HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG KHOA CÔNG NGHỆ THÔNG TIN 1

—&-

BÁO CÁO THỰC TẬP CƠ SỞ

Đề tài: Java Core

Giảng viên hướng dẫn : TS.Nguyễn Tất Thắng

Sinh Viên thực hiện : Nguyễn Đức Anh

Mã Sinh Viên : B20DCCN056

Lớp : D20CQCN08 - B

Hệ đại học : Chính quy

Niên khóa : 2020 - 2025

HÀ NỘI, 05/2023

I. Phần mở đầu

1. Giới thiệu về ngôn ngữ lập trình java

- Java là một trong những ngôn ngữ lập trình hướng đối tượng. Nó được sử dụng trong phát triển phần mềm, trang web, game hay ứng dụng trên các thiết bị di động.

- Đặc điểm của ngôn ngữ lập trình Java:
    - Tương tự C++, hướng đối tượng hoàn toàn
    - Độc lập phần cứng và hệ điều hành
    - Ngôn ngữ thông dịch
    - Cơ chế thu gom rác tự động
    - Đa luồng
    - Tính an toàn và bảo mật

2. Mục đích

- Nâng cao kỹ năng lập trình Java
- Nắm vững Java core
- Làm ra được sản phẩm sử dụng Java core

3. Phạm vi tìm hiểu.

- Java core
- Java swing

4. Phương pháp

- Tìm hiểu lý thuyết kết hợp bài tập vận dụng
- Tự tạo sản phẩm cuối khóa sử dụng Java core

II. Phần nội dung

1. Ôn và nắm chắc lại kiến thức

1.1 Kĩ năng lập trình

- Tìm hiểu môi trường lập việc Java: các khái niệm JDK, JVM, JRE ….
- Các kiểu dữ liệu cơ bản trong java: int, long, char, double, String
- Các toán tử trong java gồm toán tử logic và toán tử điều kiện
- Vòng lặp trong java
- Mảng trong java
- Các syntax cơ bản khác: Wrapper class, String class, Math, class

- Các syntax về OOP: class, đóng gói, kế thừa, đa hình, trừu tượng, ghi đè phương thức, nạp chồng phương thức

1.2 Collection, Collections

- Tập trung tìm hiểu các Collection phổ biến: List, ArrayList, Map, Set
- Một số method thường dung trong collections: sort, …
- Tìm hiểu them các Collection khác như: Stack, Queue, PriorityQueue

1.3 Exception

- Tìm hiểu về cách xử lý Exception: từ khóa try, catch, finally, throw
- Tìm hiểu một số exception đơn giản cơ bản
- Tự custom Exception class

1.4 Java Swing

- Tìm hiểu về JFrame, JTextField, JButton, JPanel, …
- Các layout, cách sử dụng

2. Bài tập

2.1 Bài tập kĩ năng lập trình

- Viết chương trình mô phỏng thuật toán DFS
- Đề bài

SSAM419A - THUẬT TOÁN DFS

Cho đồ thị vô hướng G=(V, E). Hãy thực hiện thuật toán duyệt đồ thị DFS bắt đầu tại một đỉnh u ∈ V.

Input

Dòng đầu tiên gồm một số nguyên T (1 ≤ T ≤ 20) là số lượng bộ test.

Tiếp theo là T bộ test, mỗi bộ test có dạng sau:

- Dòng đầu tiên gồm 3 số nguyên N=|V|, M=|E|, u (1 ≤ N ≤ 10^3, 1 ≤ M ≤ 10^5, 1 ≤ u ≤ N).
- M dòng tiếp theo, mỗi dòng gồm 2 số nguyên a, b (1 ≤ a, b ≤ N, a ≠ b) tương ứng cạnh nối hai chiều từ a tới b.
- Dữ liệu đảm bảo giữa hai đỉnh chỉ tồn tại nhiều nhất một cạnh nối.

Output

Với mỗi bộ test, in ra trên một dòng theo thứ tự các đỉnh được duyệt trong quá trình duyệt đồ thị bằng thuật toán DFS bắt đầu tại đỉnh u.

