

TỔNG HỢP MỘT SỐ CÂU HỎI ÔN TẬP CUỐI KỲ OOP 2024.1

Thực ra là đề leak được :>

Câu 1: Kết quả của đoạn code sau là gì?

```
1 public class Test {  
2     public static void main(String[] args) {  
3         byte b = 6;  
4         b += 8;  
5         System.out.println(b);  
6         b = b + 7;  
7         System.out.println(b);  
8     }  
9 }
```

1. 14 21
2. 14 13
3. **Lỗi biên dịch tại dòng 6**
4. Lỗi biên dịch tại dòng 4

Giải thích:

- Dòng `b += 8;` hợp lệ do có cơ chế ép kiểu ngầm định.
- Dòng `b = b + 7;` gây lỗi biên dịch vì không có ép kiểu tường minh từ `int` về `byte`.
- Đương nhiên có lỗi biên dịch nên sẽ không in ra gì.
- Có thể sửa lỗi bằng cách ép kiểu tường minh: `b = (byte) (b + 7);`

Câu 2: Cho 3 lớp Person, Employee và Manager như sau:

```
abstract class Person {
    protected String name;
    public abstract String print();
}
abstract class Employee extends Person {
    public String print() {
        return name;
    }
}
class Manager extends Employee {
    protected String print() {
        return "Manager" + name;
    }
}
```

Những lớp nào sẽ bị lỗi biên dịch?

1. Lớp Person
2. Lớp Employee
3. **Lớp Manager**
4. Không lớp nào bị lỗi biên dịch

Giải thích:

- Lớp Person là một lớp abstract, có một phương thức abstract print(), điều này hoàn toàn hợp lệ.

- Lớp Employee kế thừa từ Person và đã ghi đè (override) phương thức print(), nghĩa là nó đã cung cấp một triển khai hợp lệ cho phương thức abstract từ Person.

- Lớp Manager kế thừa từ Employee và cố gắng ghi đè phương thức print(). Lỗi biên dịch xảy ra tại lớp Manager, vì trong Java, khi ghi đè phương thức, phạm vi truy cập không được giảm. Tức là:

- Nếu phương thức gốc (print() trong Employee) là public, thì phương thức ghi đè trong Manager cũng phải là public.
- Ở đây, print() trong Manager được khai báo là protected, phạm vi hẹp hơn so với public, dẫn đến lỗi biên dịch.

- Sửa lỗi bằng cách sửa lại phạm vi truy cập thành public String print().

Câu 3: Cho chương trình như dưới đây được lưu toàn bộ trong file B.java (chú ý số dòng được đánh ngoài lề chương trình):

```
1  class A {
2      int a = 5;
3      protected int b = 6;
4      public int c = 7;
5  }
6
7  public class B {
8      public static void main(String[] args) {
9          A a = new A();
10         System.out.print(" " + a.a);
11         System.out.print(" " + a.b);
12         System.out.println(" " + a.c);
13     }
14 }
```

Lựa chọn nào là chính xác?

1. Lỗi khi biên dịch ở dòng 10
2. Lỗi khi biên dịch ở dòng 11
3. Chương trình không lỗi biên dịch nhưng khi chạy xuất hiện exception
4. Chương trình chạy thành công và in ra màn hình 5 6 7

Giải thích:

- Lớp A:

- Biến a có phạm vi truy cập mặc định (package-private): Chỉ có thể truy cập từ các lớp cùng package.
- Biến b có phạm vi truy cập protected: Có thể truy cập từ cùng package hoặc lớp con.
- Biến c có phạm vi truy cập public: Có thể truy cập từ bất kỳ đâu.

- Lớp B:

- B nằm trong cùng package với A (do không có khai báo package nào).
- Vì vậy, B có quyền truy cập: a (do cùng package), b (do cùng package) và c (vì là public).

- Do đó, không có lỗi biên dịch, đồng thời cũng không xảy ra ngoại lệ. Vì vậy, chương trình sẽ chạy thành công và in ra lần lượt các giá trị a, b, c của A lần lượt là 5, 6, 7.

Câu 4: Đoạn mã sau khi chạy in ra gì ?

```
public class Main {  
    public Main(int i, int j) {  
        System.out.println(method(i, j));  
    }  
    int method(int i, int j) {  
        return i++ + ++j;  
    }  
    public static void main(String[] ss) {  
        Main main = new Main(123, 456);  
    }  
}
```

1. 581

2. 579

3. 580

Giải thích:

- Giá trị của i++:

- i++ là hậu tố (post-increment), nghĩa là nó trả về giá trị ban đầu của i rồi mới tăng i lên.
- i = 123, nên i++ trả về 123, nhưng sau đó i sẽ tăng lên 124 (việc này không ảnh hưởng đến kết quả trả về).

- Giá trị của ++j:

- ++j là tiền tố (pre-increment), nghĩa là nó tăng giá trị của j trước khi sử dụng.
- j = 456, sau ++j, giá trị j = 457. ++j trả về 457.

- Vì i++ trả về 123 và ++j trả về 457, nên return i++ + ++j là $123 + 457 = 580$, in ra 580.

Câu 5: Đoạn mã sau khi chạy in ra gì ?

```
class X {
    int method(int i) {
        return i *= i;
    }
}
class Y extends X {
    double method(double d) {
        return d /= d;
    }
}
class Z extends Y {
    float method(float f) {
        return f += f;
    }
}
public class Test {
    public static void main(String[] ss) {
        Z z = new Z();
        System.out.println(z.method(210.2));
    }
}
```

1. 1.0
2. Báo lỗi biên dịch
3. 210.2
4. 420.4

Giải thích:

- Ở đây không có lỗi biên dịch, chương trình chạy thành công.
- Biến z là một đối tượng của lớp Z. Khi gọi z.method(210.2), trình biên dịch sẽ tìm kiếm phương thức phù hợp nhất. Vì 210.2 là double, Java sẽ chọn method(double d) trong lớp Y vì nó nhận tham số double.
- Giá trị trả về là: $210.2 / 210.2 = 1.0$. Vậy in ra 1.0.

Câu 6: Chọn phát biểu đúng về lớp trừu tượng (abstract class)

1. Không thể tạo thể hiện trực tiếp của lớp trừu tượng.
2. Lớp trừu tượng thường chưa hoàn thiện, được dùng làm lớp cơ sở để các lớp khác kế thừa.
3. Lớp trừu tượng phải có ít nhất 1 phương thức trừu tượng.
4. Lớp trừu tượng không cần có phương thức khởi dựng.

