Midterm

- Due No due date
- Points 30
- Questions 30
- Available Dec 10 at 9:05am - Dec 10 at 9:25am 20 minutes
- Time Limit 20 Minutes

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	19 minutes	29 out of 30

Score for this quiz: 29 out of 30
Submitted Dec 10 at 9:24am
This attempt took 19 minutes.

Question 1
1 / 1 pts

Tính đa hình trong Java cho phép một lớp có nhiều phương thức cùng tên nhưng có các cách thực thi khác nhau, điều này được gọi là _____.

Correct!

- [x] overloading

Correct!

- [x] nạp chồng

Question 2
1 / 1 pts

Khả năng của một lớp con cung cấp cách thực thi riêng cho một phương thức đã được định nghĩa trong lớp cha được gọi là _____.

Correct!

- [x] overriding

Correct!

- [x] ghi đè

Question 3
0 / 1 pts

Khái niệm tính đa hình trong Java chủ yếu được thực hiện thông qua _____.

- [ ] nạp chồng

Correct Answer

- [ ] kế thừa
- [ ] liên kết

Correct Answer

- [ ] giao diện

You Answered

- [x] ghi đè

Question 4

1 / 1 pts

Trong Java, một phương thức được khai báo trong lớp cha và được ghi đè trong lớp con sẽ được xử lý trong quá trình _____, thể hiện tính đa hình ở thời gian chạy.

Correct!

- [x] runtime

Correct!

- [x] thời gian chạy

Question 5

1 / 1 pts

Từ khóa _____ được sử dụng để gọi một phương thức trong lớp cha đã bị ghi đè bởi lớp con.

Correct!

super

Correct Answers

super

Question 6

1 / 1 pts

Liên kết tĩnh trong Java xảy ra trong thời gian _____ và được áp dụng cho các phương thức private, final hoặc static.

- [ ] dịch

Correct!

- [x] biên dịch
- [ ] dịch chương trình

Correct!

- [x] compile

Question 7
1 / 1 pts

Liên kết động còn được gọi là liên kết _____ vì nó diễn ra trong thời gian chạy và được xác định bởi kiểu của đối tượng thay vì kiểu của biến tham chiếu.

Correct!

- (•) muộn
- ( ) sớm
- ( ) linh hoạt
- ( ) trừu tượng

Question 8
1 / 1 pts

Quá trình xác định phương thức nào sẽ được gọi trong liên kết động phụ thuộc vào _____ của đối tượng

Correct!

- (•) lớp
- ( ) phương thức
- ( ) kiểu của biến tham chiếu
- ( ) quyền truy cập

Question 9
1 / 1 pts

Trong hệ thống I/O của Java, **Mẫu thiết kế Decorator** cho phép thêm _____ vào luồng dữ liệu một cách linh hoạt. Ví dụ, để nén dữ liệu khi ghi vào một tệp, bạn có thể bọc một `FileOutputStream` bằng GZIPOutputStream.

Correct!

- [x] functionalities

Correct!

- [x] chức năng

Question 10
1 / 1 pts

Chọn 2 options để điền vào chỗ trống:

_____ trong Java cho phép kết hợp các dòng stream linh hoạt. Ví dụ, để chuyển đổi dữ liệu byte từ một `FileInputStream` thành các ký tự, bạn có thể bọc nó với `InputStreamReader`, sau đó tiếp tục bọc với _____ để đọc dữ liệu theo từng dòng

Correct!

- [x] Decorator design pattern
- [ ] Design pattern

Correct!

- [x] BufferedReader
- [ ] CharacterStream
- [ ] OutputDataStream

Question 11

1 / 1 pts

Câu nào dưới đây mô tả chính xác về Factory Method Pattern trong Java?

- Factory Method Pattern được sử dụng để tạo một đối tượng duy nhất của một lớp và đảm bảo rằng đối tượng chỉ được tạo ra một lần.
- Factory Method Pattern được sử dụng để chuyển đổi một loại đối tượng này thành một loại đối tượng khác trong thời gian chạy.

Correct!

- (•) Factory Method Pattern cung cấp một giao diện để tạo đối tượng, nhưng cho phép các lớp con thay đổi kiểu đối tượng sẽ được tạo ra.

Question 12

1 / 1 pts

Đâu là câu phát biểu đúng về mẫu thiết kế **Abstract Factory** trong Java?

- Mẫu thiết kế Abstract Factory được sử dụng để tạo một thể hiện duy nhất của một lớp trong môi trường đa luồng.

Correct!

- (•) Mẫu thiết kế Abstract Factory cung cấp một giao diện để tạo ra một gia đình các đối tượng liên quan hoặc phụ thuộc mà không xác định các lớp cụ thể của chúng.

