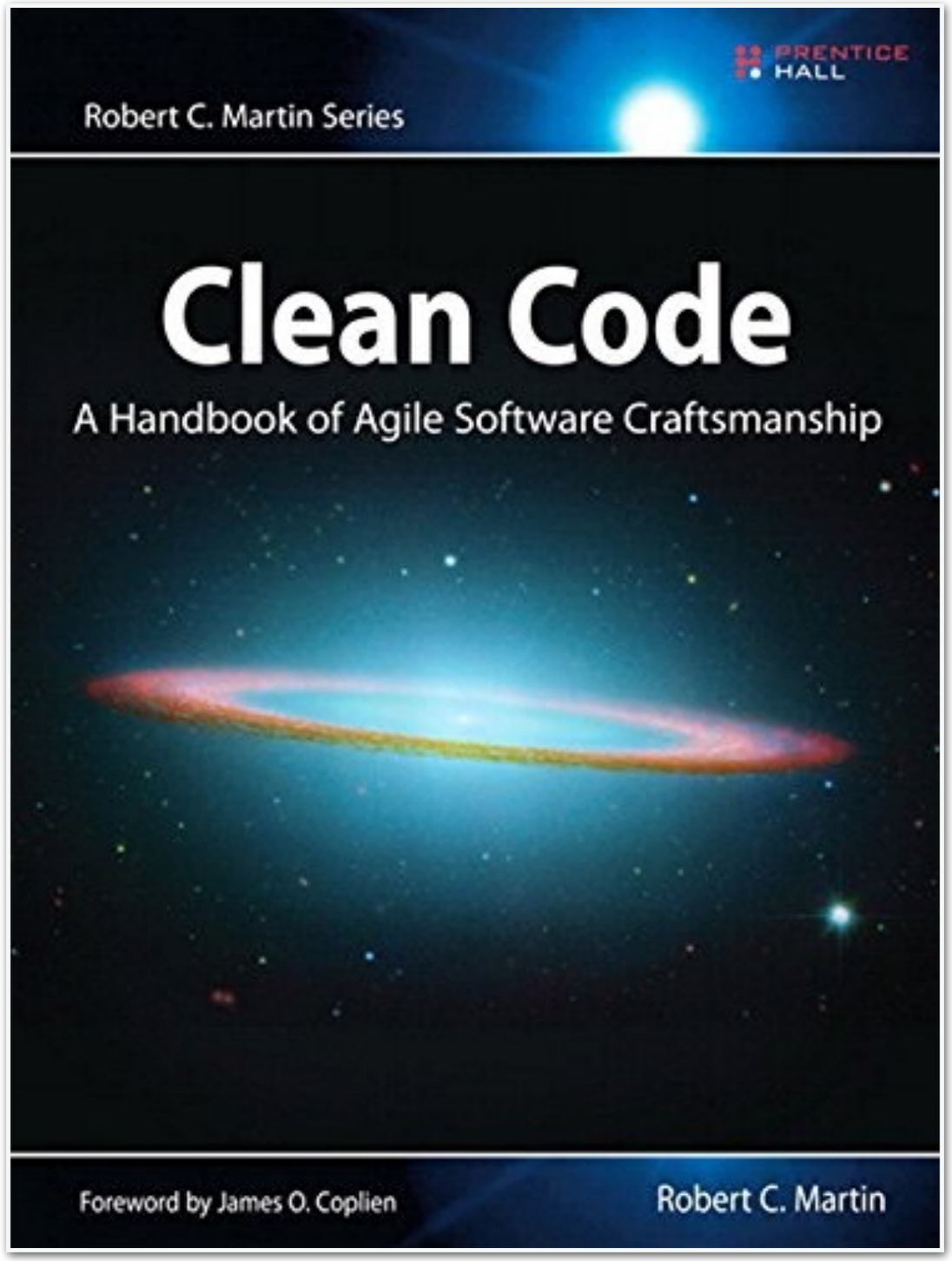
Engenharia de Software II

Código limpo - parte 4: classes

Prof. André Hora
DCC/UFMG
2019.1

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Agenda

- Legibilidade (cap 2)
- Comentários em código (cap 4)
- Formatação (cap 5)
- Funções (cap 3)
- Código externo (cap 8)
- Testes de unidade (cap 9)
- Classes (cap 10)