Giải thích:

1. Đúng. Trong Java, lớp trừu tượng không thể được khởi tạo trực tiếp. Chỉ có thể tạo đối tượng của các lớp con kế thừa lớp trừu tượng.
2. Đúng. Lớp trừu tượng thường chứa một số phương thức chưa được cài đặt (hoặc chỉ có cài đặt không đầy đủ) để các lớp con thực hiện chi tiết cụ thể.
3. Sai. Trong Java, bạn có thể khai báo một lớp là abstract dù lớp đó không chứa bất kỳ phương thức trừu tượng nào. Việc đánh dấu lớp là abstract có thể nhằm mục đích ngăn chặn việc tạo đối tượng của lớp đó.
4. Đúng. Lớp trừu tượng có thể có hoặc không có constructor. Thông thường, lớp trừu tượng có constructor để các lớp con có thể gọi (thông qua từ khóa super) trong quá trình khởi tạo, nhưng không bắt buộc phải định nghĩa một constructor nào đó.

Câu 7: Có thể dùng những chỉ định truy cập nào với phương thức khởi tạo?

1. public
2. package (default)
3. protected
4. private

Giải thích:

Không có gì phải giải thích vì đây là lý thuyết :v

Câu 8: Đoạn mã sau khi chạy in ra gì ?

```
class M {
    int i = 50;
    public M(int j) {
        System.out.print(i);
        this.i = j * 10;
    }
}
class N extends M {
    public N(int j) {
        super(j);
        System.out.print(i);
        this.i = j * 20;
    }
}
public class Test {
    public static void main(String[] ss) {
        N n = new N(25);
        System.out.print(n.i);
    }
}
```

1. 02500500
2. 0500
3. 505050
4. **50250500**

Giải thích:

- Khi tạo đối tượng `N n = new N(25);`, phương thức khởi tạo của lớp `N` được gọi.
- Trong phương thức khởi tạo của lớp `N`, `super(j)` gọi đến phương thức khởi tạo của lớp cha `M(int j)`.
 - Lúc này `M(int j)` sẽ in giá trị `i` (lúc này `i = 50` mặc định).
 - Sau đó, `this.i = j * 10` sẽ thay đổi giá trị của `i` trong lớp `M` thành $25 * 10 = 250$.
- Quay lại phương thức khởi tạo của lớp `N`, `System.out.print(i)` in ra giá trị `i` hiện tại của lớp `N`, tức là `250`.
 - Sau đó, `this.i = j * 20` trong lớp `N` sẽ thay đổi giá trị `i` của lớp `N` thành $25 * 20 = 500$.
- Cuối cùng, `System.out.print(n.i)` in ra giá trị `i` trong đối tượng `n`, tức là `500`.
- Kết quả in ra: **50250500**.

Câu 9: Đoạn mã sau khi chạy in ra gì ?

```
class A {  
    int i = 200;  
}  
class B extends A {  
    int i = 100;  
}  
public class Test {  
    public static void main(String[] ss) {  
        B a = new B();  
        System.out.println(a.i);  
    }  
}
```

1. 100
2. Báo lỗi biên dịch
3. 200

Giải thích:

- Ở đây không có lỗi biên dịch, chương trình chạy thành công.
- Ở lớp B, trường i được khai báo lại với giá trị 100, làm che khuất trường i của lớp A (được gán là 200). Do đó, khi gọi a.i với a là kiểu B, trường được lấy là của lớp B.
- Vậy, kết quả in ra là 100.

Câu 10: Kết quả in ra màn hình khi thực thi đoạn code sau là gì?

```
public class Person {  
    private int age;  
    private String name;  
  
    public Person(int age, String name) {  
        this.age = age;  
        this.name = name;  
    }  
  
    public static void main(String[] args) {  
        Person p1 = new Person(20, "Hung");  
        Person p2 = new Person(20, "Hung");  
        Person p3 = p2;  
  
        System.out.print((p1 == p2) + " ");  
        System.out.print(p1.equals(p2) + " ");  
        System.out.print((p1 == p3) + " ");  
        System.out.print((p2 == p3) + " ");  
    }  
}
```

1. false false false true
2. true true false true
3. true true true true
4. false false false false
5. false false true true
6. true true false false

Giải thích:

- (p1 == p2): So sánh địa chỉ của p1 và p2. Mặc dù cả hai đối tượng có giá trị giống nhau, nhưng chúng là hai đối tượng khác nhau trong bộ nhớ, do đó kết quả là false.
- p1.equals(p2): Mặc định, phương thức equals() so sánh địa chỉ của các đối tượng, nên kết quả cũng là false, trừ khi phương thức equals() được ghi đè để so sánh các thuộc tính của đối tượng. Trong trường hợp này, lớp Person không ghi đè equals().
- (p1 == p3): p3 trỏ đến p2, nên địa chỉ của p1 và p3 khác nhau. Kết quả là false.
- (p2 == p3): p3 và p2 đều trỏ đến cùng một đối tượng, vì vậy kết quả là true.

Câu 11: Cho chương trình Java sau được lưu toàn bộ trong file Test.java:

```
class A {}
class B extends A {}

public class Test {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        a = b;
    }
}
```

Nhận định nào sau đây đúng:

1. Câu lệnh gán $a=b$; là chuyển đổi kiểu dữ liệu tham chiếu upcasting
2. Câu lệnh gán $a=b$; là chuyển đổi kiểu dữ liệu tham chiếu downcasting
3. Câu lệnh gán $a=b$; là chuyển đổi kiểu dữ liệu tham chiếu clientservercasting
4. Câu lệnh gán $a=b$; là chuyển đổi kiểu dữ liệu tham chiếu peertopeer casting

Giải thích:

- Upcasting: Gán một đối tượng của lớp con (B) cho một biến của lớp cha (A). Không cần ép kiểu.
- Downcasting: Gán một đối tượng của lớp cha (A) cho một biến của lớp con (B). Cần ép kiểu.
- Câu lệnh $a = b$; là "Upcasting", tức là chuyển đổi kiểu dữ liệu tham chiếu từ lớp con (B) lên lớp cha (A).

