Universidade Federal de Minas Gerais

Bacharel em Sistemas de Informação
Programação Modular

Trabalho Prático 1
Setembro 2018

Gabriel Silva Bastos
Matrícula: 2016058204

1 Passeio do cavalo

No trabalho prático, recebemos a tarefa de implementar um algoritmo para calcular o passeio do cavalo. O passeio do cavalo é um caminho que o cavalo percorre, cobrindo todo o tabuleiro de xadrez, partindo de uma casa inicial definida. O passeio só é válido se o cavalo percorrer todas as casas do tabuleiro.

2 Algoritmo

De acordo com a descrição do algoritmo a ser utilizado, implementei o código com a técnica de *backtracking*. Este explora todas as rotas, e retrocede caso um caminho sem saída seja alcançado antes de se completar o passeio. Ao completar o tabuleiro, o algoritmo chega em seu fim. Caso nenhum passeio completo seja encontrado, indica-se a impossibilidade deste.

3 Análise experimental

Para a análise experimental da eficiência do algoritmo, executei a simulação até obter os resultados para cinco coordenadas iniciais distintas. Desta forma, podemos comparar o efeito da posição inicial no tempo de execução e no número de movimentos realizados.

Coordenada inicial	Movimentos Realizados	Tempo de execução [1]
(3, 2)	2252173	00:00:00.55
(4, 5)	11807342	00:00:01.98
(0, 3)	59468789	00:00:09.05
(3, 5)	403776881	00:01:01.66
(2, 2)	1173382931400	34:35:42.14

54	45	50	59	56	43	36	33
49	60	55	44	51	34	57	42
26	53	46	29	58	37	32	35
61	48	27	52	21	30	41	38
14	25	62	47	28	39	20	31
63	6	15	22	1	18	9	40
24	13	4	7	16	11	2	19
5	64	23	12	3	8	17	10

Passeio partindo da posição (4, 5)

56	49	60	33	54	47	40	35
61	32	55	48	59	34	53	46
28	57	50	31	52	41	36	39
1	62	29	58	23	38	45	42
16	27	2	51	30	43	22	37
63	8	17	24	3	20	11	44
26	15	6	9	18	13	4	21
7	64	25	14	5	10	19	12

Passeio partindo da posição (0, 3)

38	31	52	59	36	45	50	47
53	58	37	32	51	48	35	44
30	39	54	1	60	33	46	49
57	62	25	40	55	2	43	34
14	29	56	61	24	41	20	3
63	8	15	26	11	18	23	42
28	13	6	9	16	21	4	19
7	64	27	12	5	10	17	22

Passeio partindo da posição (3, 5)

57	54	21	36	45	34	27	24
20	37	58	55	22	25	44	33
53	56	1	46	35	28	23	26
38	19	52	59	2	43	32	29
51	60	39	18	47	30	3	42
8	63	48	11	40	17	14	31
61	50	9	6	15	12	41	4
64	7	62	49	10	5	16	13

Passeio partindo da posição (2, 2)

A partir dos resultados obtidos, concluí que a posição inicial apresenta grande influência no tempo de execução do algoritmo. Isto se deve ao fato de que algumas posições iniciais dispõem de muitas rotas válidas como passeio, enquanto outras apresentam pouquíssimas rotas completas. Assim, caso uma posição inicial com poucos passeios válidos seja sorteada, o tempo de execução do algorítmo é altamente influenciado negativamente.

[1] Formato do tempo: hh:mm:ss.ms

4 Módulos

4.1 Board

4.1.1 Coord

Representa uma coordenada no espaço $\mathbb{Z}^2$, utilizada para o posicionamento no tabuleiro.