Agenda

- Legibilidade (cap 2)
- Comentários em código (cap 4)
- Formatação (cap 5)
- Funções (cap 3)
- Código externo (cap 8)
- Testes de unidade (cap 9)
- **Classes (cap 10)**

Classes

Classes

- Até o momento: focamos em escrever linhas e blocos de código e funções limpas

- Foco atual: subir o nível para as classes

Organização

- Ordem das entidades em uma class Java:

1. Constantes públicas
2. Atributos privados estáticos
3. Atributos privados
4. Métodos públicos
5. Métodos privados


DEUXIÈME ANNÉE. — N° 109. PRIX : 3 francs DERNIÈRE ÉDITION

Le Monde

Une révolution scientifique

LES AMERICAINS LANCENT
LEUR PREMIÈRE BOMBE ATOMIQUE SUR LE JAPON

LE PROBLEME ESPAGNOL

LA DÉCLARATION DU PRÉSIDENT TRUMAN

Soixante villes japonaises sont copieusement détruites

```java
public final class RemoteActionCompat {

    private static final String EXTRA_ICON = "icon";
    private static final String EXTRA_TITLE = "title";
    private static final String EXTRA_CONTENT_DESCRIPTION = "desc";
    private static final String EXTRA_ACTION_INTENT = "action";
    private static final String EXTRA_ENABLED = "enabled";
    private static final String EXTRA_SHOULD_SHOW_ICON = "showicon";

    private final IconCompat mIcon;
    private final CharSequence mTitle;
    private final CharSequence mContentDescription;
    private final PendingIntent mActionIntent;
    private boolean mEnabled;
    private boolean mShouldShowIcon;

    public RemoteActionCompat(@NonNull IconCompat icon, @NonNull CharSequence title,
            @NonNull CharSequence contentDescription, @NonNull PendingIntent intent) {
        mIcon = Preconditions.checkNotNull(icon);
        mTitle = Preconditions.checkNotNull(title);
        mContentDescription = Preconditions.checkNotNull(contentDescription);
        mActionIntent = Preconditions.checkNotNull(intent);
        mEnabled = true;
        mShouldShowIcon = true;
    }
```

Classes devem ser pequenas

- Mesma regra das funções: classes devem ser pequenas
- LOC: medida de tamanho nas funções
- Qual a medida de tamanho nas classes?

Classes devem ser pequenas

- Mesma regra das funções: classes devem ser pequenas
- LOC: medida de tamanho nas funções
- Qual a medida de tamanho nas classes?

Responsabilidades

Classe com muitas responsabilidades (“God class”)

```
public class SuperDashboard extends JFrame implements MetaDataUser
   public String getCustomizerLanguagePath()
   public void setSystemConfigPath(String systemConfigPath)
   public String getSystemConfigDocument()
   public void setSystemConfigDocument(String systemConfigDocument)
   public boolean getGuruState()
   public boolean getNoviceState()
   public boolean getOpenSourceState()
   public void showObject(MetaObject object)
   public void showProgress(String s)
   public boolean isMetadataDirty()
   public void setIsMetadataDirty(boolean isMetadataDirty)
   public Component getLastFocusedComponent()
   public void setLastFocused(Component lastFocused)
   public void setMouseSelectState(boolean isMouseSelected)
   public boolean isMouseSelected()
   public LanguageManager getLanguageManager()
   public Project getProject()
   public Project getFirstProject()
   public Project getLastProject()
   public String getNewProjectName()
   public void setComponentSizes(Dimension dim)
   public String getCurrentDir()
   public void setCurrentDir(String newDir)
   public void updateStatus(int dotPos, int markPos)
   public Class[] getDataBaseClasses()
   public MetadataFeeder getMetadataFeeder()
   public void addProject(Project project)
   public boolean setCurrentProject(Project project)
   public boolean removeProject(Project project)
```

Classe com muitas responsabilidades (“God class”)

