16 import javax.swing.JPasswordField;
17 import javax.swing.JSeparator;
18 import java.awt.Color;
19
20 public class AppView {
21
22     private JFrame frame;
23     private static JTextField textField;
24     private static JPasswordField passwordField;
25 }
```

Figura 75 AppView após Refactoring

INNAPROPRIATE INTIMACY

```
JLabel lblNewLabel_1 = new JLabel("New label");
lblNewLabel_1.setFont(new Font("Tahoma", Font.PLAIN, 18));
lblNewLabel_1.setBounds(10, 30, 284, 42);
contentPane.add(lblNewLabel_1);
lblNewLabel_1.setText("Montante Investido: " + n.getPlafondInvestidoModalidade(connect.getUserLogin(), "VENDA") + "€");

JButton btnFecharVenda = new JButton("Fechar Venda");
btnFecharVenda.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int id = Integer.parseInt(textField.getText());
        n.fechaVenda(id, connect.getUserLogin());
        table.setModel(connect.resultSetToTableModel(a.getAtivosVenda(connect.getUserLogin())));
        textField.setText(null);
        lblNewLabel_1.setText("Montante Investido: " + n.getPlafondInvestidoModalidade(connect.getUserLogin(), "VENDA") + "€");
    }
});
btnFecharVenda.setBounds(436, 62, 138, 23);
contentPane.add(btnFecharVenda);
```

Figura 76 Excerto de código da classe AtivoVenda

REFACTORING

- Move Field – Foram identificados inúmeros casos de Innapropriate Intimacy. Todos pelo mesmo motivo, a necessidade de aceder a um campo que identifica o id do utilizador autenticado no momento da execução dos métodos que dele necessitam e que se encontra na classe AppESS. Ora, os métodos que deste campo necessitam, estão todos eles em classes que não AppESS. A solução foi criar um campo userLogin em cada um dessas classes (Negocio e Ativo) e aceder ao mesmo localmente.
- O número de parâmetros de cada método reduziu, visto que deixa de ser preciso passar o campo userLogin como parâmetro, indo buscar o mesmo à classe AppESS, cada vez que fosse necessário. Cada método passa a aceder ao valor userLogin que encontra guardado na sua própria classe.

```
public class Negocio implements Negociacao {  
  
    private ResultSet rs;  
    private Connection con;  
    private int userLogin;
```

Figura 77 Inserção do campo userLogin em Negocio

```
public class Ativo implements Mercado {  
  
    private ResultSet rs;  
    private Connection con;  
    private int userLogin;
```

Figura 78 Inserção do campo userLogin em Ativo

```
public ViewPortfolio() throws ClassNotFoundException, SQLException {  
    AppESS connect = new AppESS();  
    Ativo a = new Ativo();  
    a.setLogin(connect.getUserLogin());
```

Figura 79 Escrita no campo userLogin no início de cada execução

```

public ViewPortfolio() {
    Ativo a = new Ativo();
    connect.fazLogin(AppView.getTextField().getText(), passText);
    a.setLogin(connect.getUserLogin());
}

```

Figura 80 Alteração na classe ViewPortfolio

```

public MenuPrincipal(AppESS connect) throws SQLException, IOException, ClassNotFoundException {
    Ativo a = new Ativo();
    Negocio n = new Negocio();
    MenuPrincipal.connect = connect;
    connect.registerObserver(this);
    connect.fazLogin(AppView.getTextField().getText(), passText);
    a.setLogin(connect.getUserLogin());
    n.setLogin(connect.getUserLogin());
}

```

Figura 81 Alteração na classe MenuPrincipal

```

public DefineVenda() throws SQLException, IOException, ClassNotFoundException {
    Negocio n = new Negocio();
    connect.fazLogin(AppView.getTextField().getText(), passText);
    n.setLogin(connect.getUserLogin());
}

```

Figura 82 Alteração na classe DefineVenda

```

public DefineCompra() throws SQLException, IOException, ClassNotFoundException {
    Negocio n = new Negocio();
    connect.fazLogin(AppView.getTextField().getText(), passText);
    n.setLogin(connect.getUserLogin());
}

```

Figura 83 Alteração na classe DefineCompra

```

public AtivosVenda() throws SQLException, IOException, ClassNotFoundException {
    Negocio n = new Negocio();
    Ativo a = new Ativo();
    connect.fazLogin(AppView.getTextField().getText(), passText);
    a.setLogin(connect.getUserLogin());
    n.setLogin(connect.getUserLogin());
}

```

Figura 84 Alteração na classe AtivosVenda