Mẫu thiết kế Abstract Factory cho phép tạo các đối tượng theo thứ tự tuần tự từ một tập hợp các nhà máy đã được định nghĩa sẵn.

Mẫu thiết kế Abstract Factory chủ yếu được sử dụng để quản lý kết nối cơ sở dữ liệu và giao dịch trong một ứng dụng Java.

Question 13
1 / 1 pts

Nguyên tắc nào trong SOLID nhấn mạnh rằng một lớp chỉ nên có một lý do để thay đổi?

- Nguyên tắc Đảo ngược Sự phụ thuộc (Dependency Inversion Principle)
- Nguyên tắc Tách giao diện (Interface Segregation Principle)
- Nguyên tắc Mở/Đóng (Open/Closed Principle)

Correct!

- Nguyên tắc Trách nhiệm Đơn lẻ (Single Responsibility Principle)

Question 14
1 / 1 pts

Nguyên tắc Mở/Đóng phát biểu rằng:

- Các lớp nên mở để chỉnh sửa và đóng để mở rộng.
- Các lớp không nên phụ thuộc vào các trừu tượng.
- Các lớp chỉ nên có một trách nhiệm duy nhất.

Correct!

- Các lớp nên mở để mở rộng và đóng để chỉnh sửa.

Question 15
1 / 1 pts

Nguyên tắc **Thay thế Liskov (Liskov Substitution Principle)** đảm bảo điều gì trong Java?

- Các lớp con không được ghi đè các phương thức của lớp cha.
- Các lớp con phải thay thế lớp cha và định nghĩa lại tất cả các phương thức.
- Các kiểu con không được kế thừa bất kỳ chức năng nào từ kiểu cha.

Correct!

- Các kiểu con phải có thể thay thế kiểu cha mà không làm thay đổi hành vi của chương trình.

Question 16
1 / 1 pts

Nguyên tắc Phân tách Giao diện (Interface Segregation Principle) nói rằng:

Correct!

- (selected) Một lớp chỉ nên triển khai các giao diện liên quan đến chức năng của nó.
- Tất cả các lớp nên triển khai một giao diện chung duy nhất.
- Các giao diện nên càng lớn và bao quát càng tốt.
- Một lớp không nên phụ thuộc vào các giao diện nào cả.

Question 17
1 / 1 pts

Mục đích chính của Nguyên tắc Đảo ngược Phụ thuộc (Dependency Inversion Principle) là gì?

- Giảm thiểu việc sử dụng giao diện trong ứng dụng.

Correct!

- (selected) Làm cho các module cấp cao độc lập với các module cấp thấp bằng cách phụ thuộc vào các trừu tượng.
- ảm bảo rằng các module cấp cao phụ thuộc vào các module cấp thấp.
- Loại bỏ sự phụ thuộc giữa các lớp không liên quan.

Question 18
1 / 1 pts

Nguyên tắc nào trong SOLID đảm bảo rằng một thực thể phần mềm không nên phụ thuộc vào bất kỳ thứ gì mà nó không sử dụng?

Correct!

- (selected) Nguyên tắc Phân tách Giao diện (Interface Segregation Principle)
- Nguyên tắc Trách nhiệm Duy nhất (Single Responsibility Principle)
- Nguyên tắc Đảo ngược Phụ thuộc (Dependency Inversion Principle)
- Nguyên tắc Mở/Đóng (Open/Closed Principle)

Question 19
1 / 1 pts

Điều nào sau đây đúng về **lớp trừu tượng** so với **interface** trong Java?

- Lớp trừu tượng hỗ trợ kế thừa đa (multiple inheritance), trong khi interface thì không.

Correct!

- (selected)

Lớp trừu tượng có thể chứa cả phương thức trừu tượng và phương thức cụ thể, trong khi interface chỉ chứa phương thức trừu tượng (trước Java 8).

- Lớp trừu tượng không thể có biến instance, trong khi interface có thể.

Lớp trừu tượng và interface không có sự khác biệt về cách triển khai phương thức.

Question 20

1 / 1 pts

Một sự khác biệt quan trọng giữa lớp trừu tượng và interface trong Java là gì?

Interface hỗ trợ các trường (field) với bất kỳ phạm vi truy cập nào, trong khi lớp trừu tượng thì không.

Interface có thể có phương thức tĩnh (static) và mặc định (default) (từ Java 8), trong khi lớp trừu tượng thì không.

Correct!

Lớp trừu tượng có thể có constructor, trong khi interface thì không.

Lớp trừu tượng cho phép kế thừa đa, trong khi interface thì không.

Question 21

1 / 1 pts

Sự khác biệt giữa **checked exception** và **unchecked exception** trong Java là gì?