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public String getCustomizerLanguagePath()
    public void setSystemConfigPath(String systemConfigPath)
    public String getSystemConfigDocument()
    public void setSystemConfigDocument(String systemConfigDocument)
    public boolean getGuruState()
    public boolean getNoviceState()
    public boolean getOpenSourceState()
    public void showObject(MetaObject object)
    public void showProgress(String s)
    public
    public
    public
    public
    public
    public
    public
    public
    public
    public void setComponentSizes(Dimension dim)
    public String getCurrentDir()
    public void setCurrentDir(String newDir)
    public void updateStatus(int dotPos, int markPos)
    public Class[] getDataBaseClasses()
    public MetadataFeeder getMetadataFeeder()
    public void addProject(Project project)
    public boolean setCurrentProject(Project project)
    public boolean removeProject(Project project)
```

E se a classe contém apenas 5 métodos?

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Classe com muitas responsabilidades (“God class”)

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public String getCustomizerLanguagePath()
    public void setSystemConfigPath(String systemConfigPath)
    public String getSystemConfigDocument()
    public void setSystemConfigDocument(String systemConfigDocument)
    public boolean getGuruState()
    public boolean getNoviceState()
    public boolean getOpenSourceState()
    public void showObject(MetaObject object)
    public void showProgress(String s)
    public void setComponentSizes(Dimension dim)
    public String getCurrentDir()
    public void setCurrentDir(String newDir)
    public void updateStatus(int dotPos, int markPos)
    public Class[] getDataBaseClasses()
    public MetadataFeeder getMetadataFeeder()
    public void addProject(Project project)
    public boolean setCurrentProject(Project project)
    public boolean removeProject(Project project)
```

E se a classe contém apenas 5 métodos?

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Classe com muitas responsabilidades (“God class”)

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public String getCustomizerLanguagePath()
    public void setSystemConfigPath(String systemConfigPath)
    public String getSystemConfigDocument()
    public void setSystemConfigDocument(String systemConfigDocument)
    public boolean getGuruState()
    public boolean getNoviceState()
    public boolean getOpenSourceState()
    public void showObject(MetaObject object)
    public void showProgress(String s)
    public
    public
    public
    public
    public
    public
    public
    public
    public
    public
    public
    public void setComponentSizes(Dimension dim)
    public String getCurrentDir()
    public
    public
    public
    public
    public
    public
    public
```

E se a classe contém apenas 5 métodos?

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Fornece acesso ao último componente focado

e

permite rastrear números de versões e builds

Classe com muitas responsabilidades ("God class")

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public String getCustomizerLanguagePath()
    public void setSystemConfigPath(String systemConfigPath)
    public String getSystemConfigDocument()
    public void setSystemConfigDocument(String systemConfigDocument)
    public boolean getGuruState()
    public boolean getNoviceState()
    public boolean getOpenSourceState()
    public void showObject(MetaObject object)
    public void showProgress(String s)
    public
    public
    public
    public
    public
    public
    public
    public
    public
    public
    public void setComponentSizes(Dimension dim)
    public String getCurrentDir()
    public
    public
    public
    public
    public
    public
    public
```

E se a classe contém apenas 5 métodos?

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Fornece acesso ao último componente focado

e

permite rastrear números de versões e builds

2 responsabilidades

Single Responsibility Principle (SRP)

- SRP: uma classe deve ter apenas uma razão para mudar

- Classes devem ter apenas uma responsabilidade (ou seja, uma razão para mudar)

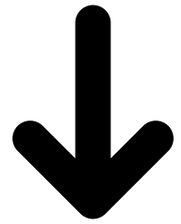
SuperDashboard tem 2 razões para mudar:

1. Rastreia números de versões e builds, logo deve ser atualizada a cada nova versão

2. Gerencia componentes do Java Swing, logo deve ser atualizada em mudanças de GUI

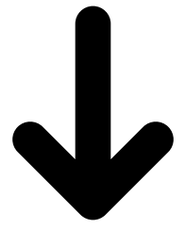
```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```



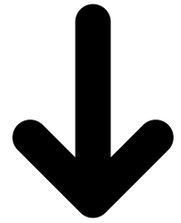
?

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```