Ví dụ

Input	Output
1 5 5 3 1 2 1 3 2 4 3 5 4 5	3 1 2 4 5

- Biểu diễn đồ thị đã cho dưới dạng danh sách kề
- Sử dụng dfs để duyệt, nếu tại đỉnh u đang đứng mà có cạnh nối trực tiếp tới đỉnh v thì sẽ duyệt đỉnh v và đánh dấu nó đã được duyệt

```
no usages  ducanh *
public class Main {
    no usages
    static ArrayList<ArrayList<Integer>> g;
    no usages
    static Boolean visited[];
    1 usage  new *
    public static void init() {
        g = new ArrayList<>();
        visited = new Boolean[1005];
        // init
        for(int i = 0; i < 1005; i++) {
            g.add(new ArrayList<>());
            visited[i] = false;
        }
    }

    2 usages  new *
    public static void dfs(int u) {
        visited[u] = true;
        System.out.print(u + " ");
        for(Integer i: g.get(u)) {
            if(visited[i] == false) {
                dfs(i);
            }
        }
    }
```

- Hàm main xử lý Input:

```
no usages  ducanh *
public static void main(String[] args) {
    init();
    Scanner sc = new Scanner(System.in);
    int t = sc.nextInt();
    for(int i = 1; i <= t; i++) {
        int n = sc.nextInt(), m = sc.nextInt(), start = sc.nextInt();
        for(int j = 1; j <= m; j++) {
            int u = sc.nextInt(), v = sc.nextInt();
            g.get(u).add(v);
            g.get(v).add(u);
        }
        dfs(start);
        System.out.println();
    }
}
```

- Link source code: https://ideone.com/2T5rQQ

2.2 Tự định nghĩa 1 Exception

- Khai báo 1 class AgeException. Nếu tuổi không hợp lệ sẽ xảy ra 1 Exception

```
2 usages
class AgeException extends Exception {
    1 usage
    AgeException(String message) {
        super(message);
    }
}
```

- Khai báo 1 class Student. Với các thuộc tính tên, tuổi, cùng với đó là các phương thức setter, getter kèm theo. Trong đó phương thức setAge, nếu tuổi nhỏ hơn < 18 sẽ trả về 1 Exception là không đủ tuổi. Sử dụng

throww để định nghĩa 1 hàm có thể sẽ xảy ra ngoại lệ, và throw để ném ra ngoại lệ trong trường hợp xảy ra lỗi

```
class Student {
    3 usages
    private String name;
    3 usages
    private int age;

    1 usage
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    no usages
    public String getName() {
        return this.name;
    }

    no usages
    public void setName(String name) {
        this.name = name;
    }

    no usages
    public int getAge() {
        return this.age;
    }

    1 usage
    public void setAge(int age) throws AgeException {
        if(age < 18) {
            throw new AgeException("Khong du tuoi");
        }
        this.age = age;
    }
}
```

- Hàm main

```java
public class Main {
    no usages
    public static void main(String[] args) {
        Student student = new Student( name: "DucAnh",  age: 21);
        try {
            student.setAge(15);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- Console log ngoại lệ:

```
"C:\Program Files\Java\jdk1.8.0_341\bin\java.exe" ...
AgeException: Khong du tuoi
    at Student.setAge(Main.java:36)
    at Main.main(Main.java:5)

Process finished with exit code 0
```

2.3 Bài tập về class, sử dụng dụng design parttern dependency injection

- Tạo intefercace sort và 2 class QuickSort, BubbleSort để ghi đè cụ thể cách sort

```
4 usages  2 implementations
interface AlgorithmSort {
    1 usage  2 implementations
    void sort(int []a);
}

no usages
class QuickSort implements AlgorithmSort {
    1 usage
    @Override
    public void sort(int []a) {
        // Code QuickSort
    }
}

1 usage
class BubbleSort implements AlgorithmSort {
    1 usage
    @Override
    public void sort(int []a) {
        // Code BubbleSort
    }
}
```

- Hàm main