Interface

```
public class Coord {
  public final int x, y; // Axis coordinates.

  public static final Coord origin; // The origin coordinate, i.e. (0, 0).

  public Coord(int x, int y); // Basic contructor.

  public static Coord random(Bounds b); // Create a random coord within the bounds.
}
```

Implementação

```
package dcc.gahag.chess.board;

import java.util.Random;


/**
 * A bidimensional immutable coordinate.
 */
public class Coord {
  public final int x, y;

  /**
   * The origin coordinate, i.e. (0, 0).
   */
  public static final Coord origin = new Coord(0, 0);

  public Coord(int x, int y) {
    this.x = x;
    this.y = y;
  }

  /**
   * Construct a random Coord, within the given bounds, if any.
   * @param b the bounds within the Coord will be randomized, possibly null
   */
  public static Coord random(Bounds b) {
    if (b == null) {
      Random r = new Random();

      return new Coord(r.nextInt(), r.nextInt());
    }

    return new Coord(
      b.lower.x + (int) (Math.random() * (b.upper.x - b.lower.x)),
      b.lower.y + (int) (Math.random() * (b.upper.y - b.lower.y))
    );
  }
}
```

Coesão

O módulo apresenta coesão funcional, cumprindo apenas o papel de representar coordenadas.

Acoplamento

O módulo apresenta acoplamento de dados com os módulos padrão *Math* e *Random*, e também com o módulo *Bounds*.

4.1.2 Bounds

Representa os limites de um subespaço de $\mathbb{Z}^2$, delimitando o tabuleiro e o movimento das peças.

Interface

```
public class Bounds {
  public final Coord lower, upper; // The upper and lower bounds.
  public Bounds(Coord lower, Coord upper); // Basic constructor.
  public boolean contains(Coord c);  // Checks wether the bounds contains the coordinate.
}
```

Implementação

```
package dcc.gahag.chess.board;

/**
 * Immutable bidimensional boundaries.
 */
public class Bounds {
  public final Coord lower, upper;

  /**
   * Construct a Bounds object from the given lower and upper coordinates.
   * Lower's coordinates must be equal or lower than upper's.
   * @param lower the lower coordinate, mustn't be null
   * @param upper the upper coordinate, mustn't be null
   */
  public Bounds(Coord lower, Coord upper) {
    if (lower == null || upper == null)
      throw new IllegalArgumentException("lower/upper mustn't be null");

    if (upper.x < lower.x || upper.y < lower.y)
      throw new IllegalArgumentException(
        "lower's coordinates must be equal or lower than upper's"
      );

    this.lower = lower;
    this.upper = upper;
  }

  /**
   * Checks wether the bounds contains the given coordinate.
   * @param c the coordinate to check, possibly null
   */
  public boolean contains(Coord c) {
    if (c == null)
      return false;

    return this.lower.x <= c.x && c.x <= this.upper.x
        && this.lower.y <= c.y && c.y <= this.upper.y;
  }
}
```

Coesão

O módulo apresenta coesão funcional, cumprindo apenas o papel de representar, criar e manipular limites.

Acoplamento

O módulo apresenta acoplamento de dados com o módulo *Coord*.

4.1.3 Board

Representa o tabuleiro de xadrez, com seus limites e os valores de cada casa. As peças não pertencem ao tabuleiro.

Interface

```
public class Board {
  public final int width;      // The width of the board.
  public final int size;       // The size of the board. Equivalent to `width * width`.
  public final Bounds bounds;  // The boundaries of the board.

  public Board(int width); // Construct a Board of the given width.

  public int getTile(Coord c);             // get the piece's position.
  public void setTile(Coord c, int value); // set the piece's position.
  public void resetTile(Coord c); // Reset the value of a given tile (set to 0).

  // Calculate the possible steps for the given piece within the board:
  public Iterable<Coord> steps(final IPiece p);

  public void print(PrintStream p);  // Print the board to the given PrintStream.
}
```