```
public class Version {
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```



```
public class Version {
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Classe com grande potencial de reuso

Single Responsibility Principle (SRP)

- Um dos conceitos mais importantes no projeto OO
- Um conceito muito simples de entender
- Ainda assim, é um conceito frequentemente violado, por quê?

Single Responsibility Principle (SRP)

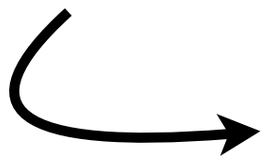
- Um dos conceitos mais importantes no projeto OO
- Um conceito muito simples de entender
- Ainda assim, é um conceito frequentemente violado, por quê?

fazer o software funcionar ≠ tornar o código limpo

Single Responsibility Principle (SRP)

- Um dos conceitos mais importantes no projeto OO
- Um conceito muito simples de entender
- Ainda assim, é um conceito frequentemente violado, por quê?

fazer o software funcionar ≠ tornar o código limpo



Frequentemente paramos aqui, e vamos resolver o próximo problema

Coesão

- Idealmente, classes devem ter poucos atributos
- Cada método deve manipular um ou mais atributos
- Em geral, quanto mais atributos um método manipula, mais coesivo é esse método
  - Coesão máxima: uma classe em que cada atributo é utilizado por cada método (cenário irreal)
- Idealmente, coesão de uma classe deve ser alta
  - Coesão alta: métodos e atributos são dependentes e juntos representam um “todo”

Classe coesa?

```
public class Stack {
  private int topOfStack = 0;
  List<Integer> elements = new LinkedList<Integer>();

  public int size() {
    return topOfStack;
  }

  public void push(int element) {
    topOfStack++;
    elements.add(element);
  }

  public int pop() throws PoppedWhenEmpty {
    if (topOfStack == 0)
      throw new PoppedWhenEmpty();
    int element = elements.get(--topOfStack);
    elements.remove(topOfStack);
    return element;
  }
}
```

Classe coesa? Sim

```
public class Stack {
  private int topOfStack = 0;
  List<Integer> elements = new LinkedList<Integer>();

  public int size() {
    return topOfStack;
  }

  public void push(int element) {
    topOfStack++;
    elements.add(element);
  }

  public int pop() throws PoppedWhenEmpty {
    if (topOfStack == 0)
      throw new PoppedWhenEmpty();
    int element = elements.get(--topOfStack);
    elements.remove(topOfStack);
    return element;
  }
}
```

Manter alta coesão resulta em classes pequenas

- Para manter alta coesão, classes com baixa coesão devem devem ser divididas em classes menores (e mais coesas)

- Quebrar métodos longos em métodos pequenos também pode causar a criação de classes coesas

- Mais classes: fornece ao programa mais organização e estruturas transparentes

```
public class PrintPrimes {
  public static void main(String[] args) {
    final int M = 1000;
    final int RR = 50;
    final int CC = 4;
    final int WW = 10;
    final int ORDMAX = 30;
    int P[] = new int[M + 1];
    int PAGENUMBER;
    int PAGEOFFSET;
    int ROWOFFSET;
    int C;
    int J;
    int K;
    boolean JPRIME;
    int ORD;
    int SQUARE;
    int N;
    int MULT[] = new int[ORDMAX + 1];
    J = 1;
    K = 1;
    P[1] = 2;
    ORD = 2;
    SQUARE = 9;