Câu 12: Cho 4 hàm f là nạp chồng của nhau như sau:

<pre>public static void f(int i) { p("int"); } public static void f(long l) { p("long"); } public static void p(String s) { System.out.print(s + " "); }</pre>	<pre>public static void f(char c) { p("char"); } public static void f(float f) { p("float"); }</pre>
--	---

Kết quả khi lần lượt thực thi độc lập các lệnh sau là gì (ví dụ có lệnh in ra màn hình xâu "int" hoặc "char", có lệnh bị lỗi biên dịch)?

f((byte)5); f(6); f('c'); f(5.5); f(7L);

1. int, int, char, float, long
2. int, int, char, lỗi biên dịch, long
3. char, int, char, float, long
4. char, int, char, lỗi biên dịch, long

Giải thích:

- f((byte)5): byte có thể được tự động nâng cấp thành int. Nên hàm f(int) được gọi, in ra "int".
- f(6): 6 là số nguyên mặc định là int trong Java. Hàm f(int) được gọi, in ra "int".
- f('c'): char có thể được ánh xạ trực tiếp đến f(char) (vì có một phương thức khớp chính xác). Hàm f(char) được gọi, in ra "char".
- f(5.5): 5.5 là số thực mặc định là double trong Java. Không có phương thức f(double), và double không thể tự động chuyển xuống float mà không có ép kiểu tường minh. Gây lỗi biên dịch.
- f(7L): 7L là số nguyên có hậu tố L, nghĩa là kiểu long. Hàm f(long) được gọi, in ra "long".
- Kết quả là: int, int, char, lỗi biên dịch, long.

Câu 13: Biết rằng lớp B kế thừa lớp A, lớp C kế thừa lớp B. Chọn những phát biểu đúng về đoạn code sau:

```
1. ArrayList<B> list1 = new ArrayList<B>();  
2. ArrayList<A> list2 = (ArrayList<A>) list1;  
3. ArrayList<C> list3 = (ArrayList<C>) list1;
```

1. Dòng 2 không có lỗi biên dịch, nhưng có lỗi thực thi (runtime error)
2. Dòng 3 không có lỗi biên dịch, nhưng có lỗi thực thi (runtime error)
3. Dòng 2 có lỗi biên dịch
4. Dòng 3 có lỗi biên dịch
5. Không có lỗi biên dịch hay thực thi nào

Giải thích:

- Dòng 1: Hợp lệ, vì list1 là một ArrayList chứa các đối tượng kiểu B.
- Dòng 2: Dòng này sẽ gây lỗi biên dịch vì Java không cho phép ép kiểu trực tiếp giữa các kiểu có tham số generics khác nhau vì nó có thể dẫn đến lỗi không an toàn về kiểu dữ liệu.
- Dòng 3: Dòng này cũng sẽ gây lỗi biên dịch. Dù B là lớp cha của C, nhưng vì ArrayList có kiểu tham số hóa, Java không cho phép ép kiểu trực tiếp giữa các kiểu danh sách generics mà không bảo đảm sự tương thích về kiểu.
- Vì vậy, cả Dòng 2 và Dòng 3 đều sẽ gây lỗi biên dịch.

Câu 14: Đoạn mã sau khi chạy in ra gì ?

```
public class Demo {  
    public static void main(String[] s) {  
        try {  
            nestedTry();  
        } catch(Exception ex) {  
            System.out.println("AAA");  
        }  
    }  
    static void nestedTry() {  
        try {  
            int i = Integer.parseInt("abc");  
        } catch(NullPointerException ex) {  
            System.out.println("BBB ");  
        }  
    }  
}
```

1. BBB
2. Không in ra gì
3. AAA
4. BBBAAA

Giải thích:

- Integer.parseInt("abc") sẽ ném ra một ngoại lệ NumberFormatException, vì "abc" không thể được chuyển đổi thành số nguyên.
- Tuy nhiên, khối catch chỉ bắt NullPointerException, ngoại lệ không được bắt tại đây. Do đó, NumberFormatException sẽ lan ra ngoài lên phương thức gọi nó.
- nestedTry() được gọi trong main() có khối try-catch. Vì NumberFormatException là một ngoại lệ con của Exception, nên nó sẽ bị bắt bởi catch(Exception ex) và in ra "AAA".

Câu 15: Những khai báo mảng nào trong JAVA bị lỗi?

1. `boolean bit[] = new boolean[5];`
2. `float[] value = new float[2*3];`
3. `int[] number1 = {10, 9, 8, 7, 6};`
4. `int number2[] = new int[]{10, 9};`
5. `int[][] number3 = { {1,2}, {1,2}, {1,2} };`
6. `int number4 [][] = { {1,2}, {1,2}, {1,2} };`
7. **Tất cả khai báo đều đúng.**

Giải thích:

Tất cả các khai báo trên đều hợp lệ trong Java.

Câu 16: Chọn những phát biểu đúng về kỹ thuật ghi đè (overriding)

1. Phương thức ở lớp con hoàn toàn giống về chữ ký với phương thức kế thừa ở lớp cha
2. Phương thức lớp con KHÔNG bắt buộc có cùng kiểu trả về với phương thức kế thừa trong lớp cha
3. Chỉ định truy cập của phương thức ở lớp con KHÔNG giới hạn chặt hơn chỉ định truy cập của phương thức kế thừa trong lớp cha
4. Phương thức lớp con KHÔNG tung ra kiểu ngoại lệ mới so với phương thức kế thừa trong lớp cha

Giải thích:

1. Đúng. Trong Java, để ghi đè một phương thức, lớp con phải định nghĩa lại phương thức có cùng tên và cùng danh sách tham số (chữ ký), lưu ý rằng kiểu trả về không thuộc phần chữ ký của phương thức.
2. Đúng. Khi ghi đè, kiểu trả về của phương thức lớp con phải là kiểu trả về giống hệt hoặc là kiểu con của kiểu trả về ở lớp cha. Điều này có nghĩa là không nhất thiết phải hoàn toàn giống nhau.
3. Đúng. Khi ghi đè, phương thức lớp con phải có mức truy cập ít hạn chế hoặc bằng mức truy cập của phương thức lớp cha. Ví dụ, nếu lớp cha là `public`, thì phương thức ghi đè cũng phải là `public`; nếu lớp cha là `protected`, phương thức ở lớp con có thể là `protected` hoặc `public` nhưng không thể là `default` hay `private`.
4. Đúng. Khi ghi đè, phương thức lớp con không được khai báo ném ra các ngoại lệ mới hoặc có phạm vi rộng hơn so với các ngoại lệ được khai báo trong phương thức của lớp cha. Nó có thể ném ra ít ngoại lệ hơn hoặc ngoại lệ có phạm vi hẹp hơn.

Câu 17: Đối tượng trong chương trình Java thông thường sẽ được cấp phát trong loại bộ nhớ nào?

1. Bộ nhớ RAM
2. Bộ nhớ ROM
3. Bộ nhớ Cache
4. Các đáp án đưa ra ở đây đều không đúng

Giải thích:

Khi một đối tượng được tạo ra trong Java, nó được cấp phát trong bộ nhớ heap (heap memory), mà bộ nhớ heap nằm trong bộ nhớ RAM.