```
class ManagerInteger {
    1 usage
    private int []a = new int[10];
    2 usages
    private AlgorithmSort algorithmSort;
    1 usage
    public ManagerInteger(AlgorithmSort algorithmSort) {
        this.algorithmSort = algorithmSort;
    }

    1 usage
    public void sortMyInteger() {
        algorithmSort.sort(a);
    }
}
no usages
public class Main {
    no usages
    public static void main(String[] args) {
        ManagerInteger managerInteger = new ManagerInteger(new BubbleSort());
        managerInteger.sortMyInteger();
    }
}
```

2.4 Bài tập về class, sử dụng dụng factory parttern

- Tạo 1 interface là thể hiện của các hình học

```
interface Shape {
    2 usages  2 implementations  ducanh
    void draw();
}
```

- Tạo lớp Shape Factory để tạo các hình học 1 cách dễ dàng hơn

```
3 usages  ducanh
enum ShapeType {
    2 usages
    RECTANGLE, CIRCLE
}
```

```
2 usages  ducanh
class ShapeFactory {
    2 usages  ducanh
    public static Shape init(ShapeType type) {
        switch (type) {
            case RECTANGLE:
                return new Rectangle();
            case CIRCLE:
                return new Circle();
            default:
                return null;
        }
    }
}
```

- Tạo 2 lớp cụ thể

```
class Rectangle implements Shape {
    2 usages  ducanh
    @Override
    public void draw() {
        System.out.println("Rectangle");
    }
}

1 usage  ducanh
class Circle implements Shape {
    2 usages  ducanh
    @Override
    public void draw() {
        System.out.println("Circle");
    }
}
```

- Hàm main

```
no usages  ducanh
public class FactoryPattern {
    no usages  ducanh
    public static void main(String[] args) {
        Shape shape = ShapeFactory.init(ShapeType.CIRCLE);
        shape.draw();