```

public AtivosCompra() throws SQLException, IOException, ClassNotFoundException {
    Negocio n = new Negocio();
    Ativo a = new Ativo();
    connect.fazLogin(AppView.getTextField().getText(), passText);
    a.setLogin(connect.getUserLogin());
    n.setLogin(connect.getUserLogin());
}

```

Figura 85 Alteração na classe AtivosCompra

ANTI-PADRÃO DE DESENHO

Um anti padrão de desenho, em engenharia de software, é uma decisão arquitetural, de gestão ou de desenvolvimento que é suposto criar um efeito negativo no projeto onde é aplicado.

A sua contraparte, os padrões de desenho normais, trazem benefícios que resultam em efeitos positivos no projeto.

Os anti padrões, são muitas vezes resultado de falta de conhecimento das partes interessadas, ou desentendimentos entre as mesmas. Em casos mais raros, podem ser o fruto de sabotagens.

Ao longo dos anos, o trabalho de investigadores na área do desenvolvimento de software, permitiu a identificação de milhares destes padrões através da presença de um conjunto de sintomas comuns, e a criação de soluções para os mesmos.

Neste projeto foi pedido que, ao contrário do que é a boa prática, fosse implementado um destes padrões, no código da primeira versão do trabalho prático, produzindo um efeito negativo no mesmo.

ANTI PADRÃO – *DARK SINGLETON*

O padrão de desenho Singleton é reconhecido pela comunidade científica de desenvolvimento de software, como sendo um padrão de desenho benéfico para o código onde o mesmo é implementado. No entanto, dentro da comunidade e da área, há quem defenda que a sua vertente positiva, pode não ser tão benéfica quanto se pensa. Há muita gente que o classifica como um anti padrão, por ser muito fácil de criar incongruências e efeitos indesejados com a sua implementação.

O padrão Singleton restringe a instanciação de uma determinada classe a apenas um objeto, i.e., uma classe X que implemente o padrão Singleton, apenas admite a criação de um objeto do tipo X, ao longo da execução do programa. Torna-se útil quando se pretende que apenas um objeto de um dado tipo seja criado ao longo de uma execução, ou para garantir, no caso do uso de Threads, que, por algum motivo indevidamente explorado, não sejam gastos recursos desnecessários com a criação de inúmeros objetos de um determinado tipo descontroladamente.

No entanto a sua aplicação em determinados pontos da primeira versão do trabalho prático, permitiu-me introduzir pelo menos um bug e um redundância que passo a explicar.

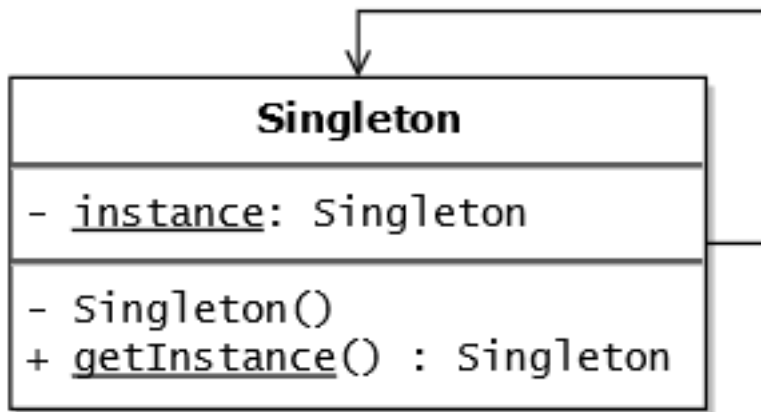


Figura 86 Padrão Singleton

“BUG”

Tendo em conta o tipo de implementação da primeira versão do meu projeto e as suas particularidades a adição do padrão Singleton aquando da criação do MenuPrincipal (que traduz o sucesso na autenticação de um utilizador no sistema) resultou no seguinte bug:

- Alterações ao código:

```
private static MenuPrincipal firstInstance = null;
```

Figura 87 Criação de uma variável de verificação

```
public static MenuPrincipal getInstance() throws ClassNotFoundException, SQLException, IOException {  
    if(firstInstance == null) {  
        firstInstance = new MenuPrincipal();  
    }  
    return firstInstance;  
}
```

Figura 88 Criação do método de verificação