Câu 18: Chọn phát biểu đúng về giao diện (interface)

1. Là kiểu dữ liệu trừu tượng, được dùng để đặc tả các hành vi mà các lớp phải thực thi
2. Giao diện chỉ chứa các phương thức public (Từ Java 9 trở đi thì có thể có thêm phương thức static và phương thức default)
3. Giao diện giúp mô phỏng đa thừa kế
4. Sử dụng giao diện giúp mã nguồn ít bị phụ thuộc vào nhau hơn

Giải thích:

1. Đúng. Giao diện trong Java là một kiểu dữ liệu trừu tượng, không chứa mã thực thi mà chỉ cung cấp các chữ ký phương thức mà các lớp cài đặt (implement) giao diện này phải thực thi.
2. Đúng. Theo lý thuyết là vậy.
3. Đúng. Trong Java, một lớp chỉ kế thừa được một lớp cha, nhưng có thể implements nhiều giao diện. Nhờ đó, ta mô phỏng được tính năng “đa thừa kế”.
4. Đúng. Giao diện giúp giảm sự phụ thuộc giữa các lớp, vì các lớp chỉ cần thực thi giao diện mà không cần biết về chi tiết cụ thể của lớp cài đặt. Điều này giúp tăng tính mở rộng và tái sử dụng mã nguồn.

Câu 19: Chương trình Java sau được lưu toàn bộ trong file Test.java sẽ cho kết quả gì?

```
public class Test {  
    public static void main(String args[]) {  
        if(args.length > 0)  
            System.out.println(args.length);  
    }  
}
```

1. Chương trình chạy thành công, không in gì hoặc in một số nguyên > 0 ra màn hình
2. Chương trình chạy thành công và in xâu kí tự Infinity ra màn hình
3. Chương trình chạy thành công và in xâu kí tự NaN ra màn hình
4. Chương trình biên dịch thành công nhưng xuất hiện exception khi chạy
5. Lỗi biên dịch

Giải thích:

- Chương trình tuân thủ đầy đủ cú pháp Java, nên không có lỗi biên dịch.
- args là một mảng String[] chứa các tham số dòng lệnh (command-line arguments) khi chạy chương trình.
- Kiểm tra args.length > 0, nghĩa là nếu chương trình được chạy có truyền tham số, thì số lượng tham số sẽ được in ra màn hình.
- Nếu không truyền tham số, thì args.length = 0, điều kiện args.length > 0 không thỏa mãn, và chương trình không in gì.
- Ví dụ:
 - java Test => Không in gì, vì args.length == 0.
 - java Test hello world => In ra 2, vì args.length == 2.

Câu 20: Chương trình Java dưới đây được lưu toàn bộ trong file Test.java:

```
final class A {  
    int i;  
}  
class B extends A {  
    int j;  
    void display() {  
        System.out.println(j + " " + i);  
    }  
}  
public class Test {  
    public static void main(String args[]) {  
        B obj = new B();  
        B.i = 2;  
        B.j = 3;  
        obj.display();  
    }  
}
```

Nhận định nào dưới đây là đúng với chương trình này?

1. Chương trình chạy thành công và in ra màn hình 2 3
2. Chương trình chạy thành công và in ra màn hình 3 2
3. Lỗi khi chạy chương trình
4. Lỗi khi biên dịch chương trình

Giải thích:

- final class A không thể bị kế thừa. Khi cố gắng khai báo class B extends A, trình biên dịch báo lỗi.

- i và j là biến instance của một đối tượng cụ thể, nhưng trong main(), ta lại truy cập như biến static (B.i và B.j) nên sẽ gây ra lỗi biên dịch.

Câu 21: Kết quả khi thực thi lớp Test là gì?

```
class Person {
    private int age;

    public Person(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}

class Test {
    public static void increaseAge(Person p) {
        p.setAge(p.getAge() + 1);
    }

    public static void swap(Person p1, Person p2) {
        Person tmp = p1;
        p1 = p2;
        p2 = tmp;
    }

    public static void main(String[] args) {
        Person p1 = new Person(15);
        Person p2 = new Person(20);

        increaseAge(p1);
        swap(p1, p2);

        System.out.print(p1.getAge() + " ");
        System.out.print(p2.getAge() + " ");
    }
}
```

1. 15 20
2. 16 20
3. 20 15
4. 20 16

Giải thích:

- increaseAge(p1): Phương thức này gọi p1.setAge(p1.getAge() + 1), tức là tuổi của p1 được tăng lên 1 từ 15 thành 16.

- swap(p1, p2):

Lỗi tư duy phổ biến về tham chiếu trong Java:

- Java truyền tham số theo cơ chế "pass-by-value", nghĩa là bản sao của tham chiếu p1 và p2 được truyền vào hàm.
- Khi ta gán p1 = p2;, chỉ thay đổi bản sao của tham chiếu bên trong phương thức swap(), nhưng không ảnh hưởng đến p1 và p2 trong main().
- Vì vậy, hoán đổi trong swap() không làm thay đổi giá trị gốc của p1 và p2 trong main().

- Sau khi chạy phương thức increaseAge, p1 có tuổi là 16. Do đó, p1.getAge() trả về 16.

- Sau phương thức swap, tuổi của p2 vẫn là 20, vì phương thức swap không thay đổi giá trị của các đối tượng.

- Kết quả in ra sẽ là 16 20.

Câu 22: Hai phương thức sau có chồng nhau không?

```
class A {  
    public int method(int a) {  
        return a;  
    }  
}  
class B {  
    public int method(int a, int b) {  
        return a + b;  
    }  
}
```

1. Có

2. Không

Giải thích:

Hai phương thức này không phải là phương thức chồng nhau (method overloading) vì chúng không có cùng chữ ký phương thức (method signature).

- Phương thức trong lớp A có một tham số (int a).
- Phương thức trong lớp B có hai tham số (int a và int b).

Câu 23: Phát biểu nào đúng về gói (package) trong Java?

1. Mỗi gói sẽ có một thư mục tương ứng
2. Tên gói theo quy ước nên dùng chữ viết thường
3. Sử dụng dấu . trong tên gói sẽ tạo quan hệ gói cha gói con (gói nằm trong gói)
4. Kết hợp với chỉ định truy cập của lớp, gói giúp xác định phạm vi hoạt động của lớp
5. Cho lớp nào vào gói nào là tùy ý, ta không cần phải chú ý tổ chức sao cho một gói chỉ chứa các lớp có quan hệ với nhau
6. Gói giúp tạo các lớp có tên giống nhau
7. Gói giúp tổ chức và xác định vị trí lớp dễ dàng

Giải thích:

- 1, 2 Đúng.