        shape = ShapeFactory.init(ShapeType.RECTANGLE);
        shape.draw();
    }
}
```

3. Bài tập cuối khóa

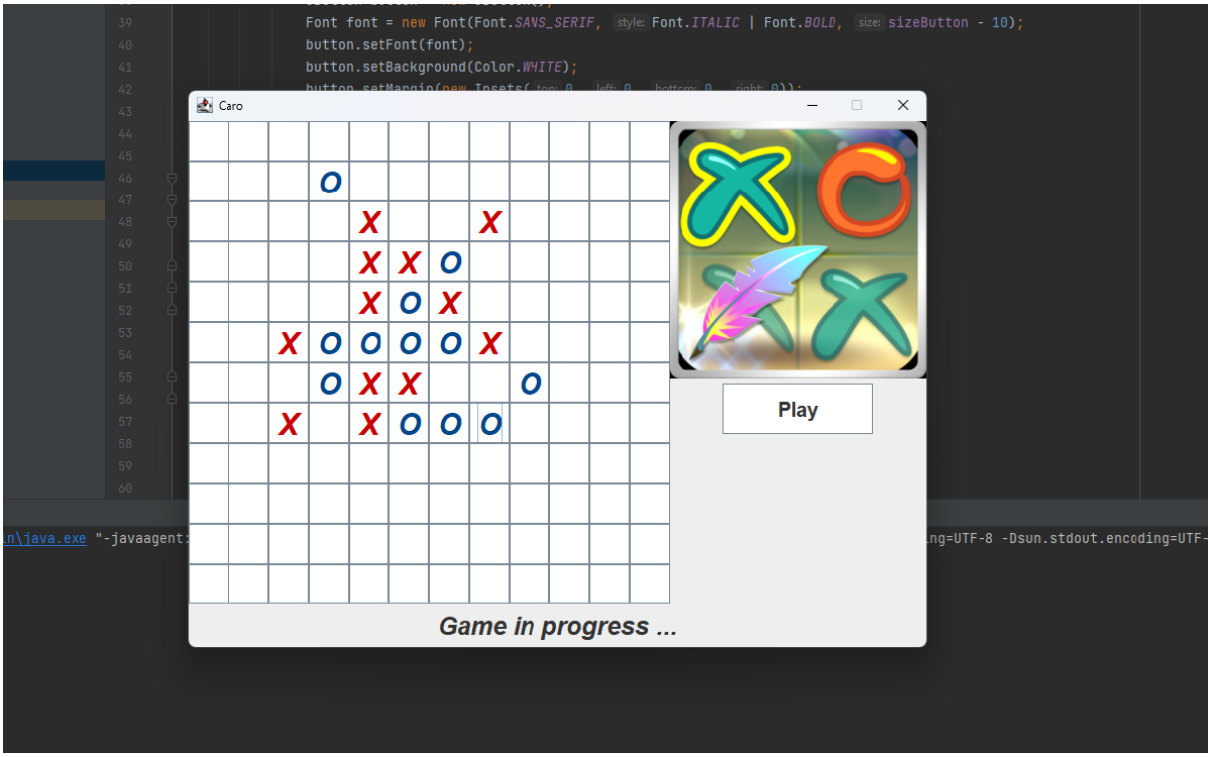
- **Chủ đề:** Game Caro sử dụng thuật toán cắt tỉa alpha beta pruning.

- **Luật chơi:** Bàn cờ 12x12 ô, mỗi lượt người chơi sẽ đánh X và máy tính sẽ đánh O. Bên nào tạo thành 1 dãy liên tiếp 5 ô giống nhau trước sẽ dành chiến thắng theo các hướng bất kì ngang, dọc, chéo.

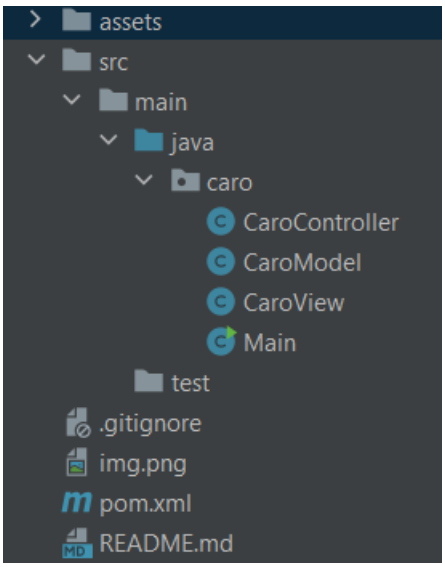
- **Công nghệ:** JavaCore, Java Swing

- **Thuật toán: Alpha beta pruning** là một thuật toán tìm kiếm nâng cao của minimax, thuật toán này làm giảm số lượng các node cây được đánh giá bởi thuật toán minimax trong cây tìm kiếm. Thuật toán này dựa theo tìm kiếm đối nghịch trong một số trò chơi với máy (Tic-tac-toe, Cờ vua, …).

- **Giao diện:**



- **Cấu trúc thư mục:**



- **Class Main:** Chứa hàm Main của chương trình và sẽ gọi instance của ClassView

```
public class Main {
    DucAnh
    public static void main(String[] args) {
        CaroView.getInstance();
    }
}
```

- **ClassView**: Kế thừa class JFrame để xây dựng UI cho dự án.

```
public class CaroView extends JFrame {
    private CaroController caroController;
    private JButton[][] buttons;
    private int sizeButton = 40;
    private int[] preBotMove = {0, 0};
    private boolean isEndGame = false;
    private boolean isWaiting = false;
    private static CaroView caroView = null;
    private JLabel labelNotification = null;
```

Hàm khởi tạo của class CaroView sẽ tạo 144 JButton tương ứng với bàn cờ 12x12. Tạo JButton dùng để chơi lại, JLabel hiển thị xem trò chơi đang diễn ra hay người chơi/máy đã thắng …