```
private MenuPrincipal() throws SQLException, IOException, ClassNotFoundException {
```

Figura 89 Declaração da classe como private

```
MenuPrincipal.getInstance().setVisible(true);
```

Figura 90 Alteração no modo como MenuPrincipal é criado em AppView

- 1) Criação de uma variável “static” do tipo MenuPrincipal, onde estará guardada a única instância que poderá ser criada, numa dada execução, do tipo MenuPrincipal.
- 2) Criação do método getInstance(), que verifica que se já foi criada ou não uma instância do tipo MenuPrincipal. Caso já tenha sido criada, nada é alterado e é “recusada” a criação de uma outra instância, visto já haver outra guardada em firstInstance. Caso firstInstance esteja vazia é então criada e guardada uma instância de MenuPrincipal.
- 3) Declaração de MenuPrincipal como “private”, para impedir o acesso ou criação de instâncias MenuPrincipal por parte de outras classes sem ser pelo método getInstance().
- 4) Criação de uma instância de MenuPrincipal através do método getInstance(), resultado da ação de pressionar o botão “Log In”, na interface gráfica.

- **MenuPrincipal sem Singleton:**

- Após o programa entrar em execução o utilizador colocará os seus dados de Log In nos devidos campos, e pressionará Log In. Este ato cria uma instância de MenuPrincipal, em que o conteúdo serão os dados encontrados na BD associados à conta com que o utilizador se autenticou, apresentados numa nova janela.
- De seguida, se for esse o desejo do utilizador e sem fechar a conta anteriormente aberta, o mesmo poderá colocar novos dados de acesso, mas desta vez referentes a uma outra conta que o utilizador possua no sistema. O resultado desta ação resultará na criação de uma nova instância de MenuPrincipal, e consequentemente de uma nova janela, desta feita com os dados encontrados na BD associados à nova conta com que o utilizador se autenticou. É permitido ao utilizador monitorizar várias contas ao mesmo tempo, visto que até haver uma ação de Logout numa das contas, as diversas instâncias de MenuPrincipal (janelas) que representarão as diversas contas estarão abertas e prontas a receber instruções.

- **MenuPrincipal com *Dark Singleton*:**

- Após o programa entrar em execução o utilizador colocará os seus dados de Log In nos devidos campos, e pressionará Log In. Este ato cria uma instância de MenuPrincipal, em que o conteúdo serão os dados encontrados na BD associados à conta com que o utilizador se autenticou.
- De seguida, se for esse o desejo do utilizador e sem fechar a conta anteriormente aberta, o mesmo poderá querer colocar novos dados de acesso, mas desta vez referentes a uma outra conta que o utilizador possua no sistema. No entanto, a aplicação indevida do Singleton nesta situação, resultará no impedimento do utilizador em abrir e, portanto, monitorizar uma segunda, ou terceira conta ao mesmo tempo, pois o padrão impedirá a existência de mais que uma instância do tipo MenuPrincipal.
- Este bug poderá afastar possíveis utilizadores, pois uma das liberdades que o anti padrão veio retirar era a possibilidade de monitorizar e negociar ativos em várias frentes/contas simultaneamente.

REDUNDÂNCIA

Tendo em conta o tipo de implementação da primeira versão do meu projeto e as suas particularidades a adição do padrão Singleton aquando da consulta dos ativos com negociação aberta em modo de Compra (que traduz o sucesso na autenticação de um utilizador no sistema) resultou na seguinte redundância:

- Alterações ao código:

```
private static AtivosCompra firstInstance = null;
```

Figura 91 Criação de uma variável de verificação

```
public static AtivosCompra getInstance() throws ClassNotFoundException, SQLException, IOException {  
    if(firstInstance == null) {  
        firstInstance = new AtivosCompra();  
    }  
    return firstInstance;  
}
```

Figura 92 Criação do método de verificação

```
private AtivosCompra() throws SQLException, IOException, ClassNotFoundException {
```

Figura 93 Declaração da classe como private