Implementação

```
package dcc.gahag.chess.board;


import java.io.PrintStream;

import dcc.gahag.chess.piece.IPiece;
import dcc.gahag.chess.util.FilterIterator;
import dcc.gahag.chess.util.TransformIterator;


/**
 * A chess board with mutable tiles and immutable size.
 * It's tiles are represented by integers, possibly indicating the movement number.
 * The tiles' default value is 0.
 */
public class Board {
  protected int[][] tiles;

  /**
   * The width of the board.
   */
  public final int width;
  /**
   * The size of the board. Equivalent to `width * width`.
   */
  public final int size;
```

```java
/**
 * The boundaries of the board.
 */
public final Bounds bounds;


/**
 * Construct a Board of the given width.
 * @param width the width of the board, must be a positive number
 */
public Board(int width) {
  if (width < 1)
    throw new IllegalArgumentException();

  this.bounds = new Bounds(
    Coord.origin,
    new Coord(width - 1, width - 1)
  );
  this.tiles = new int[width][width];

  this.width = width;
  this.size = width * width;
}

/**
 * Get the value of a given tile.
 * @param c the Coord of the tile, must be within the board's bounds and mustn't be null
 */
public int getTile(Coord c) {
  if (c == null)
    throw new IllegalArgumentException("Coord mustn't be null");

  if (!this.bounds.contains(c))
    throw new IllegalArgumentException("Coord out of bounds");

  return this.tiles[c.x][c.y];
}

/**
 * Set the value of a given tile.
 * @param c the Coord of the tile, must be within the board's bounds and mustn't be null
 * @param value the value to be set
 */
public void setTile(Coord c, int value) {
  if (c == null)
    throw new IllegalArgumentException("Coord mustn't be null");

  if (!this.bounds.contains(c))
    throw new IllegalArgumentException("Coord out of bounds");

  this.tiles[c.x][c.y] = value;
}

/**
 * Reset the value of a given tile (i.e., set the tile's value to 0).
 * @param c the Coord of the tile, must be within the board's bounds and mustn't be null
 * @param value the value to be set
 */
public void resetTile(Coord c) {
  this.setTile(c, 0);
}
```

```
  /**
   * Calculate the possible steps for the given piece within the board.
   * The values are calculated from the piece's movement, current position at the moment
   * of the function call, and the board's boundaries.
   * @param p the piece, mustn't be null
   */
  public Iterable<Coord> steps(final IPiece p) {
    if (p == null)
      throw new IllegalArgumentException("IPiece mustn't be null");

    // Keep the piece's current position, for if it changes the iterator won't change.
    final Coord c = p.getPosition();

    return () -> new FilterIterator<Coord>(
      new TransformIterator<Coord, Coord>(
        p.movement().iterator(),
        (Coord m) -> new Coord(c.x + m.x, c.y + m.y)
      ),
      (Coord m) -> this.bounds.contains(m)
    );
  }


  /**
   * Print the board's representation to the given PrintStream.
   * @param p the PrintStream to be used, mustn't be null
   */
  public void print(PrintStream p) {
    if (p == null)
      throw new IllegalArgumentException("PrintStream mustn't be null");

    for (int i = 0; i < tiles.length; i++) {
      for (int j = 0; j < tiles.length; j++)
        p.format("%2d ", this.tiles[j][i]);

      p.println();
    }
  }
}
```

Coesão

O módulo apresenta coesão funcional, cumprindo apenas o papel de representar, criar e manipular o tabuleiro.

Acoplamento

O módulo apresenta acoplamento de dados com o módulo padrão *PrintStream*, e também com os módulos *Coord*, *Bounds*, *IPiece*, *FilterIterator* e *TransformIterator*.

4.2 Piece

4.2.1 IPiece

A interface para as peças do tabuleiro, utilizada para posicionamento e movimentação, além da base para o algoritmo do passeio. Esta interface permite a implementação para o passeio do cavalo, bem como para de qualquer outra peça.

