Bài tập 20.

20.1

Trong cơ chế quản lý bộ nhớ của C#, khi một đối tượng trong Gen 0 được di chuyển lên Gen 1, điều gì xảy ra?

A. Đối tượng đó sẽ được xóa ngay lập tức nếu không có tham chiếu.

B. Đối tượng đó sẽ được thu gom trong lần thu hồi tiếp theo của Gen 0.

**C. Đối tượng đó sẽ được thu gom trong lần thu hồi tiếp theo của Gen 1.**

D. Đối tượng đó sẽ được di chuyển trực tiếp lên Gen 2.

20.2

Khi nào bộ sưu tập rác toàn bộ (Full GC) được thực hiện trong .NET?

A. Khi không còn đủ bộ nhớ trong Gen 0 và Gen 1.

**B. Khi không còn đủ bộ nhớ trong Gen 2.**

C. Khi tất cả các đối tượng trong Gen 1 đều được tham chiếu.

D. Khi không còn đủ bộ nhớ trong Large Object Heap (LOH).

20.3

Cơ chế nào sau đây được sử dụng để giảm thiểu thời gian dừng của ứng dụng khi thực hiện thu gom rác trong .NET?

A. Workstation GC

B. Server GC

**C. Concurrent GC**

D. Ephemeral GC

**Bài tập 21. so sánh OOP trong Java và OOP trong C# theo 4 nguyên lý lập trình của OOP**

so sánh OOP trong Java và C# theo 4 nguyên lý cơ bản của lập trình hướng đối tượng (OOP), gồm:

Tính đóng gói (Encapsulation)

Tính kế thừa (Inheritance)

Tính đa hình (Polymorphism)

Tính trừu tượng (Abstraction)

Tính đóng gói (Encapsulation):

Java và C# đều hỗ trợ tính đóng gói bằng cách cho phép bạn đóng gói dữ liệu và phương thức vào trong các lớp.

Cả Java và C# đều cung cấp các từ khóa truy cập (access modifiers) như public, private, protected để kiểm soát mức độ truy cập vào các thành phần trong lớp.

Tính kế thừa (Inheritance):

Cả Java và C# đều hỗ trợ kế thừa, cho phép một lớp con kế thừa từ một lớp cha.

Cú pháp khai báo lớp con khác nhau một chút: Java dùng từ khóa extends còn C# dùng từ khóa :.

Java chỉ hỗ trợ đơn kế thừa (một lớp con chỉ có thể kế thừa từ một lớp cha), trong khi C# hỗ trợ cả đơn kế thừa và đa kế thừa (một lớp có thể kế thừa từ nhiều lớp cha khác nhau).

Tính đa hình (Polymorphism):

Cả Java và C# đều hỗ trợ tính đa hình, cho phép các đối tượng của các lớp khác nhau có thể thực thi cùng một phương thức theo cách riêng của chúng.

Cú pháp khai báo phương thức ảo (virtual) và ghi đè (override) phương thức cũng khác nhau một chút giữa hai ngôn ngữ.

Tính trừu tượng (Abstraction):

Cả Java và C# đều hỗ trợ tính trừu tượng thông qua các lớp trừu tượng (abstract class) và giao diện (interface).

Cú pháp khai báo lớp trừu tượng và giao diện khác nhau một chút giữa hai ngôn ngữ.

Một điểm khác biệt là C# hỗ trợ phương thức trừu tượng trong giao diện, trong khi Java không.

**Bài tập 22. Nêu sự khác biệt của Java và C# khi xử lý các phương thức mà có thể bị ghi đè (override) trong các lớp con. Cho ví dụ.**

Giải:

- Java: Không có từ khóa 'virtual'. Mặc định các phương thức có thể bị ghi đè trừ khi được khai báo là 'final' hoặc 'private'.

- C#: Sử dụng từ khóa 'virtual' để khai báo một phương thức có thể bị ghi đè và 'override' để ghi đè phương thức đó trong lớp con.