```
AtivosCompra.getInstance().setVisible(true);
```

Figura 94 Alteração no modo como AtivosCompra é criado em MenuPrincipal

- 1) Criação de uma variável “static” do tipo AtivosCompra, onde estará guardada a única instância que poderá ser criada, numa dada execução, do tipo AtivosCompra.
- 2) Criação do método getInstance(), que verifica que se já foi criada ou não uma instância do tipo AtivosCompra. Caso já tenha sido criada, nada é alterado e é “recusada” a criação de uma outra instância, visto já haver outra guardada em firstInstance. Caso firstInstance esteja vazia é então criada e guardada uma instância de AtivosCompra.
- 3) Declaração de AtivosCompra como “private”, para impedir o acesso ou criação de instâncias AtivosCompra por parte de outras classes sem ser pelo método getInstance().
- 4) Criação de uma instância de AtivosCompra através do método getInstance(), resultado da ação de pressionar o botão “Ver Ativos em Compra”, na interface gráfica.

- **AtivosCompra sem Singleton:**

- Após estar autenticado, uma das funcionalidades oferecidas ao utilizador é a visualização dos ativos que o utilizador está a negociar naquele preciso momento no modo de compra. Caso seja essa a intenção do utilizador então o resultado de se pressionar o botão “Ver Ativos em Compra” será a criação de uma instância do tipo AtivosCompra, que graficamente resultará numa nova janela com a informação desejada.
- De seguida, se for esse o desejo do utilizador e sem fechar a janela anteriormente aberta, o mesmo poderá querer abrir novamente o menu de ativos em compra. Apesar de muito improvável, visto que não houve alteração que levasse à intenção de visualizar um nova janela, o utilizador terá essa liberdade, e uma nova instância de AtivosCompra será criada, que resultará em nova janela.

