OOPConcept

C++

```cpp
#include <iostream>
using namespace std;

class Animal {
public:
    Animal() {
        cout << "create animal" << endl;
    }

    virtual ~Animal() {
        cout << "destroy animal" << endl;
    }

    virtual void sound() {
        cout << "make a sound" << endl;
    }
};

class Dog : public Animal {
public:
    Dog() {
        cout << "dog" << endl;
    }

    ~Dog() {
        cout << "destroy dog" << endl;
    }
    void sound() override {
        cout << "barks" << endl;
    }
};

class Cat : public Animal {
public:
    Cat() {
        cout << "cat" << endl;
    }

    ~Cat() {
        cout << "destroy cat" << endl;
    }

    void sound() override {
        cout << "meows" << endl;
    }
};
int main() {
    // สร้างออบเจกต์ของ Animal, Dog, และ Cat
    Animal* a = new Animal();  // เรียก constructor ของ Animal
    Animal* d = new Dog();     // เรียก constructor ของ Dog
    Animal* c = new Cat();     // เรียก constructor ของ Cat

    // เรียกใช้งานเมธอด sound() ของแต่ละออบเจกต์ตามลำดับ
    a->sound();  // จะพิมพ์ "make a sound"
    d->sound();  // จะพิมพ์ "barks"
    c->sound();  // จะพิมพ์ "meows"

    // ลบออบเจกต์เพื่อเรียก destructor
    delete a;
    delete d;
    delete c;

    return 0;
}
```

Python

```python
class Animal:
    def __init__(self):
        print("create animal")

    def sound(self):
        print("make a sound")

    def __del__(self):
        print("destroy animal")

class Dog(Animal):
    def __init__(self):
        super().__init__()
        print("dog")

    def sound(self):
        print("barks")

    def __del__(self):
        print("destroy dog")

class Cat(Animal):
    def __init__(self):
        super().__init__()
        print("cat")

    def sound(self):
        print("meows")

    def __del__(self):
        print("destroy cat")
a = Animal()  # เรียก constructor ของ Animal
d = Dog()     # เรียก constructor ของ Dog
c = Cat()     # เรียก constructor ของ Cat

# เรียกใช้งานเมธอด sound() ของแต่ละออบเจกต์ตามลำดับ
a.sound()  # จะพิมพ์ "make a sound"
d.sound()  # จะพิมพ์ "barks"
c.sound()  # จะพิมพ์ "meows"

# ลบออบเจกต์เพื่อเรียก destructor
del a
del d
del c
```

Java

```java
public class Animal {
    Animal(){
        System.out.println("create animal");
    }

    public void sound(){
        System.out.println("make a sound");
    }

    @Override
    protected void finalize() throws Throwable {
        try {
            System.out.println("destroy animal");
        } finally {
            super.finalize();
        }
    }

    public static void main(String[] args) {
        // สร้างออบเจกต์ของ Animal, Dog, และ Cat
        Animal a = new Animal(); // เรียก constructor ของ Animal
        Animal d = new Dog();    // เรียก constructor ของ Dog
        Animal c = new Cat();    // เรียก constructor ของ Cat

        // เรียกใช้งานเมธอด sound() ของแต่ละออบเจกต์ตามลำดับ
        a.sound(); // จะพิมพ์ "make a sound"
        d.sound(); // จะพิมพ์ "barks"
        c.sound(); // จะพิมพ์ "meows"

        // ช่วยให้ Garbage Collector ทำงาน (ไม่รับประกันว่าจะเรียก finalize() ทันที)
        a = null;
        d = null;
        c = null;
        System.gc(); // ขอให้ Garbage Collector ทำงาน
    }
}
class Dog extends Animal {
    Dog(){
        System.out.println("dog");
    }

    @Override
    public void sound(){
        System.out.println("barks");
    }

    @Override
    protected void finalize() throws Throwable {
        try {
            System.out.println("destroy dog");
        } finally {
            super.finalize();
        }
    }
}
```

```java
class Cat extends Animal {
    Cat(){
        System.out.println("cat");
    }

    @Override
    public void sound(){
        System.out.println("meows");
    }

    @Override
    protected void finalize() throws Throwable {
        try {
            System.out.println("destroy cat");
        } finally {
            super.finalize();
        }
    }
}
```

1. การจัดการหน่วยความจำ (Memory Management):

C++:

-Manual Memory Management: ต้องใช้ new เพื่อสร้างออบเจกต์ และ delete เพื่อปลดปล่อยหน่วยความจำ