- 3 Đúng. Dấu chấm được dùng để phân cấp tên package (ví dụ: com.example) thể hiện mối quan hệ cấp bậc giữa các package.

- 4 Đúng. Các mức truy cập (ví dụ: package-private, protected) phụ thuộc vào việc lớp đó thuộc package nào, do đó package giúp xác định phạm vi truy cập.

- 5 Sai. Ta nên tổ chức các lớp trong một package sao cho chúng có liên quan với nhau.

- 6 Đúng. Nhờ có package, ta có thể có các lớp cùng tên nhưng thuộc các package khác nhau mà không gây xung đột.

- 7 Đúng. Package là công cụ để tổ chức mã nguồn, giúp việc quản lý, tìm kiếm và bảo trì mã nguồn trở nên hiệu quả.

Câu 24: Từ khóa được dùng để định nghĩa một giao diện là:

1. intf
2. Intf
3. interface
4. Interface

Giải thích:

Lý thuyết nó vậy đó :>

Câu 25: Đoạn mã sau khi chạy in ra gì?

```
public class Test {  
    static int method() {  
        int i = 0;  
        try {  
            i = 1;  
            return i;  
        } catch (Exception e) {  
            i = 2;  
            return i;  
        } finally {  
            i = 3;  
        }  
    }  
  
    public static void main(String[] ss) {  
        System.out.println(method());  
    }  
}
```

1. 1
2. 2
3. 3

Giải thích:

- Trong khối try, i được gán giá trị 1, sau đó phương thức trả về giá trị của i = 1.
- Khối catch bị bỏ qua do không có ngoại lệ xảy ra.
- Khối finally luôn được thực thi, nhưng nó chỉ thay đổi giá trị i = 3, chứ không ghi đè giá trị trả về.
- Giá trị đã được return trước đó (trong try) vẫn giữ nguyên.
- Kết quả in ra màn hình là: 1.

Câu 26: Cho 2 interface như sau

```
interface Printable {  
    public void print();  
}  
interface Stringable {  
    String stringify();  
}
```

Lớp Person sau bị lỗi ở những dòng nào

```
1  class Person implements Printable, Stringable {  
2      protected String name;  
3      public String stringify() {  
4          return name;  
5      }  
6      public void print() {  
7          System.out.println(name);  
8      }  
9      public static void main(String[] args) {  
10         Printable p1 = new Printable();  
11         Printable p2 = new Person();  
12         String name = p2.stringify();  
13     }  
14 }
```

1. Dòng 1
2. Dòng 6
3. Dòng 10
4. Dòng 11
5. Dòng 12

Giải thích:

- Dòng 1: Person triển khai cả hai interface Printable và Stringable. Vì Person đã cung cấp đầy đủ phương thức print() và stringify(), không có lỗi.
- Dòng 6: print() được định nghĩa chính xác theo interface Printable, không có lỗi.
- Dòng 10: Lỗi biên dịch do Interface không thể được khởi tạo trực tiếp.
- Dòng 11: Không có lỗi vì lớp Person thực hiện interface Printable, do đó có thể tạo đối tượng Person và gán cho biến Printable.
- Dòng 12: Lỗi biên dịch do p2 có kiểu Printable, mà Printable không có phương thức stringify().

Câu 27: Đoạn mã sau khi chạy in ra gì?

```
public class Test {  
  
    static void m1(String s1) {  
        s1.toLowerCase();  
        System.out.print(s1);  
    }  
  
    static void m2(String s1) {  
        s1 = s1.toUpperCase();  
        System.out.print(s1);  
    }  
  
    static void m3(StringBuffer s1) {  
        s1.delete(0,1);  
        System.out.print(s1);  
    }  
  
    static void m4(StringBuffer s1) {  
        s1 = s1.delete(0,1);  
        System.out.print(s1);  
    }  
  
    public static void main(String[] s) {  
        String s1 = "Ab";  
        StringBuffer s2 = new StringBuffer("CdE");  
        m1(s1);  
        m2(s1);  
        System.out.print(s1 + " ");  
        m3(s2);  
        m4(s2);  
        System.out.print(s2);  
    }  
}
```

1. abABAB dEEE
2. AbABAb dEEE
3. abABAB dEdEdE
4. AbABAB dEEE

Giải thích:

- Phương thức m1(String s1): Tham số s1 ban đầu là "Ab". Lệnh s1.toLowerCase(); tạo ra chuỗi "ab" nhưng không gán lại cho s1. System.out.print(s1); in ra "Ab".
- Phương thức m2(String s1): Tham số s1 ban đầu là "Ab". Lệnh s1 = s1.toUpperCase(); tạo ra chuỗi "AB" gán cho biến cục bộ s1. System.out.print(s1); in ra "AB". Vì đối tượng s1 được truyền vào không bị thay đổi (immutable), sau m2, s1 vẫn là "Ab".
- Lệnh System.out.print(s1 + " "); In ra nội dung hiện tại của s1, tức "Ab".
- Phương thức m3(StringBuffer s1): Tham số s2 ban đầu là một StringBuffer chứa "CdE". Lệnh s1.delete(0,1); xóa ký tự từ vị trí 0 đến 1 (xóa ký tự đầu tiên, là 'C'). Sau khi xóa, s2 trở thành "dE". System.out.print(s1); in ra "dE".
- Phương thức m4(StringBuffer s1): Tham số s2 hiện tại là "dE". Lệnh s1 = s1.delete(0,1); xóa ký tự từ vị trí 0 đến 1, tức xóa ký tự 'd', nên chuỗi trở thành "E". System.out.print(s1); in ra "E". Vì đối tượng s2 được truyền vào đã bị thay đổi (mutable), sau m4, s2 chỉ còn lại "E".
- Lệnh System.out.print(s2); In ra nội dung hiện tại của s2, tức "E".

Câu 27: Cho 5 lớp như sau:

```
abstract class Writer {
    public abstract void write(String s) throws RuntimeException;
}

class HTMLWriter extends Writer {
    public void write(String s) throws
RuntimeException { // ...
        // ...
    }
}

class JSONWriter {
    public void write(String s) throws
Exception { // ...
        // ...
    }
}

class CSVWriter extends Writer {
    public void write(String s) throws
Exception { // ...
        // ...
    }
}

class TEXTWriter extends Writer {
    public void write(String s) throws
ArithmeticException { // ...
        // ...
    }
}
```

Những lớp nào bị lỗi biên dịch (Gợi ý: ArithmeticException là lớp dẫn xuất từ RuntimeException)

1. HTMLWriter
2. CSVWriter
3. JSONWriter
4. TEXTWriter

Giải thích:

- Phương thức write của lớp Writer khai báo ném ra RuntimeException (một unchecked exception). Khi ghi đè phương thức này trong các lớp con, phương thức ghi đè phải có chữ ký giống hệt. Ngoại lệ mà nó ném ra phải là cùng loại hoặc là con của ngoại lệ được khai báo trong phương thức của lớp cha.