- **AtivosCompra com *Dark Singleton*:**

- Após estar autenticado, uma das funcionalidades oferecidas ao utilizador é a visualização dos ativos que o utilizador está a negociar naquele preciso momento no modo de compra. Caso seja essa a intenção do utilizador então o resultado de se pressionar o botão “Ver Ativos em Compra” será a criação de uma instância do tipo AtivosCompra, que graficamente resultará numa nova janela com a informação desejada.
- De seguida, se for esse o desejo do utilizador e sem fechar a janela anteriormente aberta, o mesmo poderá querer abrir novamente o menu de ativos em compra. No entanto a presença do padrão Singleton irá impedir a criação de uma outra instância de AtivosCompra, impossibilitando assim que seja aberta uma nova janela, antes da outra ser fechada.
- Sem dúvida que a manutenção e gestão de recursos ganhará com esta ação. No entanto, a menos que se trate de uma ação deliberada com más intenções, o cenário em que o desejo do utilizador é abrir uma nova janela de Ativos em Compra, com outra já aberta, é um de probabilidade extremamente diminuta, criando assim um mecanismo redundante e que apenas injeta mais complexidade no código.

IMPACTO DO REFACTORING

No final de contas, apenas é possível chegar a uma conclusão no que diz respeito ao valor do refactoring se medirmos um conjunto de métricas antes e depois do mesmo.

Para tal recorri à ferramenta SourceMonitor, para poder tirar conclusões acerca da validade do Refactoring em cada uma das versões do trabalho prático.

TP1

Java Checkpoints In Project 'TP1-BEFORE'

Item Name	Created ...	Files	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
	27 Dec 2017	13	2 519	1 883	7,4	1 279	3,2	57	2,70	9,57	46	6	2,24	2,56

Java Checkpoints In Project 'TP1-AFTER'