    while (K < M) {
      do {
        J = J + 2;
        if (J == SQUARE) {
          ORD = ORD + 1;
          SQUARE = P[ORD] * P[ORD];
          MULT[ORD - 1] = J;
        }
        N = 2;
        JPRIME = true;
        while (N < ORD && JPRIME) {
          while (MULT[N] < J)
            MULT[N] = MULT[N] + P[N] + P[N];
          if (MULT[N] == J)
            JPRIME = false;
          N = N + 1;
        }
      } while (!JPRIME);
      K = K + 1;
      P[K] = J;
    }
    {
      PAGENUMBER = 1;
      PAGEOFFSET = 1;
      while (PAGEOFFSET <= M) {
        System.out.println("The First " + M +
                           " Prime Numbers --- Page " + PAGENUMBER);
        System.out.println("");
        for (ROWOFFSET = PAGEOFFSET; ROWOFFSET < PAGEOFFSET + RR; ROWOFFSET++){
          for (C = 0; C < CC;C++)
          if (ROWOFFSET + C * RR <= M)
            System.out.format("%10d", P[ROWOFFSET + C * RR]);
          System.out.println("");
        }
        System.out.println("\f");
        PAGENUMBER = PAGENUMBER + 1;
        PAGEOFFSET = PAGEOFFSET + RR * CC;
      }
    }
  }
}
```

```
public class PrintPrimes {
  public static void main(String[] args) {
    final int M = 1000;
    final int RR = 50;
    final int CC = 4;
    final int WW = 10;
    final int ORDMAX = 30;
    int P[] = new int[M + 1];
    int PAGENUMBER;
    int PAGEOFFSET;
    int ROWOFFSET;
    int C;
    int J;
    int K;
    boolean JPRIME;
    int ORD;
    int SQUARE;
    int N;
    int MULT[] = new int[ORDMAX + 1];
    J = 1;
    K = 1;
    P[1] = 2;
    ORD = 2;
    SQUARE = 9;

    while (K < M) {
      do {
        J = J + 2;
        if (J == SQUARE) {
          ORD = ORD + 1;
          SQUARE = P[ORD] * P[ORD];
          MULT[ORD - 1] = J;
        }
        N = 2;
        JPRIME = true;
        while (N < ORD && JPRIME) {
          while (MULT[N] < J)
            MULT[N] = MULT[N] + P[N] + P[N];
          if (MULT[N] == J)
            JPRIME = false;
          N = N + 1;
        }
      } while (!JPRIME);
      K = K + 1;
      P[K] = J;
    }
    {
      PAGENUMBER = 1;
      PAGEOFFSET = 1;
      while (PAGEOFFSET <= M) {
        System.out.println("The First " + M +
                           " Prime Numbers --- Page " + PAGENUMBER);
        System.out.println("");
        for (ROWOFFSET = PAGEOFFSET; ROWOFFSET < PAGEOFFSET + RR; ROWOFFSET++){
          for (C = 0; C < CC;C++)
          if (ROWOFFSET + C * RR <= M)
            System.out.format("%10d", P[ROWOFFSET + C * RR]);
          System.out.println("");
        }
        System.out.println("\f");
        PAGENUMBER = PAGENUMBER + 1;
        PAGEOFFSET = PAGEOFFSET + RR * CC;
      }
    }
  }
}
```

```
public class PrimePrinter {
  public static void main(String[] args) {
  final int NUMBER_OF_PRIMES = 1000;
  int[] primes = PrimeGenerator.generate(NUMBER_OF_PRIMES);

  final int ROWS_PER_PAGE = 50;
  final int COLUMNS_PER_PAGE = 4;
  RowColumnPagePrinter tablePrinter =
    new RowColumnPagePrinter(ROWS_PER_PAGE,
                             COLUMNS_PER_PAGE,
                             "The First " + NUMBER_OF_PRIMES +
                                    " Prime Numbers");
    tablePrinter.print(primes);
  }

}
```

```
public class PrimePrinter {
  public static void main(String[] args) {
  final int NUMBER_OF_PRIMES = 1000;
  int[] primes = PrimeGene

  final int ROWS_PER_PAGE
  final int COLUMNS_PER_PA
  RowColumnPagePrinter tab
    new RowColumnPagePrint



    tablePrinter.print(pri
  }

}
```

```
public class RowColumnPagePrinter {
  private int rowsPerPage;
  private int columnsPerPage;
  private int numbersPerPage;
  private String pageHeader;
  private PrintStream printStrean

  public RowColumnPagePrinter(int
                              int
                              Str
    this.rowsPerPage = rowsPerPag
    this.columnsPerPage = columns
    this.pageHeader = pageHeader;
    numbersPerPage = rowsPerPage
    printStream = System.out;
}
```