- HTMLWriter: Phương thức này ghi đè chính xác theo chữ ký của lớp cha và ném ra RuntimeException (đúng yêu cầu). Không lỗi biên dịch.

- CSVWriter: Phương thức ghi đè khai báo ném ra Exception, là lớp cha của RuntimeException và là một ngoại lệ kiểm tra, mở rộng phạm vi ngoại lệ so với lớp cha. Lỗi biên dịch.

- JSONWriter: Không kế thừa từ Writer, nên phương thức write của nó không phải ghi đè phương thức của Writer. Việc khai báo throws Exception ở đây là hoàn toàn hợp lệ. Không lỗi biên dịch.

- TEXTWriter: ArithmeticException là con của RuntimeException. Hợp lệ vì nó là một ngoại lệ không kiểm tra và thuộc phạm vi ngoại lệ của phương thức lớp cha. Không lỗi biên dịch.

Câu 28: Cho chương trình như dưới đây được lưu toàn bộ trong file HelloWorld.java:

```
public class Hello {
    static int add(int i, int j) {
        return i + j;
    }
}
public class HelloWorld extends Hello {
    public static void main(String argv[]) {
        short sNum = 10;
        System.out.println(add(sNum, 6));
    }
}
```

Nhận định nào sau đây là đúng với chương trình này?

1. Chương trình chạy và in 16 ra màn hình
2. **Lỗi khi biên dịch**
3. Lỗi khi chạy chương trình
4. Chương trình chạy và in 6 ra màn hình

Giải thích:

- Lỗi biên dịch do có hai lớp public trong cùng một file.

- Quy tắc trong Java:

- Trong một file .java, chỉ được có một lớp public, và tên lớp public phải trùng với tên file.
- Ở đây, tập tin có tên HelloWorld.java nhưng lại chứa hai lớp public là Hello và HelloWorld, gây lỗi biên dịch.

Câu 29: Chương trình Java sau được lưu toàn bộ trong file Output.java sẽ cho kết quả là gì?

```
public class Output {  
    public static void main(String args[]) {  
        double a, b, c;  
        a = 3.0 / 0;  
        b = 0 / 4.0;  
        c = 0 / 0.0;  
        System.out.println("a=" + a + " b=" + b + " c=" + c);  
    }  
}
```

1. Lỗi khi chạy chương trình
2. Lỗi khi biên dịch chương trình
3. Chương trình chạy thành công và in ra màn hình a=NaN b=0.0 c=Infinity
4. Chương trình chạy thành công và in ra màn hình a=0.0 b=Infinity c=NaN
5. Chương trình chạy thành công và in ra màn hình a=Infinity b=0.0 c=NaN

Giải thích:

- Trong Java, phép chia một số thực khác 0 cho 0 sẽ không gây lỗi mà trả về Infinity => $a = 3.0 / 0 = \text{Infinity}$.
- Trong toán học, $0 / \text{số khác } 0$ luôn bằng 0 => $b = 0 / 4.0 = 0.0$.
- Trong Java, phép chia $0.0 / 0.0$ là một phép toán vô nghĩa (undefined), kết quả là NaN (Not a Number) => $c = 0 / 0.0 = \text{NaN}$.
- Lưu ý, nếu đổi a và c sang dạng số nguyên (int) thì chương trình vẫn biên dịch thành công, tuy nhiên sẽ có lỗi thực thi: ArithmeticException: / by zero.

Câu 30: Cho trước lớp Employee. Kết quả in ra màn hình khi thực thi đoạn chương trình sau là gì?

```
System.out.print("Begin ");
Employee e;
try {
    System.out.println(e.toString());
} catch (Exception ex) {
    System.out.print("Ex ");
} catch (RuntimeException ex) {
    System.out.print("RunEx ");
}
System.out.println("End ");
```

1. Begin Ex End
2. Begin RunEx End
3. Begin End
4. Không biên dịch được nên không thực thi được

Giải thích:

- Đoạn code trên có nhiều vấn đề.

- Exception là lớp cha của RuntimeException. Java yêu cầu rằng các khối catch phải được sắp xếp theo thứ tự từ các lớp ngoại lệ con lên lớp ngoại lệ cha. Vì vậy, khối catch cho RuntimeException phải được đặt trước Exception (chứ không phải ngược lại).

- Nếu sửa lỗi trên thì code vẫn lỗi. Biến e chỉ được khai báo nhưng chưa được khởi tạo, dẫn đến lỗi biến cục bộ chưa được gán giá trị. Trong Java, biến cục bộ (local variable) phải được khởi tạo trước khi sử dụng. Vì e chưa được khởi tạo, khi truy cập e.toString(), chương trình không thể biên dịch.

- Vì vậy, chương trình này không biên dịch được và gây lỗi biên dịch.

Câu 31: Cho 3 lớp: lớp Person, lớp Employee, và lớp Manager như sau:

```
class Person {
    public void getDetail() {
        System.out.print("P ");
    }
}

class Employee extends Person {
    public void getDetail() {
        System.out.print("E ");
    }
}

class Manager extends Employee {
    public void getDetail() {
        System.out.print("M ");
    }

    public static void main(String[]
args) {
        Person o1 = new Employee();
        Employee o2 = new Manager();
        Person o3 = new Person();

        o1.getDetail();
        o2.getDetail();
        o3.getDetail();
    }
}
```

Kết quả thực thi lớp Manager in ra màn hình gì?

1. Bị lỗi biên dịch liên quan tới ép kiểu downcasting
2. Bị lỗi thực thi liên quan tới ép kiểu downcasting
3. P E M
4. E M P

Giải thích:

- Không có ép kiểu xuống (downcasting) nào trong chương trình nên không bị lỗi biên dịch, cũng không bị lỗi thực thi.

- o1 có kiểu Person, nhưng đối tượng thực sự là Employee. Khi gọi o1.getDetail(), do tính chất đa hình, phương thức của đối tượng thực tế (Employee) sẽ được gọi. In ra: "E".

- o2 có kiểu Employee, nhưng thực tế là Manager. Khi gọi o2.getDetail(), vì phương thức getDetail() đã bị ghi đè trong Manager, nên phương thức của Manager sẽ được gọi. In ra: "M".

- o3 có kiểu Person và đối tượng thực tế cũng là Person. Khi gọi o3.getDetail(), phương thức trong Person được gọi. In ra: "P".