Item Name	Created ...	Files	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
	28 Dec 2017	12	1 994	1 435	5,6	1 165	5,6	56	1,88	10,89	46	6	2,12	2,66

Figura 95 Output SourceMonitor

	<u>FILES</u>	<u>LINES</u>	<u>STATEMENTS</u>	<u>% BRANCHES</u>	<u>CALLS</u>	<u>% COMMENTS</u>
TP1-BEFORE	13	2519	1883	7.4	1279	3.2
TP1-AFTER	12	1994	1435	5.6	1165	5.6
DELTA (%)	-7.69	-20.84	-23.79	-24.32	-8.91	+75

<u>CLASSES</u>	<u>METHODS/CLASS</u>	<u>AVG STMTS/METHOD</u>	<u>MAX COMPLEXITY</u>	<u>MAX DEPTH</u>	<u>AVERAGE DEPTH</u>	<u>AVERAGE COMPLEXITY</u>
TP1-BEFORE	2.7	9.57	46	6	2.24	2.56
TP1-AFTER	1.88	10.89	46	6	2.12	2.66
DELTA (%)	-30.37	+13.79	0	0	-5.36	+3.91

(Medidas com melhorias após o refactoring estão marcadas a verde, medidas cujo o efeito do refactoring foi negativo estão marcadas a vermelho)

Como é visível pelos dados recolhidos pela ferramenta SourceMonitor, a eliminação de Code Smells, como LongMethod, Large Class, permitiu uma redução significativa no que diz respeito a chamadas (calls), linhas (lines) e statements.

A complexidade máxima 46, deve-se à classe AfterSignUp, que apenas entra em execução quando algum utilizador quer criar uma conta, explica-se pela necessidade de verificar se todos os campos de registo estão devidamente preenchidos pelo utilizador.

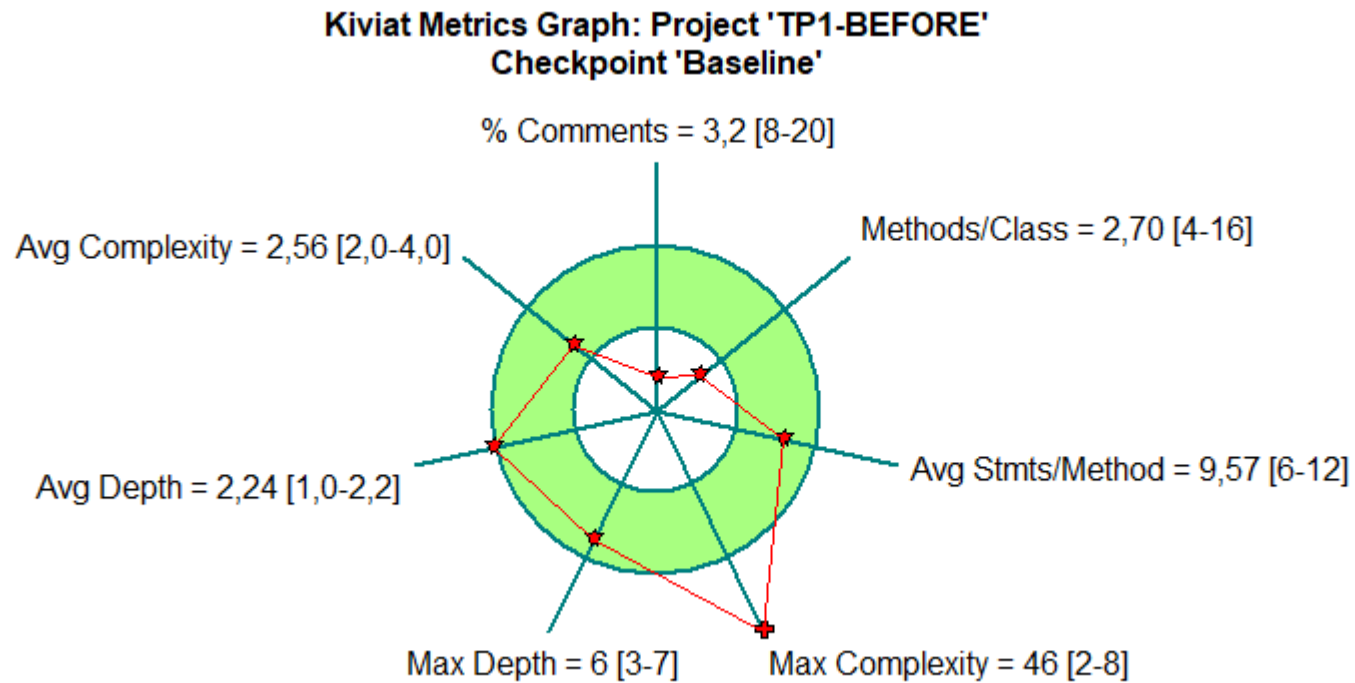


Figura 96 Gráfico de Kiviat - TP1 - Pré Refactoring

Como é possível verificar a única medida fora dos parâmetros ideais é o da complexidade máxima. A percentagem de comentários está abaixo do esperado, no entanto se fosse sobre desenvolvida tornar-se-ia rapidamente num problema por si só. Os métodos por classe apesar de estarem abaixo do esperado explicam-se pelo facto de muitas classes não conterem sequer um método, por se tratarem de classes apenas de suporte gráfico JavaSwing.