```
JPanel mainView = new JPanel();
mainView.setLayout(new BorderLayout());
buttons = new JButton[sizeMatrix + 1][sizeMatrix + 1];
JPanel panelButtons = new JPanel();
panelButtons.setLayout(new GridLayout(sizeMatrix, sizeMatrix));
panelButtons.setSize(sizeMatrix * sizeButton, sizeMatrix * sizeButton);
for(int i = 1; i <= sizeMatrix; i++) {
    for(int j = 1; j <= sizeMatrix; j++) {
        JButton button = new JButton();
        Font font = new Font(Font.SANS_SERIF, Font.ITALIC | Font.BOLD, sizeButton - 10);
        button.setFont(font);
        button.setBackground(Color.WHITE);
        button.setMargin(new Insets(0, 0, 0, 0));
        button.setPreferredSize(new Dimension(sizeButton, sizeButton));
        int finalI = i;
        int finalJ = j;
        button.addActionListener(e -> {
            if(buttons[finalI][finalJ].getText().length() == 0) {
                if(!isEndGame) {
                    caroController.playerMove(finalI, finalJ);
                }
            }
        });
        panelButtons.add(button);
        buttons[i][j] = button;
    }
}
```

Mỗi JButton sẽ nhận sự kiện click chuột từ người chơi và kiểm tra xem nếu text của Button đó đang rỗng (length() = 0) thì sẽ chạy phương thức playMove của class CaroController.

```
DucAnh +1
public void playerMove(int x, int y) {
    // Event Player Click
    CaroModel.getInstance().playerMove(x, y);
    CaroView.getInstance().playerMove(x, y);
    Integer checkStatusGame = CaroModel.getInstance().checkStatusGame(x, y);
    if(checkStatusGame != 0) {
        CaroView.getInstance().endGame("You Win !");
    }
    else {
        this.botMove();
    }
}
DucAnh +1
public void botMove() {
    // Event Bot execute
    int bestMove[] = CaroModel.getInstance().findBestMove();
    CaroModel.getInstance().botMove(bestMove[0], bestMove[1]);
    CaroView.getInstance().botMove(bestMove[0], bestMove[1]);
    int checkStatusGame = CaroModel.getInstance().checkStatusGame(bestMove[0], bestMove[1]);
    if(checkStatusGame != 0) {
        CaroView.getInstance().endGame("Bot Win !");
    }
}
```

Ở phương thức playerMove() của CaroController sẽ gọi playerMove() của CaroModel nhằm đánh dấu tọa độ a[x][y] = -1 (đại diện cho giá trị của ô đó là -1, là ô mà player đã đánh vào) và sẽ gọi playerMove() của CaroView nhằm đánh dấu button[x][y] = “O”. Đồng thời sẽ kiểm tra xem trạng thái của ván đấu đó sau khi player đã đánh là gì. Nếu trạng thái vẫn là 0, có nghĩa vẫn đang hòa, chưa có bên nào thắng, thì sẽ gọi phương thức botMove() để cho bot đánh.

Với phương thức botMove() có nghĩa là máy tính xử lý khi tới lượt. Đầu tiên sẽ gọi findBestMove() của CaroModel, đây thực chất là tìm vị trí mà tìm lại lợi thế nhất cho máy tính, ở bước này sẽ sử dụng cắt tỉa alpha beta prunning

DucAnh

```
public int[] findBestMove() {
    long bestScore = Integer.MIN_VALUE;
    long alpha = Integer.MIN_VALUE;
    int x = -1, y = -1;
    for(int i = 1; i <= sizeMatrix; i++) {
        for(int j = 1; j <= sizeMatrix; j++) {
            if(a[i][j] == 0) {
                a[i][j] = 1;
                long scoreMove = evaluateMove(i, j);
                long score = alphaBeta(false, 2, alpha, Integer.MAX_VALUE, i, j);
                a[i][j] = 0;
                if(bestScore < score + scoreMove) {
                    bestScore = score + scoreMove;
                    x = i;
                    y = j;
                }
                alpha = Math.max(alpha, bestScore);
            }
        }
    }
    return new int[]{x, y};
}
```

DucAnh +1