Interface

```
public interface IPiece {
  Coord getPosition();        // get the piece's position.
  void setPosition(Coord c);  // set the piece's position.

  Iterable<Coord> movement(); // get the piece's movement mechanics.

  // The tour algorithm:
  default boolean tour(final Board board);
  default boolean tour(final Board board, final Box<Long> moves);
}
```

Implementação

```
package dcc.gahag.chess.piece;

import java.util.function.IntPredicate;

import dcc.gahag.chess.board.Board;
import dcc.gahag.chess.board.Coord;
import dcc.gahag.chess.util.Box;


/**
 * The basic interface of a chess piece.
 */
public interface IPiece {
  /**
   * The piece's current position.
   */
  Coord getPosition();
  /**
   * Set the piece's current position.
   */
  void setPosition(Coord c);

  /**
   * The movement mechanic of the piece.
   * An iterator of coordinates that indicates the possible movements of the piece from the
   * origin.
   */
  Iterable<Coord> movement();


  /**
   * The tour algorithm.
   * The default implementation is a simple call to the overload with a null moves counter.
   * @param board the board to tour, mustn't be null
   */
  default boolean tour(final Board board) {
    return this.tour(board, null);
  }
```

```
  /**
   * The tour algorithm.
   * The default implementation is a backtracking brute-force algorithm, that suits all
   * the possible pieces.
   * @param board the board to tour, mustn't be null
   * @param moves a output counter for the number of movements calculated, possibly null
   */
  default boolean tour(final Board board, final Box<Long> moves) {
    if (board == null)
      throw new IllegalArgumentException("board mustn't be null");

    final IPiece piece = this;

    IntPredicate tour = new IntPredicate() {
      public boolean test(int move) {
        if (move == board.size + 1) // The board is complete.
          return true;

        if (moves != null)
          moves.value++;

        for (Coord s : board.steps(piece))
          if (board.getTile(s) == 0) { // unvisited tile.
            // Save the initial position, in case a backtrack is needed:
            Coord c = piece.getPosition();

            piece.setPosition(s);
            board.setTile(s, move);

            if (this.test(move + 1))
              return true;

            // backtrack:
            piece.setPosition(c);
            board.resetTile(s);
          }

        return false;
      }
    };

    board.setTile(this.getPosition(), 1); // Start the tour in the current position.

    if (moves != null)
      moves.value = 1L;

    return tour.test(2); // Attempt the tour from the start.
  }
}
```

Coesão

O módulo apresenta coesão funcional, cumprindo apenas o papel de representar, criar e manipular peças, além de implementar o algoritimo mais adequado para o passeio da peça.

Acoplamento

O módulo apresenta acoplamento de dados com o módulo padrão *IntPredicate*, e também com os módulos *Coord*, *Board* e *Box*.

4.2.2 Knight

Esta classe representa a peça do cavalo, incluindo sua mecânica de movimentos.

Interface

A classe implementa a interface IPiece. Além desta, possui a seguinte interface:

```
public class Knight implements IPiece {
  public Knight(Coord c); // Constructs a knight in the given coordinate.
}
```

Implementação

```
package dcc.gahag.chess.piece;

import java.util.List;
import dcc.gahag.chess.board.Coord;


/**
 * The knight piece.
 */
public class Knight implements IPiece {
  protected Coord _position;

  public Coord getPosition() {
    return this._position;
  }
  public void setPosition(Coord c) {
    this._position = c;
  }


  protected final Iterable<Coord> _movement = List.of(
    new Coord(2, 1),
    new Coord(1, 2),
    new Coord(-1, 2),
    new Coord(-2, 1),
    new Coord(-2, -1),
    new Coord(-1, -2),
    new Coord(1, -2),
    new Coord(2, -1)
  );

  public Iterable<Coord> movement() {
    return this._movement;
  }


  /**
   * Constructs a knight in the given coordinate.
   * @param c the coordinate for the knight to be placed, possibly null
   */
  public Knight(Coord c) {
    this.setPosition(c);
  }
}
```

Coesão

O módulo apresenta coesão funcional, cumprindo apenas o papel de representar e manipular o cavalo.

Acoplamento

O módulo apresenta acoplamento de dados com o módulo padrão *List*, e também com os módulos *Coord* e *IPiece*.