Correct!

Checked exception phải được bắt hoặc khai báo trong chữ ký phương thức, trong khi unchecked exception không yêu cầu xử lý.

Unchecked exception phải được khai báo trong chữ ký phương thức, trong khi checked exception thì không.

Checked exception là các lớp con của RuntimeException, trong khi unchecked exception là các lớp con của Exception.

Checked exception không thể được bắt bằng khối try-catch.

Question 22

1 / 1 pts

Ngoại lệ nào là ví dụ của **unchecked exception**?

Correct!

ArithmeticException

IOException

SQLException

FileNotFoundException

Question 23

1 / 1 pts

Điều gì xảy ra nếu một ngoại lệ được ném ra trong khối `try` nhưng không có khối `catch` phù hợp?

- Chương trình dừng thực thi ngay lập tức.
- Chương trình tự động tạo một khối catch mặc định cho ngoại lệ đó.
- Chương trình bỏ qua ngoại lệ và tiếp tục thực thi.

Correct!

- Ngoại lệ được lan truyền lên phương thức cao hơn trong ngăn xếp lệnh gọi.

Question 24

1 / 1 pts

Câu nào sau đây là **sai** về khối `finally` trong Java?

- Khối finally được sử dụng để giải phóng tài nguyên, chẳng hạn như đóng tệp hoặc kết nối mạng.
- Khối finally luôn luôn được thực thi, bất kể ngoại lệ có được ném ra hay không.
- Nếu một câu lệnh return được thực thi trong khối try hoặc catch, khối finally vẫn sẽ được thực thi.

Correct!

- Khối finally chỉ được thực thi nếu có ngoại lệ được ném ra trong khối try.

Question 25

1 / 1 pts

Mục đích của từ khóa `throw` trong Java là gì?

Correct!

- Để ném một ngoại lệ một cách rõ ràng từ phương thức hoặc khối mã.
- Để tạo ra một loại ngoại lệ tùy chỉnh.
- Để bắt một ngoại lệ được ném ra bởi phương thức khác.
- Để định nghĩa các ngoại lệ mà một phương thức có thể ném ra.

Question 26

1 / 1 pts

Điều nào sau đây đúng về **lớp generic** trong Java?

- Lớp generic chỉ có thể nhận các kiểu dữ liệu nguyên thủy làm tham số kiểu.
- Lớp generic chỉ có thể được sử dụng với các kiểu dữ liệu tích hợp sẵn như String hoặc Integer.

Correct!

- Lớp generic cho phép bạn định nghĩa các lớp, giao diện và phương thức với tham số kiểu.
- Một lớp generic chỉ có thể nhận một tham số kiểu.

Question 27

1 / 1 pts

Cú pháp đúng để khai báo **phương thức generic** trong Java là gì?

- public <T> void method(T[] args)
- public void <T> method(T arg)

Correct!

- public <T> void method(T arg)
- public <T> T method(void arg)

Question 28
1 / 1 pts

Lợi ích của việc sử dụng **generics** trong Java là gì?

Correct!

Generics cho phép tái sử dụng mã bằng cách cung cấp cơ chế định nghĩa các lớp, giao diện và phương thức có thể hoạt động trên bất kỳ kiểu dữ liệu nào.

- Generics cho phép bạn chỉ sử dụng các kiểu dữ liệu nguyên thủy, điều này làm cho mã chạy nhanh hơn.
- Generics giúp giảm bớt nhu cầu sử dụng các khối try-catch trong xử lý ngoại lệ.
- Generics cung cấp hiệu suất tốt hơn vì chúng cho phép tối ưu hóa cấp thấp.

Question 29
1 / 1 pts

Đoạn mã khai báo lớp generic sau có nghĩa là gì?

```
public class Box<T> {
    private T value;
    public void setValue(T value) {
        this.value = value;
    }
    public T getValue() {
        return value;
    }
}
```

- Lớp Box chỉ có thể chứa các giá trị kiểu String.
- Lớp Box không thể được sử dụng với các kiểu dữ liệu nguyên thủy.
- Lớp Box chỉ có thể chứa các giá trị có kiểu dữ liệu số.

Correct!

- Lớp Box là một lớp generic có thể chứa bất kỳ kiểu đối tượng nào được chỉ định khi khởi tạo.

Question 30

1 / 1 pts

Làm thế nào để tạo một đối tượng của lớp generic `Box` để chứa một đối tượng kiểu `Integer`?

- [ ] Box box = new Box<Integer>();

Correct!

- [x] Box<Integer> box = new Box<Integer>();
- [ ] Box box = new Box();

Correct!

- [x] Box<Integer> box = new Box<>();

Quiz Score: 29 out of 30