BÀI THỰC HÀNH HỌC PHẦN CÔNG NGHỆ JAVA

Ngày thực hành: 12/09/2025

Lớp: 68PM1, 68PM2

Các nội dung chính bao gồm: Collections (ArrayList), Generics (method), Exceptions.

Mục tiêu

- Giúp sinh viên làm quen với **ArrayList** trong Java: thêm, sửa, xóa, duyệt phần tử.
- Hiểu cách viết **method generic** để xử lý dữ liệu linh hoạt.
- Nắm vững **cách kiểm soát Exceptions** trong lập trình Java.

Yêu cầu thực hành:

Sinh viên đọc yêu cầu từng bài tập, lập trình theo mã nguồn gợi ý, thực thi chương trình và review lại.

Bài tập 1 – ArrayList cơ bản

Viết chương trình quản lý danh sách tên sinh viên:

- Thêm tên sinh viên vào danh sách.
- In ra toàn bộ danh sách.
- Xóa một sinh viên theo tên.
- Tìm kiếm một sinh viên trong danh sách.

Mã nguồn tham khảo:

```
import java.util.ArrayList;
import java.util.Scanner;


public class StudentList {
    public static void main(String[] args) {
        ArrayList<String> students = new ArrayList<>();
        Scanner sc = new Scanner(System.in);


        students.add("An");
        students.add("Bình");
```

```
        students.add("Chi");

        System.out.println("Danh sách ban đầu: " + students);

        // Thêm sinh viên
        System.out.print("Nhập tên sinh viên mới: ");
        String newStudent = sc.nextLine();
        students.add(newStudent);

        System.out.println("Danh sách sau khi thêm: " + students);

        // Xóa sinh viên
        System.out.print("Nhập tên sinh viên cần xóa: ");
        String removeStudent = sc.nextLine();
        if (students.remove(removeStudent)) {
            System.out.println("Đã xóa " + removeStudent);
        } else {
            System.out.println("Không tìm thấy " + removeStudent);
        }
        System.out.println("Danh sách hiện tại: " + students);
    }
}
```

Bài tập 2 – Generics method

Tìm giá trị lớn nhất trong danh sách (Generic Method)

Mục tiêu

- Hiểu và sử dụng **Generic Method** trong Java.
- Áp dụng Generics để viết **hàm tái sử dụng,** không phụ thuộc vào kiểu dữ liệu cụ thể.
- Thấy được lợi ích của Generics trong tình huống thực tế.

Yêu cầu bài tập

1. Viết một **phương thức generic** findMax có tham số là ArrayList<T> với ràng buộc T extends Comparable<T>.
2. Phương thức sẽ trả về **giá trị lớn nhất** trong danh sách.
3. Trong main:
    - Tạo một danh sách điểm số (Integer).
    - Tạo một danh sách tên sinh viên (String).
    - Gọi findMax để tìm điểm cao nhất và tên đứng cuối theo thứ tự từ điển.

Mã nguồn tham khảo:

```
import java.util.ArrayList;


public class GenericMaxExample {


    // Generic method để tìm phần tử lớn nhất
    public static <T extends Comparable<T>> T findMax(ArrayList<T> list) {
        if (list == null || list.isEmpty()) {
            return null;
        }
        T max = list.get(0);
        for (T element : list) {
            if (element.compareTo(max) > 0) {
                max = element;
            }
        }
        return max;
    }


    public static void main(String[] args) {
        // Danh sách điểm số
```

```
        ArrayList<Integer> scores = new ArrayList<>();
        scores.add(85);
        scores.add(92);
        scores.add(76);
        scores.add(98);

        Integer maxScore = findMax(scores);
        System.out.println("Điểm cao nhất: " + maxScore);

        // Danh sách tên sinh viên
        ArrayList<String> students = new ArrayList<>();
        students.add("An");
        students.add("Bình");
        students.add("Chi");
        students.add("Dũng");

        String lastStudent = findMax(students);
        System.out.println("Tên đứng cuối theo từ điển: " + lastStudent);
    }
}
```

Giải thích

- **<T extends Comparable<T>>**: ràng buộc để đảm bảo các kiểu dữ liệu truyền vào có thể so sánh được (ví dụ: Integer, Double, String…).
- **findMax**: viết một lần, dùng cho nhiều kiểu dữ liệu khác nhau.
- Ứng dụng thực tế: tìm điểm số cao nhất, mức lương cao nhất, sản phẩm có giá cao nhất, tên lớn nhất theo alphabet…

Bài tập 3: Exceptions (Ngoại lệ)

3.1 Làm việc với chia số trong Java (try-catch-finally)

Yêu cầu bài tập

1. Viết chương trình yêu cầu người dùng nhập 2 số nguyên.
2. Thực hiện phép chia số thứ nhất cho số thứ hai.
3. Nếu số thứ hai = 0 → xảy ra lỗi chia cho 0, cần dùng **try – catch** để bắt ngoại lệ.
4. Trong khối **finally**, in ra thông báo "Kết thúc phép tính".