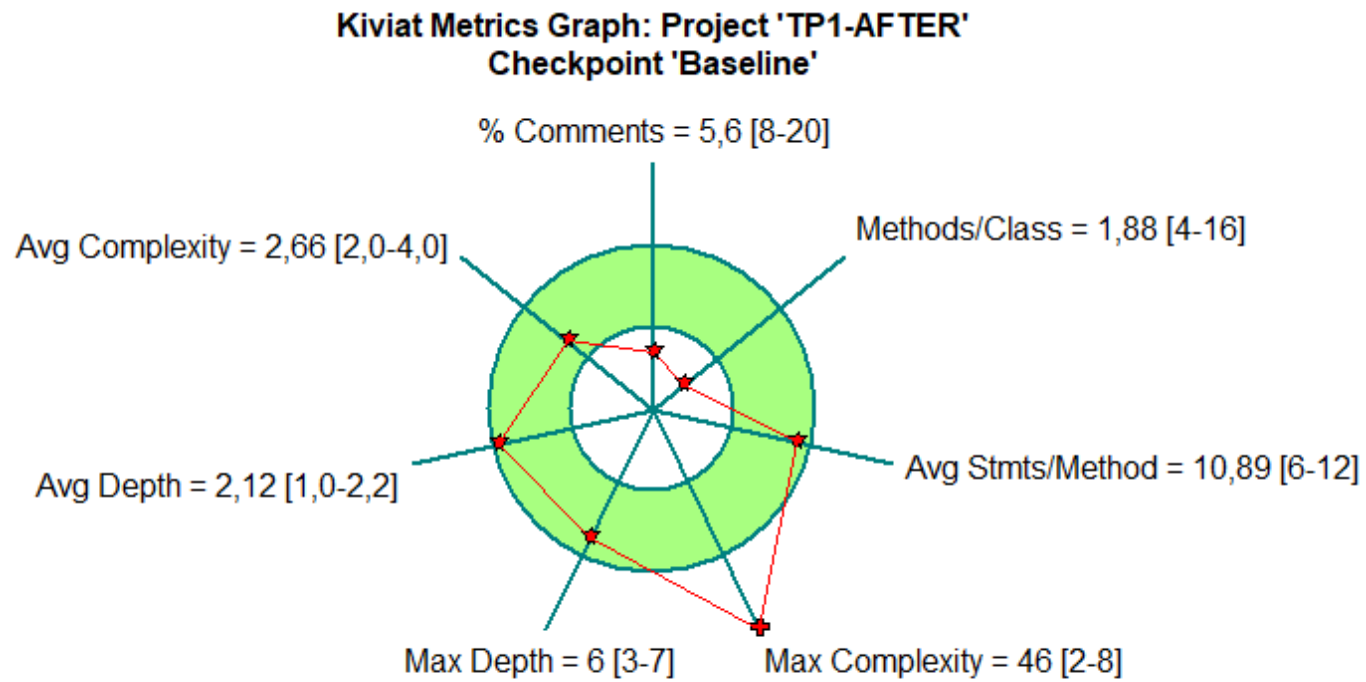


Figura 97 Gráfico de Kiviat - TP1 - Pós Refactoring

As medidas fora dos parâmetros ideais continuaram a ser as mesmas.

TP2

Java Checkpoints In Project 'TP2-BEFORE'

Name	Created ...	Files	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmt/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
	27 Dec 2017	17	3 283	2 274	7,3	1 522	5,8	72	2,75	8,77	46	8	2,24	2,50

Java Checkpoints In Project 'TP2-AFTER'

Name	Created ...	Files	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmt/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
	28 Dec 2017	19	2 516	1 827	6,0	1 413	4,2	74	2,36	7,99	46	8	2,15	2,52

Figura 98 Output SourceMonitor

	<u>FILES</u>	<u>LINES</u>	<u>STATEMENTS</u>	<u>% BRANCHES</u>	<u>CALLS</u>	<u>% COMMENTS</u>
TP1-BEFORE	17	3283	2274	7.3	1522	5.8
TP1-AFTER	19	2516	1827	6.0	1413	4.2
DELTA (%)	+11.76	-23.36	-19.65	-17.80	-7.16	-27.58

<u>CLASSES</u>	<u>METHODS/CLASS</u>	<u>AVG STMTS/METHOD</u>	<u>MAX COMPLEXITY</u>	<u>MAX DEPTH</u>	<u>AVERAGE DEPTH</u>	<u>AVERAGE COMPLEXITY</u>
TP1-BEFORE	2.75	8.77	46	8	2.24	2.50
TP1-AFTER	2.36	7.99	46	8	2.15	2.52
DELTA (%)	-14.18	-8.89	0	0	-4.01	+0.8

(Medidas com melhorias após o refactoring estão marcadas a verde, medidas cujo o efeito do refactoring foi negativo estão marcadas a vermelho)

Como é visível pelos dados recolhidos pela ferramenta SourceMonitor, a eliminação de Code Smells, como LongMethod, Large Class, permitiu uma redução significativa no que diz respeito a chamadas (calls), linhas (lines) e statements.

A complexidade máxima 46, deve-se à classe AfterSignUp, que apenas entra em execução quando algum utilizador quer criar uma conta, explica-se pela necessidade de verificar se todos os campos de registo estão devidamente preenchidos pelo utilizador.