Ví dụ:

class Animal {

void makeSound() {

System.out.println("Some sound");

}

}

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Woof");

}

}

C#

public class Animal {

public virtual void MakeSound() {

Console.WriteLine("Some sound");

}

}

public class Dog : Animal {

public override void MakeSound() {

Console.WriteLine("Woof");

}

}

Bài tập 23. So sánh cách sử dụng của instanceof của Java và is của C#.

Giải:

Cả hai từ khóa instanceof trong Java và is trong C# đều phục vụ mục đích tương tự là kiểm tra kiểu đối tượng. Chúng được sử dụng trong các ngữ cảnh rất giống nhau và cú pháp của chúng cũng tương đối tương tự, với khác biệt chủ yếu nằm ở ngôn ngữ và cú pháp chi tiết của từng ngôn ngữ.

Java: chạy code sau trong java online hoặc Netbeans

class Animal {

//...

}

class Dog extends Animal {

//...

}

public class Main {

public static void main(String[] args) {

Animal animal = new Dog();

if (animal instanceof Dog) {

System.out.println("animal is an instance of Dog");

} else {

System.out.println("animal is not an instance of Dog");

}

if (animal instanceof Animal) {

System.out.println("animal is an instance of Animal");

} else {

System.out.println("animal is not an instance of Animal");

}

}

}

C#: chạy code sau trong C# online hoặc VS 2022

public class Animal {

//...

}

public class Dog : Animal {

//...

}

public class Program {

public static void Main(string[] args) {

Animal animal = new Dog();

if (animal is Dog) {

Console.WriteLine("animal is an instance of Dog");

} else {

Console.WriteLine("animal is not an instance of Dog");

}

if (animal is Animal) {

Console.WriteLine("animal is an instance of Animal");

} else {

Console.WriteLine("animal is not an instance of Animal");

}

}

}

Bài tập 24. So sánh cách sử dụng của super của Java và base của C# khi vận dụng kế thừa.

Java: chạy code Java sau:

class Parent {

void display() {

System.out.println("Parent method");

}

}

class Child extends Parent {

void display() {

super.display(); // Calls the Parent's display method

System.out.println("Child method");

}

}

public class Main {

public static void main(String[] args) {

Child child = new Child();

child.display();

}

}

C#:chạy code C# sau

public class Parent {

public virtual void Display() {

Console.WriteLine("Parent method");

}

}

public class Child : Parent {

public override void Display() {

base.Display(); // Calls the Parent's Display method

Console.WriteLine("Child method");

}

}

public class Program {

public static void Main(string[] args) {

Child child = new Child();

child.Display();

}

}

giải:

Việc sử dụng super trong Java và base trong C# khi vận dụng kế thừa có một số điểm tương đồng và khác biệt như sau:

Java:

Trong lớp Child, phương thức display() gọi đến phương thức display() của lớp cha Parent bằng cách sử dụng super.display().

Khi gọi child.display() trong Main, nó sẽ thực hiện cả phương thức display() của lớp cha Parent và lớp con Child.

C#:

Trong lớp Child, phương thức Display() gọi đến phương thức Display() của lớp cha Parent bằng cách sử dụng base.Display().

Khi gọi child.Display() trong Main, nó cũng sẽ thực hiện cả phương thức Display() của lớp cha Parent và lớp con Child.

Điểm tương đồng:

Cả Java và C# đều cho phép gọi phương thức của lớp cha từ lớp con bằng cách sử dụng super (Java) và base (C#).

Việc sử dụng super hoặc base giúp cho lớp con có thể truy cập và thực thi phương thức của lớp cha, nhằm tái sử dụng code và đảm bảo tính kế thừa.

Điểm khác biệt:

Cú pháp khai báo sử dụng super (Java) và base (C#) khác nhau.

Java chỉ hỗ trợ đơn kế thừa, do đó super chỉ có thể gọi đến phương thức của lớp cha trực tiếp. Trong khi C# hỗ trợ cả đơn kế thừa và đa kế thừa, nên base có thể gọi đến phương thức của bất kỳ lớp cha nào.