4.3 Main

O módulo principal. Implementa o ponto de entrada para a execução do passeio.

Interface

Este módulo apenas fornece a função `main`.

```
public final class Main {
  public static void main(String args[]);
}
```

Implementação

```
package dcc.gahag.chess;

import dcc.gahag.chess.board.Board;
import dcc.gahag.chess.board.Coord;
import dcc.gahag.chess.piece.Knight;
import dcc.gahag.chess.util.Box;
import dcc.gahag.chess.util.Threading;

public final class Main {
  public static void main(String args[]) {
    Board board = new Board(8);
    Knight knight = new Knight(Coord.random(board.bounds));
    Box<Long> moves = new Box<Long>();

    Coord position = knight.getPosition();
    System.out.printf("Initial position: (%d, %d)%n", position.x, position.y);

    if (knight.tour(board, moves))
      board.print(System.out);
    else
      System.err.println("Solution does not exist");

    System.out.println();

    System.out.printf("Total moves: %d.%n", moves.value);

    System.out.printf("User time: %.3f seconds.%n", Threading.userTime());
  }
}
```

Coesão

O módulo apresenta coesão funcional ao implementar apenas a função principal do programa, delegando as tarefas específicas para os outros módulos.

Acoplamento

O módulo apresenta acoplamento de dados com os módulos padrão *System.out* e *System.err*, e também com os módulos *Coord*, *Board*, *Knight*, *Box* e *Threading*.

4.4 Util

4.4.1 Box

Provê a técnica de *boxing* para valores de referência. Útil para passagem de parâmetro permitindo a mutabilidade.

Interface

```
public class Box<T> {
  public T value;  // The boxed value.

  public Box();        // Construct a box, with a null value.
  public Box(T value); // Construct a box with the given value.
}
```

Implementação

```
package dcc.gahag.chess.util;


/**
 * A mutable boxed reference type.
 * This type allows mutation of values passed by parameter.
 */
public class Box<T> {
  public T value;

  /**
   * Construct a box, with a null value.
   */
  public Box() {
    this.value = null;
  }

  /**
   * Construct a box with the given value.
   */
  public Box(T value) {
    this.value = value;
  }
}
```

Coesão

O módulo apresenta coesão funcional, sendo genérico e implementando apenas a funcionalidade de *boxing*.

Acoplamento

O módulo não possui acoplamento com nenhum outro módulo.

4.4.2 Threading

Utilidades relacionadas à *threads*.

Interface

```
public final class Threading {
  public static double userTime(); // Gets the current thread's user time in seconds.
}
```

Implementação

```
package dcc.gahag.chess.util;

import java.lang.management.ManagementFactory;

/**
 * Static class for threading utils.
 */
public final class Threading {
  private Threading() { }

  /**
   * Gets the current thread's user time in seconds.
   */
  public static double userTime() {
    return ManagementFactory.getThreadMXBean().getCurrentThreadUserTime() / 1000000000.0;
  }
}
```

Coesão

O módulo apresenta coesão funcional, implementando apenas funcionalidades estáticas relativas à *threads*.

Acoplamento

O módulo apresenta acoplamento de dados com o módulo padrão *Managementfactory*.

4.4.3 FilterIterator

Um iterador para filtrar os elementos de outro iterador.

Interface

A classe implementa a interface Iterator<T>, não permitindo a chamada do método `remove`. Além desta, possui a seguinte interface:

```
public class FilterIterator<T> implements Iterator<T> {
  // Construct a FilterIterator from an iterator and a predicate:
  public FilterIterator(final Iterator<? extends T> it, final Predicate<? super T> p);
}
```