Mục tiêu

- Hiểu cơ bản về **ngoại lệ (Exception)** trong Java.
- Biết cách sử dụng **try – catch** để kiểm soát lỗi.
- Biết cách sử dụng **finally** để chạy đoạn code bắt buộc, dù có ngoại lệ hay không.

Yêu cầu bài tập

1. Viết chương trình yêu cầu người dùng nhập 2 số nguyên.
2. Thực hiện phép chia số thứ nhất cho số thứ hai.
3. Nếu số thứ hai = 0 → xảy ra lỗi chia cho 0, cần dùng **try – catch** để bắt ngoại lệ.
4. Trong khối **finally**, in ra thông báo "Kết thúc phép tính".

Mã nguồn tham khảo:

```
import java.util.Scanner;


public class DivisionExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);


        try {
            System.out.print("Nhập số thứ nhất: ");
            int a = sc.nextInt();


            System.out.print("Nhập số thứ hai: ");
            int b = sc.nextInt();
```

```
            int result = a / b; // Có thể gây lỗi chia cho 0
            System.out.println("Kết quả: " + result);

        } catch (ArithmeticException e) {
            System.out.println("Lỗi: Không thể chia cho 0!");
        } catch (Exception e) {
            System.out.println("Đã xảy ra lỗi: " + e.getMessage());
        } finally {
            System.out.println("Kết thúc phép tính.");
        }
    }
}
```

Giải thích:

try: chứa đoạn code có thể gây lỗi (ở đây là phép chia a / b).

catch: xử lý lỗi cụ thể (ở đây là ArithmeticException khi chia cho 0).

finally: luôn chạy, bất kể có ngoại lệ hay không → dùng để thông báo, đóng file, đóng kết nối cơ sở dữ liệu,…

3.2 **Quản lý tài khoản ngân hàng với Exceptions**

Mục tiêu

- Biết cách **tự định nghĩa và ném ngoại lệ** trong một phương thức.
- Hiểu cách **xử lý ngoại lệ bằng try-catch**.
- Biết cách **dùng từ khóa finally** để giải phóng hoặc thông báo tài nguyên.

Yêu cầu bài tập

1. Xây dựng class BankAccount có các thuộc tính: accountNumber, balance.
2. Viết phương thức:

- deposit(double amount) → nếu số tiền nạp ≤ 0 thì **ném ngoại lệ.**
- withdraw(double amount) → nếu số tiền rút lớn hơn số dư thì **ném ngoại lệ.**

3. Trong main, tạo một tài khoản, thử nạp và rút tiền với các tình huống hợp lệ + không hợp lệ.
4. Sử dụng try-catch để xử lý ngoại lệ.
5. Sử dụng finally để in ra thông báo kết thúc giao dịch (dù có lỗi hay không).

Mã nguồn tham khảo:

```
// Ngoại lệ tùy chỉnh
class InvalidTransactionException extends Exception {
    public InvalidTransactionException(String message) {
        super(message);
    }
}

class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    public void deposit(double amount) throws InvalidTransactionException {
        if (amount <= 0) {
            throw new InvalidTransactionException("Số tiền nạp phải lớn hơn
0");
        }
```

```
        balance += amount;
        System.out.println("Nạp thành công: " + amount + " | Số dư hiện tại: " + balance);
    }

    public void withdraw(double amount) throws InvalidTransactionException {
        if (amount <= 0) {
            throw new InvalidTransactionException("Số tiền rút phải lớn hơn 0");
        }
        if (amount > balance) {
            throw new InvalidTransactionException("Số dư không đủ để rút " + amount);
        }
        balance -= amount;
        System.out.println("Rút thành công: " + amount + " | Số dư hiện tại: " + balance);
    }
}

public class BankDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("123456", 1000);

        try {
            System.out.println("---- Thử nạp tiền ----");
            account.deposit(500);   // hợp lệ
            account.deposit(-100);  // lỗi → ném exception
        } catch (InvalidTransactionException e) {
            System.out.println("Lỗi giao dịch: " + e.getMessage());
        } finally {
            System.out.println("Kết thúc giao dịch nạp tiền.\n");
```

```
        }


        try {
            System.out.println("---- Thử rút tiền ----");
            account.withdraw(200);   // hợp lệ
            account.withdraw(2000);  // lỗi → ném exception
        } catch (InvalidTransactionException e) {
            System.out.println("Lỗi giao dịch: " + e.getMessage());
        } finally {
            System.out.println("Kết thúc giao dịch rút tiền.");
        }
    }
}
```

Giải thích

- **throws InvalidTransactionException**: khai báo phương thức có thể ném ngoại lệ.
- **throw new InvalidTransactionException(...)**: ném ngoại lệ thực tế khi điều kiện sai.
- **try–catch**: nơi gọi phương thức sẽ xử lý ngoại lệ.
- **finally**: luôn chạy, dù có ngoại lệ hay không → ví dụ dùng để đóng kết nối DB, giải phóng tài nguyên, hoặc in ra log giao dịch.