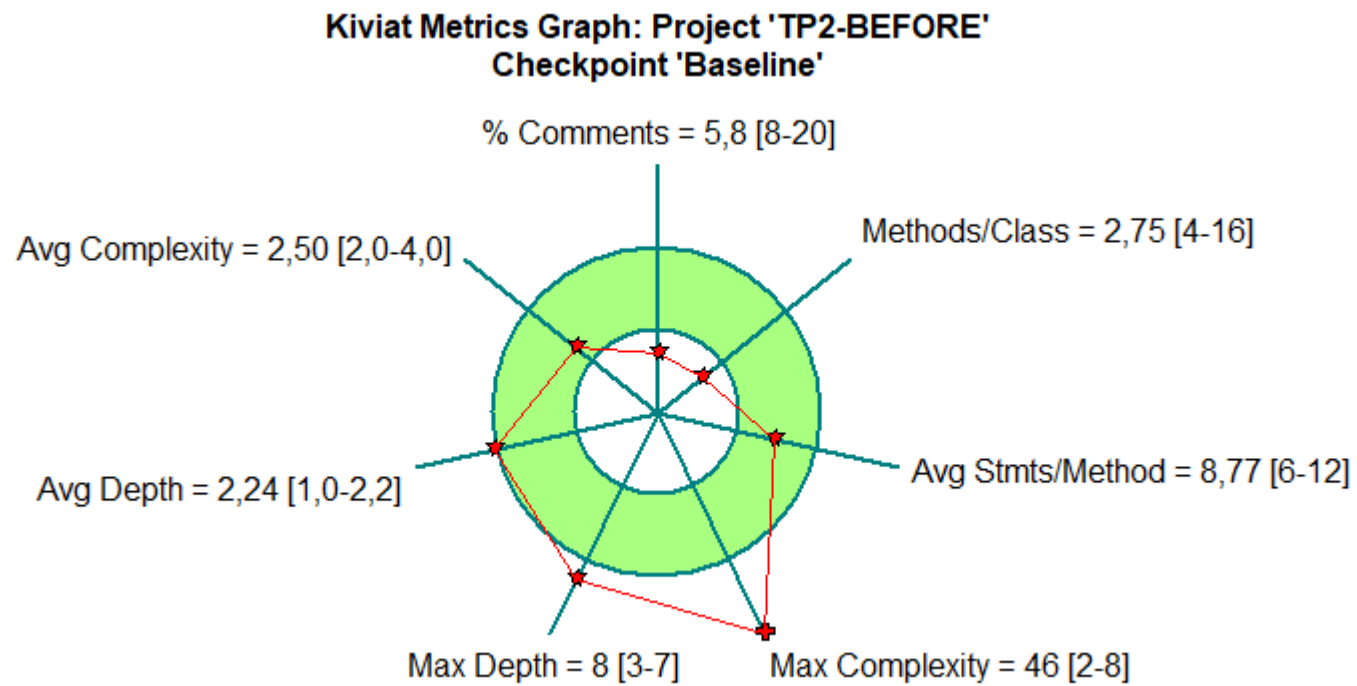


Figura 99 Gráfico Kiviat - TP2 - Pré Refactoring

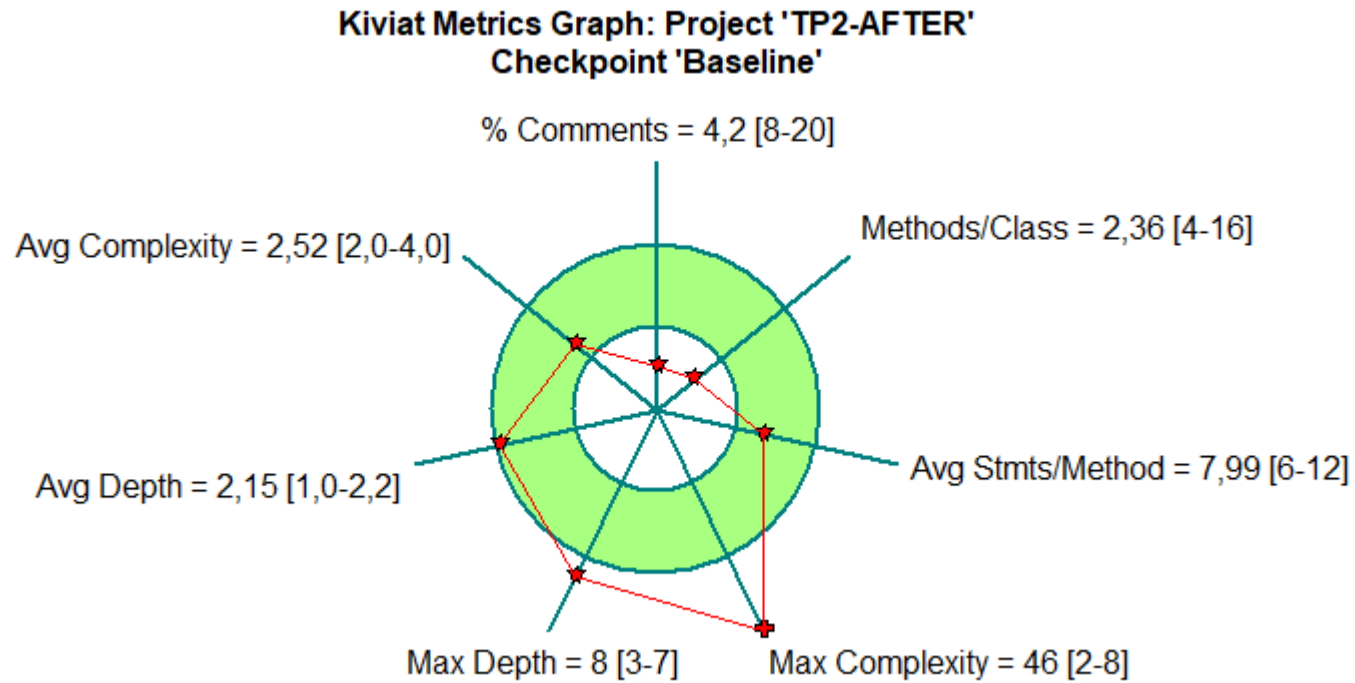


Figura 100 Gráfico Kiviat - TP2 - Pós Refactoring

A única medida que salta à vista, como estando um pouco acima dos valores ideais, algo que não acontecia na primeira versão é a profundidade (Depth) máxima. Esse valor traduz aspetos relacionados com a indentação do código e está presente numa das classes de suporte gráfico JavaSwing, nomeadamente, AtivosCompra.

CONCLUSÃO

Tendo em conta o tipo de Refactoring que foi feito, ou seja, com muita incidência nos aspetos de redução de tamanho de métodos, redução de statements, e divisão de classes muito grandes, diria que as 3 métricas mais importantes a serem analisadas serão Max Depth, Max Complexity e Number of Statements. Nestes 3 campos, houve melhorias após ter sido realizado o Refactoring. A redução no número de statements pode ser analisada nas tabelas acima, enquanto que a explicação para uma aparente manutenção dos valores de depth e complexity já foi sendo explicada, pela presença de outliers, i.e., funções que estão presentes nas classes de suporte gráfico e que estão muito fora do desvio padrão aceitável no que diz respeito a complexidade e a profundidade máxima, criando assim a percepção de que essas duas medidas estão muito fora do aceitável.

Class	Method Name	Compl...	Statements	Maximum Depth	Calls
AfterSignUp	AfterSignUp()	46	13	5	23
MenuPrincipal	MenuPrincipal()	16	13	5	12
?(instance of KeyAdapter)1	keyTyped()	8	4	5	13
?(instance of KeyAdapter)2	keyTyped()	8	4	5	13
?(instance of KeyAdapter)3	keyTyped()	8	4	5	13
?(instance of KeyAdapter)4	keyTyped()	8	4	5	13

Figura 101 Excerto da análise à Complexidade

Outlier, que altera a estatística de Max Complexity.