Implementação

```
package dcc.gahag.chess.util;

import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.function.Predicate;

/**
 * An iterator to filter the values of other iterator.
 * Event if the specified iterator supports removing, the FilterIterator doesn't.
 */
public class FilterIterator<T> implements Iterator<T> {
  protected final Iterator<? extends T> it;
  protected final Predicate<? super T> predicate;

  protected T _next;
  protected boolean _nextSet = false;
```

```
  /**
   * Construct a FilterIterator from an iterator and a predicate.
   * @param  it the iterator to be filtered, mustn't be null
   * @param  f  the predicate, possibly null
   */
  public FilterIterator(final Iterator<? extends T> it, final Predicate<? super T> p) {
    if (it == null)
      throw new IllegalArgumentException("Iterator mustn't be null");

    this.it = it;
    this.predicate = p;
  }

  public boolean hasNext() {
    return this._nextSet || this.setNext();
  }

  public T next() {
    if (!this._nextSet && !this.setNext())
      throw new NoSuchElementException();

    this._nextSet = false;
    return this._next;
  }

  protected boolean setNext() {
    while (this.it.hasNext()) {
      final T n = this.it.next();

      if (this.predicate != null && this.predicate.test(n)) {
        this._next = n;
        this._nextSet = true;
        return true;
      }
    }

    return false;
  }
}
```

Coesão

O módulo apresenta coesão funcional, sendo genérico e implementando apenas a funcionalidade de filtrar outro iterador.

Acoplamento

O módulo apresenta acoplamento de dados com os módulos padrão *Predicate* e *Iterator*.

4.4.4 TransformIterator

Um iterador para transformar os elementos de outro iterador.

Interface

A classe implementa a interface *Iterator<T>*, delegando o método `remove` para o iterador dado. Além desta, possui a seguinte interface:

```
public class TransformIterator<T, U> implements Iterator<U> {
  // Construct a TransformIterator from an iterator and a transform function:
  public TransformIterator(Iterator<? extends T> it, Function<? super T, ? extends U> f);
}
```

Implementação

```
package dcc.gahag.chess.util;


import java.util.Iterator;
import java.util.function.Function;


/**
 * An iterator to transform the values of other iterator.
 */
public class TransformIterator<T, U> implements Iterator<U> {
  protected final Iterator<? extends T> it;
  protected final Function<? super T, ? extends U> f;


  /**
   * Construct a TransformIterator from an iterator and a transform function.
   * @param  it the iterator to be transformed, mustn't be null
   * @param  f  the transformer function, mustn't be null
   */
  public TransformIterator(Iterator<? extends T> it, Function<? super T, ? extends U> f) {
    if (it == null)
      throw new IllegalArgumentException("Iterator mustn't be null");

    if (f == null)
      throw new IllegalArgumentException("Function mustn't be null");

    this.it = it;
    this.f = f;
  }


  public boolean hasNext() {
    return this.it.hasNext();
  }

  public U next() {
    return this.f.apply(this.it.next());
  }

  public void remove() {
    this.it.remove();
  }
}
```

Coesão

O módulo apresenta coesão funcional, sendo genérico e implementando apenas a funcionalidade de transformar os valores de outro iterador.

Acoplamento

O módulo apresenta acoplamento de dados com os módulos padrão *Function* e *Iterator*.

5 Pesquisa

5.1 Coesão

Coesão é a relação entre os elementos e operações de um módulo. Seguem os tipos de coesão, do pior até o melhor, sendo a melhor a coesão funcional. [1]

5.1.1 Coincidental

Na coesão coincidental, há pouca ou nenhuma relação construtiva entre os elementos de um módulo. Exemplo:

```
class Angu {
  public static int acharPadrão(String texto, String padrão) {
    // ...
  }
  public static int média(Vector números) {
    // ...
  }
  public static outputStream abreArquivo(string nomeArquivo) {
    // ...
  }
}

class Mexido extends Angu { // quer aproveitar código de Angu
  // ...
}
```