```
public class PrimeGenerator {
  private static int[] primes;
  private static ArrayList<Integer> multiplesOfPrimeFactors;

  protected static int[] generate(int n) {
    primes = new int[n];
    multiplesOfPrimeFactors = new ArrayList<Integer>();
    set2AsFirstPrime();
    checkOddNumbersForSubsequentPrimes();
    return primes;
  }

  private static void set2AsFirstPrime() {
    primes[0] = 2;
    multiplesOfPrimeFactors.add(2);
  }
```

- Código ficou mais longo
- Variáveis com nomes mais significativos
- Mais métodos e classes (como forma de documentação)
- Mais formatação
- 3 responsabilidades - 3 razões para mudar:
  - Execução, formatação e geração de primos

Exercício

- Quantas razões para mudar a classe Sql possui? Quais?
- Apresente uma solução para isolar esse problema

```
public class Sql {   public Sql(String table, Column[] columns)
   public String create()
   public String insert(Object[] fields)
   public String selectAll()
   public String findByKey(String keyColumn, String keyValue)
   public String select(Column column, String pattern)
   public String select(Criteria criteria)
   public String preparedInsert()
   private String columnList(Column[] columns)
   private String valuesList(Object[] fields, final Column[] columns)
   private String selectWithCriteria(String criteria)
   private String placeholderList(Column[] columns)
}
```

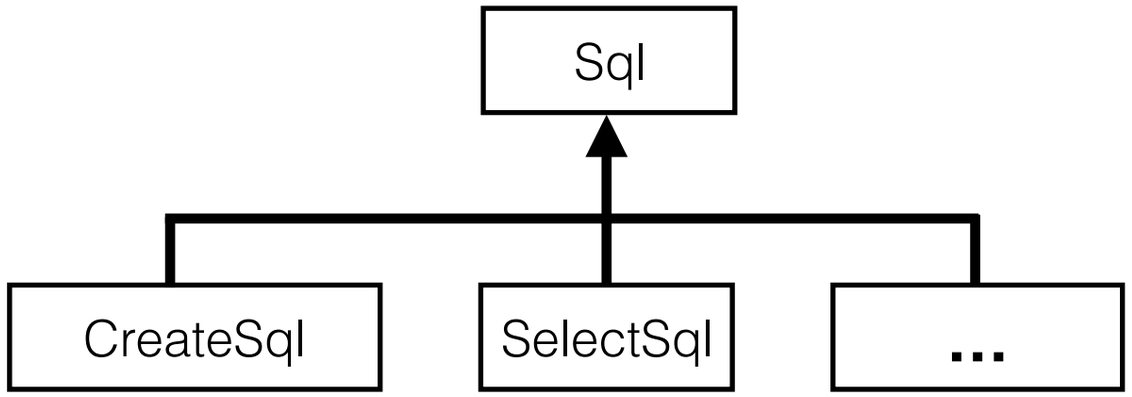
Exercício

- Quantas razões para mudar a classe Sql possui? Quais?

1. Quando novos tipos de statements são adicionados (ex: update, delete)

2. Quando alteramos os detalhes de um tipo de statement

Violação do SRP




```
abstract public class Sql {
   public Sql(String table, Column[] columns)
   abstract public String generate();
}

public class CreateSql extends Sql {
   public CreateSql(String table, Column[] columns)
   @Override public String generate()
}

public class SelectSql extends Sql {
   public SelectSql(String table, Column[] columns)
   @Override public String generate()
}

public class InsertSql extends Sql {
   public InsertSql(String table, Column[] columns, Object[] fields)
   @Override public String generate()
   private String valuesList(Object[] fields, final Column[] columns)
}