```
private long alphaBeta(boolean isBot, int depth, long alpha, long beta, int preX,int preY) {
    int scoreEvaluate = checkStatusGame(preX, preY);
    if(scoreEvaluate == Integer.MIN_VALUE || scoreEvaluate == Integer.MAX_VALUE || depth >= maxDepth) {
        return scoreEvaluate;
    }
    long bestScore = isBot ? Integer.MIN_VALUE : Integer.MAX_VALUE;
    for(int i = 1; i <= sizeMatrix; i++) {
        for(int j = 1; j <= sizeMatrix; j++) {
            if (a[i][j] == 0) {
                a[i][j] = isBot ? 1 : -1;
                long score = alphaBeta(!isBot, depth + 1, alpha, beta, i, j);
                a[i][j] = 0;
                if(isBot) {
                    bestScore = Math.max(score , bestScore);
                    alpha = Math.max(alpha, bestScore);
                }
                else {
                    bestScore = Math.min(score, bestScore);
                    beta = Math.min(beta, bestScore);
                }
                if(beta <= alpha) {
                    return bestScore;
                }
            }
        }
    }
    return bestScore;
}
```

Nếu node hiện tại đang là MIN (isBot) thì sẽ lấy max tất cả các node con liền kề của nó, và ngược lại nếu node hiện tại đang là MAX (isPlayer) thì sẽ lấy min tất cả các node con liền kề.

Sau khi tìm được tọa độ x,y tối ưu nhất cho máy qua findBestMove() thì sẽ gọi botMove() qua CaroView và botMove() qua CaroModel. Tương tự giống plaerMove() đã nói ở trên, nhằm gán mảng a[x][y] = 1 tượng trưng cho giá trị mà bot đã đánh, và button[x][y] = "X" để thể hiện cho người chơi xem.

Cụ thể về các biến cần lưu của CaroModel gồm:

- **maxDepth**: là độ sâu tối đa duyệt tới để tránh bị time limited.

- **winningLength**: là độ dài dành chiến thắng nếu có 5 ô giống nhau nằm liên tiếp.

- **sizeMatrix:** là kích thước của bảng

- **a**: là mảng 2 chiều đại diện cho bảng, với các giá trị có thể nhận là -1, 0 ,1.

- **directions**: là vector thể hiện cho 4 hướng đi (dọc, ngang, chéo chính, chéo phụ)

```
public class CaroModel {
    private final Map<String, Integer> transpositionTable = new HashMap<>();
    private final int maxDepth = 6;
    private final int winningLength = 5;
    private final int sizeMatrix = 12;
    private final int[][] a = new int[sizeMatrix + 1][sizeMatrix + 1];
    private final int[][] directions = {{0,-1}, {-1,0}, {-1,-1}, {-1,1}};
    private static CaroModel caroModel = null;
    DucAnh
```

Ngoài ra còn có các phương thức khác như kiểm tra xem trạng thái của ván đầu đã kết thúc hay chưa

```java
private int checkStatusGame(int x, int y, int[] directionModel) {
    if(!isValidMove(x, y)) {
        return 0;
    }
    int countPoint = 1;
    for(int i = -1; true; i--) {
        int X = x + directionModel[0] * i;
        int Y = y + directionModel[1] * i;
        if(isValidMove(X,Y) && a[X][Y] == a[x][y]) {
           countPoint++;
        }
        else {
            break;
        }
    }
    for(int i = +1; true; i++) {
        int X = x + directionModel[0] * i;
        int Y = y + directionModel[1] * i;
        if(isValidMove(X,Y) && a[X][Y] == a[x][y]) {
            countPoint++;
        }
        else {
            break;
        }
    }
    if(countPoint == 5) {
        return a[x][y] == 1 ? Integer.MAX_VALUE : Integer.MIN_VALUE;
    }
    return 0;
}
```

và 1 số các phương thức khác như reset lại ván đấu, cách bố trí các Layout của giao diện với GridLayout, ….

Link source code: [DucAnhNg2002/Caro (github.com)]

4. Tổng kết

- Nắm chắc các kiến thức về java core

- Hoàn thành nội dung của môn học Thực tập cơ sở
- Hoàn thành bản báo cáo ghi lại các kiến thức đã học được và project luyện tập các kiến thức đó