- Kết quả in ra: E M P.

Câu 32: Những phát biểu nào sau đây là đúng về nguyên lý đóng gói (Encapsulation)

1. Đóng gói là đưa dữ liệu và các phương thức thao tác tới dữ liệu đó vào một đơn vị duy nhất
2. Đóng gói giúp hạn chế truy cập tới một số thành phần của đối tượng
3. Các phương thức getter và setter không phải là một phần của kỹ thuật đóng gói
4. Các phương thức getter và setter giúp thực hiện che giấu thông tin (information hiding)
5. Đóng gói giúp mã nguồn an toàn hơn với người dùng cuối (end user)
6. Đóng gói giúp mã nguồn an toàn hơn với lập trình viên khác (developer)

Giải thích:

1. Đây là định nghĩa cơ bản của đóng gói trong lập trình hướng đối tượng. Đúng.
2. Mục đích của đóng gói là che giấu dữ liệu nội bộ và chỉ cho phép truy cập qua các phương thức được kiểm soát. Đúng.
3. Thực tế, getter và setter được sử dụng để truy cập và sửa đổi dữ liệu riêng tư của đối tượng một cách có kiểm soát, đóng góp vào việc che giấu thông tin. Sai.
4. Cái trên sai thì cái dưới này phải đúng rồi :v
5. Người dùng cuối (end user) thường chỉ sử dụng giao diện của ứng dụng mà không cần biết đến chi tiết cài đặt bên trong. Sai.
6. Khi các lớp được đóng gói, các chi tiết cài đặt nội bộ bị ẩn đi, giúp giảm khả năng sai sót do các lập trình viên khác truy cập hoặc thay đổi trực tiếp các thành phần quan trọng, từ đó bảo vệ tính toàn vẹn của mã nguồn. Đúng.

Câu 33: Cho chương trình như dưới đây được lưu toàn bộ trong file Student.java:

```
public class Student {  
    int marks;  
    public Student() {}  
    public Student(int x) {  
        marks = x;  
    }  
    public static void main(String[] args) {  
        Student s1 = new Student(100);  
        Student s2 = new Student();  
        Student s3;  
    }  
}
```

Lựa chọn nào nêu ra là đúng?

1. Chương trình bị lỗi khi biên dịch
2. Chương trình chạy và xuất hiện ngoại lệ (exception)
3. Chương trình chạy thành công và 2 thể hiện (instance) của lớp Student được khởi tạo ra (instantiated)
4. Chương trình chạy thành công và 3 thể hiện (instance) của lớp Student được khởi tạo ra (instantiated)

Giải thích:

- Dễ thấy chương trình không có lỗi biên dịch. Biến s3 chỉ được khai báo nhưng không được sử dụng, nên không gây lỗi NullPointerException hay bất kỳ ngoại lệ nào.
- Chương trình chỉ khởi tạo ra 2 đối tượng của lớp Student (tương ứng với s1 và s2); biến s3 chỉ được khai báo nhưng không được khởi tạo (không có lệnh new).

Câu 34: Đoạn mã sau bị lỗi biên dịch ở những dòng nào?

```
1  class A {  
2      public int check(double a) {  
3          return (int) a;  
4      }  
5      public double check (double a) {  
6          return a;  
7      }  
8  }  
9  class B {  
10     private int check (double a, int b) {  
11         return (int) a + b;  
12     }  
13     private double check (double a, int b) {  
14         return a + b;  
15     }  
16 }
```

- | | | | |
|------------|------------|------------|------------|
| 1. Dòng 2 | 2. Dòng 3 | 3. Dòng 5 | 4. Dòng 6 |
| 5. Dòng 10 | 6. Dòng 11 | 7. Dòng 13 | 8. Dòng 14 |

Giải thích:

- Lỗi biên dịch tại dòng 5: Phương thức này có cùng tên (check) và cùng danh sách tham số (double a) như phương thức trên. Java không hỗ trợ method overloading chỉ khác nhau ở kiểu trả về.

- Lỗi biên dịch tại dòng 13: Lý do tương tự lớp A: Hai phương thức có cùng tên check và cùng danh sách tham số (double a, int b), chỉ khác kiểu trả về. Không hợp lệ trong Java.

Câu 35: Phát biểu nào đúng về lớp trong Java

1. Khai báo lớp có thể đặt trước khai báo package
2. Các chỉ định truy cập có thể dùng cho lớp là public, private, default, protected
3. Các chỉ định truy cập có thể dùng cho lớp là public, private, default
4. Các chỉ định truy cập có thể dùng cho lớp là public, private
5. Các chỉ định truy cập có thể dùng cho lớp là public, default
6. Chỉ định truy cập cho lớp xác định phạm vi sử dụng của lớp
7. Bên trong thân lớp phải có ít nhất một thuộc tính hoặc phương thức gì đó

Giải thích:

- 1 Sai. Trong Java, nếu sử dụng package, khai báo package phải xuất hiện ở đầu file, trước mọi khai báo lớp.
- Trong Java, chỉ có public và default có thể được dùng cho lớp. Không thể sử dụng private hoặc protected cho lớp (vì chúng chỉ có thể được sử dụng cho các thành viên của lớp, không phải cho lớp) => 2, 3, 4 sai, 5 đúng.
- 6 Đúng. Việc khai báo một lớp là public hay default quyết định phạm vi truy cập của lớp đó (public: có thể truy cập từ mọi nơi, default: chỉ truy cập trong cùng package).
- 7 Sai. Trong Java, lớp có thể được khai báo rỗng (không chứa thuộc tính hay phương thức nào).

Câu 36: Cho các giao diện và lớp như sau:

```
1  interface Printable {
2      void print();
3  }
4  interface Stringable extends Printable {
5      String stringify();
6  }
7  class Person implements Stringable {
8      protected String name;
9      public String stringify() {
10         return name;
11     }
12 }
```

Có những lỗi ở dòng nào

1. Dòng 4, do một giao diện không được kế thừa một giao diện khác
2. Dòng 5, do phương thức trong interface bắt buộc phải có chỉ định truy cập là public
3. Dòng 7
4. Dòng 9

Giải thích:

- Dòng 4: Không có lỗi. Giao diện (interface) hoàn toàn có thể kế thừa một hoặc nhiều giao diện khác bằng từ khóa `extends`.
- Dòng 5: Không có lỗi. Các phương thức trong giao diện mặc định là `public` và `abstract`, ngay cả khi không ghi rõ từ khóa `public`. Do đó, đoạn khai báo này hợp lệ.
- Dòng 7: Có lỗi. Lớp `Person` chưa triển khai phương thức `print()` từ giao diện `Printable`, nhưng đã khai báo `implements Stringable`. Vì `Stringable` kế thừa `Printable`, nên bất kỳ lớp nào triển khai `Stringable` cũng phải triển khai tất cả các phương thức của cả hai giao diện, tức là phải có cả `stringify()` và `print()`.
- Dòng 9: Không có lỗi. Lớp `Person` đang triển khai phương thức `stringify()`, vốn được khai báo trong `Stringable`. Phương thức này cũng có `public` (bắt buộc vì nó triển khai từ giao diện), nên đúng cú pháp.