5.1.2 Lógica

Um módulo faz um conjunto de funções relacionadas, uma das quais é escolhida através de um parâmetro ao chamar o módulo. Semelhante ao acoplamento de controle.

```
public void do(int flag) {
  switch(flag) {
  case ON:
    // coisas para tratar de ON
    break;
  case OFF:
    // coisas para tratar de OFF
    break;
  case FECHAR:
    // coisas para tratar de FECHAR
    break;
  case COR:
    // coisas para tratar de COR
    break;
  }
}
```

5.1.3 Temporal

Elementos estão agrupados no mesmo módulo porque são processados no mesmo intervalo de tempo.

```
public void inicializaDados() {
  windowSize = "200,400";
  angu.localização = "/usr/local/lib/java";
}
```

Código 1: Método de inicialização que provê valores padrão não relacionados

```
[Spacing]
LineSpacing=150%
MatrixRowSpacing=150%
MatrixColSpacing=100%
SuperscriptHeight=45%
SubscriptDepth=25%
LimHeight=25%
NumerHeight=35%
DenomDepth=100%
FractBarOver=1pt
FenceOver=1pt
SpacingFactor=100%
MinGap=8%
PrimeHeight=45%
```

Código 2: Arquivo de configuração típico

5.1.4 Procedural

Associa elementos de acordo com seus relacionamentos procedurais ou algorítmicos. Um módulo procedural depende muito da aplicação sendo tratada. Junto com a aplicação, o módulo parece razoável, mas sem este contexto, o módulo é estranho e difícil de entender. Não é possível compreender o módulo sem entender o programa, e as condições que existem quando o módulo é chamado.

5.1.5 Comunicação

Todas as funcionalidades de um módulo operam no mesmo conjunto de dados e/ou produzem o mesmo tipo de dado de saída.

Não deveria ocorrer em sistemas OO que usam polimorfismo (classes diferentes para fazer tratamentos diferentes nos dados).

5.1.6 Sequencial

A saída de uma operação do módulo serve de entrada para a próxima operação do mesmo módulo. O problema é que, mesmo que o processamento se dê de forma sequencial, as operações em si não necessariamente apresentam funcionalidades diretamente relacionadas.

5.1.7 Funcional

A melhor das formas de coesão. Um módulo tem coesão funcional se suas operações possuem forte relação funcional, no que se refere à funcionamento e propósito.

5.2 Acoplamento

Acoplamento é a relação de dependência entre dois ou mais módulos. Seguem os tipos de acoplamento, do pior até o melhor, sendo o melhor o acoplamento de dados. [2]

5.2.1 Conteúdo

O módulo depende dos dados ou operações internas de outro módulo, violando o princípio de ocultação da informação.
Exemplos:

- O módulo faz um branch para um código interno de outro módulo.
- O Módulo refere-se à dados internos de outro módulo.

5.2.2 Comum

O módulo compartilha dados globais com outro módulo.
Exemplos:

- Os módulos compartilham variáveis globais.
- Os módulos compartilhando o mesmo banco de dados.
- Os módulos compartilham o mesmo arquivo em disco.

5.2.3 Controle

O módulo controla o fluxo de execução ou o propósito de outro módulo, como por exemplo através da passagem de uma flag que determina a operação a ser realizada. Semelhante à coesão lógica.

5.2.4 Carimbo

Os módulos compartilham uma estrutura de dados composta, mas utilizam apenas pequenas partes (possivelmente distintas) dela. Idealmente, os módulos deveriam ter acesso apenas aos dados necessários para a sua operação.

5.2.5 Dados

Os módulos interagem através do compartilhamento somente dos dados necessários.
Exemplos:

- Passar um inteiro para uma função que computa uma raiz quadrada.
- Passar uma string para uma função que imprime na saída padrão.

6 Referências

[1] Dr. Jacques Philippe Sauvé. *Padrão para atribuir responsabilidades: Alta Coesão.* URL: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/pat/altacoesao.htm (acesso em 28/09/2018).

[2] Wikipedia. *Coupling (computer programming).* URL: https://en.wikipedia.org/wiki/Coupling_(computer_programming) (acesso em 28/09/2018).