public class SelectWithCriteriaSql extends Sql {
   public SelectWithCriteriaSql(
   String table, Column[] columns, Criteria criteria)
   @Override public String generate()
}
```

- Cada alteração afeta apenas a sua classe
- Novas classes podem ser adicionadas sem impactar outras
- Suporta o SRP
- Também suporta OCP Open-Closed Principle (classes devem ser abertas para extensão, mas fechadas para modificação)

Revisão Código Limpo

- Para cada tópico, liste 3 boas práticas:
  - Legibilidade, comentários e formatação (parte 1)
  - Funções (parte 2)
  - Código externo e testes de unidade (parte 3)
  - Classes (parte 4)

Legibilidade

- Usar nomes que revelam a intenção
- Fazer distinção entre nomes
- Usar nomes pronunciáveis
- Usar nomes “buscáveis”

Comentários

- Não existe nada mais útil do que um bom comentário

- Não existe nada pior do que um comentário ruim, redundante, desnecessário…

- O propósito dos comentários é compensar a falha de nos expressarmos em código

Formatação

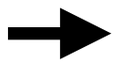
- Formatação vertical
- Separação de conceitos
- Funções dependentes
- Afinidade conceitual
- Formatação horizontal
- Indentação

Função deve ser pequena

- Primeira regra: funções devem ser pequenas
- Baseado na experiência do autor (50 anos de desenvolvimento!)
- Idealmente, ifs, elses, whiles devem ter **uma linha**

```
public static String renderPageWithSetupsAndTeardowns(
  PageData pageData, boolean isSuite
) throws Exception {
  boolean isTestPage = pageData.hasAttribute("Test");
  if (isTestPage) {
    WikiPage testPage = pageData.getWikiPage();
    StringBuffer newPageContent = new StringBuffer();
    includeSetupPages(testPage, newPageContent, isSuite);
    newPageContent.append(pageData.getContent());
    includeTeardownPages(testPage, newPageContent, isSuite);
    pageData.setContent(newPageContent.toString());
  }

  return pageData.getHtml();
}
```



```
public static String renderPageWithSetupsAndTeardowns(
  PageData pageData, boolean isSuite
) throws Exception {
  boolean isTestPage = pageData.hasAttribute("Test");
  if (isTestPage(pageData))
    includeSetupAndTeardownPages(pageData, isSuite);
  return pageData.getHtml();
}
```

Função deve realizar apenas uma tarefa

- A primeira função realiza muitas tarefas
  - Cria buffers, busca páginas, busca heranças, faz parsers, acrescenta strings, gera HTML, etc
  - Possui vários níveis de abstração:
    - Alto nível: getHtml()
    - Nível intermediário: PathParser.render(pagePath)
    - Baixo nível: append(“\n”)
- A última função faz apenas uma coisa: *adiciona setups e teardowns em páginas de testes*

Switches

Qual o problema desta função?

```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
      case COMMISSIONED:
        return calculateCommissionedPay(e);
      case HOURLY:
        return calculateHourlyPay(e);
      case SALARIED:
        return calculateSalariedPay(e);
      default:
        throw new InvalidEmployeeType(e.type);
    }
  }
```

- Grande e vai crescer se novos empregados forem adicionados
- Realiza mais de uma tarefa
- Viola o *Open Closed Principle* (entidades devem ser abertas para extensão, mas fechadas para modificação)
- **Principal**: outras funções terão a mesma estrutura (duplicação)
  - ex: isPayday(), deliverPay(), etc

Utilizando Código Externo

- Existe uma tensão natural entre produtores e usuários de interfaces

- **Produtores**: querem interfaces que funcionem em diversos ambientes para obter mais audiência

- **Usuários**: querem interfaces focadas em suas necessidades particulares

Testes de Unidade

- Código de teste é tão importante quanto de produção
  - Devem ser projetados e mantidos
  - Garante que alterações não quebram o código existente
- **Com testes**: código se mantém flexível, pois facilita alterações; sistema pode ser melhorado
- **Sem testes**: alteração é uma possível fonte de bugs

FIRST

- **Fast**: testes devem ser rápidos; rodar frequentemente
- **Independent**: testes não devem depender de outros; para evitar efeito cascata (se um falha, outros falham)
- **Repeatable**: testes devem ser repetíveis em qualquer ambiente (produção e laptop), em qualquer ordem; rodar N vezes e obter o mesmo resultado
- **Self-Validating**: testes devem ter saída boleana; ou passam ou falham (sem avaliação manual de logs)
- **Timely**: testes devem ser escritos antes do código (TDD)