Câu 37: Đoạn code sau bị lỗi biên dịch ở những dòng nào

```
1.  public class Person {  
2.      private int age;  
3.      private String name;  
4.      public void Person(int age) {  
5.          this.age = age;  
6.      }  
7.      public static void main(String[] args) {  
8.          Person p = new Person(15);  
9.          p.name = "Peter";  
10.     }  
11. }
```

1. Dòng 4
2. Dòng 5
3. **Dòng 8**
4. Dòng 9
5. Không có lỗi biên dịch nào

Giải thích:

- Trong Java, hàm khởi tạo (constructor) không có kiểu trả về, nhưng ở đây lại có void, làm cho nó trở thành một phương thức thông thường chứ không phải constructor. Điều này có nghĩa là lớp Person không có constructor hợp lệ để nhận tham số age, dẫn đến lỗi khi tạo đối tượng ở dòng 8.

Tự Luận kỳ 2024.1

Câu 1 (CTC): Nêu và phân tích 4 nguyên lý chính của lập trình hướng đối tượng.

- Tính Đóng gói (Encapsulation): Đóng gói là việc che giấu dữ liệu và chỉ cho phép truy cập thông qua các phương thức công khai. Điều này giúp bảo vệ dữ liệu và ngăn chặn sự can thiệp ngoài ý muốn. Ví dụ: Sử dụng private cho thuộc tính và public getter/setter để truy cập.
- Tính Kế thừa (Inheritance): Kế thừa cho phép một lớp con sử dụng lại hoặc mở rộng các đặc điểm của lớp cha, giúp tái sử dụng mã nguồn và giảm dư thừa. Ví dụ: `class Dog extends Animal { }` → Dog kế thừa từ Animal, có thể sử dụng hoặc ghi đè (override) phương thức của lớp cha.
- Tính Đa hình (Polymorphism): Đa hình cho phép một phương thức có nhiều hình thức khác nhau, giúp mã linh hoạt hơn. Có hai loại: đa hình tĩnh (method overloading) và đa hình động (method overriding). Ví dụ: Một phương thức `speak()` có thể hoạt động khác nhau trong Cat và Dog dù cùng tên.
- Tính Trừu tượng (Abstraction): Trừu tượng giúp ẩn đi các chi tiết phức tạp, chỉ cung cấp những gì cần thiết thông qua abstract class hoặc interface. Ví dụ: abstract class `Vehicle { abstract void move(); }` → Các lớp con như Car hoặc Bike phải triển khai phương thức `move()`.

Câu 2 (CTC): Phân biệt sự giống và khác nhau giữa Interface và Abstract Class.

- Giống nhau

- Trừu tượng: Cả interface và abstract class đều dùng để định nghĩa các hành vi mà lớp con phải triển khai.
- Không thể khởi tạo: Không thể tạo đối tượng trực tiếp từ cả interface và abstract class.
- Tính kế thừa: Lớp con phải kế thừa (abstract class) hoặc triển khai (interface) các phương thức được định nghĩa.






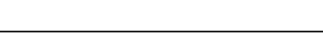
- Khác nhau

- Interface: Chỉ chứa hằng số (public static final) và phương thức mặc định là abstract (có thể có default method từ Java 8+). Không có constructor, không thể chứa biến instance, tất cả phương thức mặc định là public. Cho phép đa kế thừa (một lớp có thể triển khai nhiều interface).
- Abstract Class: Có thể chứa biến instance, phương thức có thể có thân hoặc abstract. Có constructor, có thể chứa phương thức private, protected, hoặc public. Chỉ kế thừa một lớp trừu tượng, thích hợp khi có thuộc tính chung và muốn triển khai sẵn một số phương thức.

Câu 3 (CTTT): Vẽ biểu đồ lớp mô tả hệ thống cho nhà hàng với các lớp như sau:

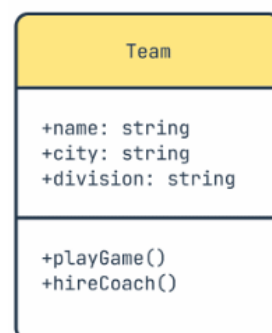
- Lớp MainDish có các thuộc tính name, price, portion (regular, large), và calories, cùng phương thức displayInfo() để hiển thị thông tin món ăn.
- Lớp Drink có các thuộc tính name, price, small, medium, large, và isCold, cùng phương thức displayInfo() để hiển thị thông tin đồ uống.
- Lớp Order lưu danh sách các món ăn và đồ uống, có các phương thức addItem() để thêm món vào đơn, displayOrder() để hiển thị đơn hàng và calculateTotal() để tính tổng giá trị đơn hàng.
- Mỗi món ăn hoặc đồ uống có thể được thêm vào đơn và tổng tiền sẽ được tính từ các món đã chọn.
- Quan hệ Kế thừa (Inheritance): Nếu MainDish và Drink đều có thuộc tính chung (name, price) và phương thức chung (displayInfo()), thì có thể tạo một lớp cha MenuItem. MainDish và Drink sẽ kế thừa (extends) từ MenuItem.
- Quan hệ Kết hợp (Association): Order có danh sách các món ăn và đồ uống, nghĩa là nó có một mối quan hệ "has-a" với MenuItem (tập hợp các MainDish và Drink). Cụ thể, Order sẽ có một danh sách (List<MenuItem> items) để lưu tất cả các món.
- Tổng quan về các lớp:
 - Lớp MenuItem (Lớp cha): Chứa name, price, displayInfo().
 - Lớp MainDish (Kế thừa MenuItem): Bổ sung portion và calories.
 - Lớp Drink (Kế thừa MenuItem): Bổ sung size và isCold.
 - Lớp Order: Chứa danh sách MenuItem và các phương thức addItem(), displayOrder(), calculateTotal().

- Rồi tự vẽ nha. Lưu ý những cái mũi tên này:

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

- Kí hiệu phạm vi truy cập:

- + Public
- Private
- # Protected
- ~ Default



<- Tên lớp đây

<- Thuộc tính

<- Phương thức

Hết rồi. Ôn cả file cũ nữa nhé :>
From Ada with luv :3