-Destructor: ถูกเรียกทันทีเมื่อใช้ delete หรือเมื่อออบเจกต์ถูกทำลายเมื่อสร้างบน stack

Python:

-Automatic Memory Management: ใช้ Garbage Collection อัตโนมัติ ไม่ต้องจัดการหน่วยความจำเอง

-Destructor: ใช้เมธอด __del__() ซึ่งจะถูกเรียกเมื่อออบเจกต์ถูกลบหรือ garbage collected

Java:

-Automatic Memory Management: ใช้ Garbage Collection อัตโนมัติ ไม่ต้องจัดการหน่วยความจำเอง

-Destructor: ไม่มี destructor จริง ใช้ finalize() ซึ่งถูก deprecated ตั้งแต่ Java 9 และไม่ควรใช้งานในโปรแกรมจริง

2. Destructor/Finalizer:

C++:

-มี destructor ที่ถูกเรียกเมื่อออบเจกต์ถูกทำลาย

-สามารถใช้งานเพื่อปลดปล่อยทรัพยากรที่จัดสรรด้วยตัวเอง

Python:

-ใช้เมธอด __del__() เพื่อทำงานคล้าย destructor

-ถูกเรียกเมื่อออบเจกต์ถูกลบหรือ garbage collected

Java:

-ใช้เมธอด finalize() ซึ่งถูก deprecated และไม่ควรใช้งาน

-ไม่มีวิธีการที่แน่นอนในการปลดปล่อยทรัพยากรเมื่อออบเจกต์ถูกทำลาย

-ควรใช้วิธีการอื่น เช่น การใช้ try-with-resources หรือเมธอด close()

3. การประกาศและใช้งานตัวชี้ (Pointers):

C++:

-ใช้ pointers อย่างชัดเจน เช่น Animal* a = new Animal();

-ต้องจัดการหน่วยความจำเองด้วย delete

Python:

-ไม่มี concept ของ pointers แบบชัดเจน ออบเจกต์ถูกจัดการโดยอ้างอิง (references) โดยอัตโนมัติ

-ไม่ต้องใช้ delete หรือทำการปลดปล่อยหน่วยความจำเอง

Java:

-ใช้ references สำหรับการอ้างอิงออบเจกต์ เช่น Animal a = new Animal();

-ไม่ใช้ pointers แบบ C++ และไม่ต้องจัดการหน่วยความจำเอง

4. การใช้ super และ virtual:

C++:

-เมธอดที่ต้องการ override ควรประกาศเป็น virtual ในคลาสแม่

-ใช้ override ในคลาสลูกเพื่อความชัดเจนPython:

Python:

-ใช้ super() เพื่อเรียก constructor ของคลาสแม่

-เมธอดทั้งหมดที่ไม่ถูกประกาศเป็น @staticmethod หรือ @classmethod จะถูกพิจารณาว่าเป็น virtual โดยอัตโนมัติ

Java:

-เมธอดทั้งหมดที่ไม่ถูกประกาศเป็น final, static, หรือ private จะถูกพิจารณาว่าเป็น virtual โดยอัตโนมัติ

-ใช้ @Override เพื่อระบุการ override เมธอดจากคลาสแม่

5. การเรียก Constructor ของคลาสแม่:

C++:

-Constructor ของคลาสแม่จะถูกเรียกโดยอัตโนมัติก่อน constructor ของคลาสลูก

-สามารถระบุ constructor ของคลาสแม่ใน initializer list ได้

Python:

-ใช้ super().__init__() เพื่อเรียก constructor ของคลาสแม่ในคลาสลูก

Java:

-Constructor ของคลาสแม่จะถูกเรียกโดยอัตโนมัติผ่าน super() หากไม่ระบุ

-สามารถระบุ constructor ของคลาสแม่ได้โดยใช้ super() พร้อมพารามิเตอร์ถ้าจำเป็น

6. การพิมพ์ (Output):

C++:

-ใช้ std::cout และ << ในการพิมพ์ข้อความ

Python:

-ใช้ฟังก์ชัน print() ในการพิมพ์ข้อความ

Java:

-ใช้ System.out.println() ในการพิมพ์ข้อความ

สรุปความแตกต่างหลักระหว่าง C++, Python, และ Java:

การจัดการหน่วยความจำ:

C++: ต้องจัดการเองด้วย new และ delete

Python: อัตโนมัติด้วย Garbage Collection

Java: อัตโนมัติด้วย Garbage Collection แต่ไม่มี destructor จริง

Destructor/Finalizer:

C++: มี destructor ชัดเจนที่ถูกเรียกเมื่อออบเจกต์ถูกทำลาย

Python: ใช้ __del__() ซึ่งถูกเรียกเมื่อออบเจกต์ถูกลบหรือ garbage collected

Java: ไม่มี destructor จริง ใช้ finalize() ซึ่งถูก deprecated และไม่ควรใช้งาน

การใช้ Pointers/References:

C++: ใช้ pointers แบบชัดเจน

Python: ใช้อ้างอิงโดยอัตโนมัติ ไม่มี pointers แบบ C++

Java: ใช้อ้างอิงแบบปลอดภัย ไม่มี pointers แบบ C++

การ Override เมธอด:

C++: ต้องใช้ virtual ในคลาสแม่ และ override ในคลาสลูก

Python: เมธอดถูกพิจารณาว่าเป็น virtual โดยอัตโนมัติ ใช้ super()

Java: เมธอดถูกพิจารณาว่าเป็น virtual โดยอัตโนมัติ ใช้ @Override และ super()

การเรียก Constructor ของคลาสแม่:

C++: ผ่าน initializer list

Python: ผ่าน super().__init__()

Java: ผ่าน super() ภายใน constructor

การพิมพ์ข้อความ:

C++: std::cout << "ข้อความ" << std::endl;

Python: print("ข้อความ")

Java: System.out.println("ข้อความ");

สรุป:

ทั้งสามภาษา C++, Python, และ Java มีหลักการ OOP เช่น การสืบทอดคลาส (Inheritance) และการ override เมธอด แต่แตกต่างกันที่การจัดการหน่วยความจำ การใช้งาน destructor/finalizer และวิธีการจัดการออบเจกต์ โดย:

C++ ให้ความยืดหยุ่นสูงในการจัดการหน่วยความจำและมี destructor ที่ชัดเจน

Python ง่ายต่อการใช้งานด้วยการจัดการหน่วยความจำอัตโนมัติและมี __del__() สำหรับการปลดปล่อยทรัพยากร

Java ใช้ Garbage Collection อัตโนมัติ ไม่มี destructor จริง แต่สามารถใช้วิธีการอื่นในการจัดการทรัพยากร

II การเปรียบเทียบส่วนประกอบของโค้ดกับ Concept ของ OOP มีดังนี้:

1. Class (คลาส)

นิยาม: คลาสคือโครงสร้างที่เป็นแม่แบบ (blueprint) สำหรับสร้างออบเจกต์ (object) ซึ่งกำหนดคุณสมบัติ (properties/fields) และพฤติกรรม methods) ของออบเจกต์นั้น ๆ

```java
public class Animal {
    Animal() {
        System.out.println("create animal");
    }
    public void sound() {
        System.out.println("make a sound");
    }
}
```

2. Object, Instance

นิยาม: ออบเจกต์คืออินสแตนซ์ของคลาส ซึ่งสร้างขึ้นจากแม่แบบที่คลาสกำหนด ออบเจกต์จะมีคุณสมบัติและพฤติกรรมตามที่คลาสกำหนดไว้

```java
Animal a = new Animal();
Dog d = new Dog();
Cat c = new Cat();
```

3. Subclass, Derived Class

นิยาม: ซับคลาสคือคลาสที่สืบทอดคุณสมบัติและพฤติกรรมจากคลาสแม่ (superclass) หรือคลาสหลัก (base class) โดยสามารถเพิ่มพฤติกรรมหรือคุณสมบัติใหม่ได้ หรือเปลี่ยนพฤติกรรมเดิมของคลาสแม่

```java
class Dog extends Animal {
    Dog() {
        System.out.println("dog");
    }
    @Override
    public void sound() {
        System.out.println("barks");
    }
}
class Cat extends Animal {
    Cat() {
        System.out.println("cat");
    }
    @Override
    public void sound() {
        System.out.println("meows");
    }
}
```

4. Message (ข้อความ)

นิยาม: ข้อความใน OOP หมายถึงการเรียกใช้งานเมธอด (method) หรือการสื่อสารกับออบเจกต์ ซึ่งทำได้โดยการส่งข้อความ (message) ไปยังออบเจกต์เพื่อให้มันทำงานตามพฤติกรรมที่กำหนดไว้ในเมธอด

```
a.sound();
d.sound();
c.sound();
```
D

5. Inheritance (การสืบทอด)

นิยาม: การสืบทอดหมายถึงการที่คลาสลูก (subclass) สามารถสืบทอดคุณสมบัติและพฤติกรรมจากคลาสแม่ (superclass) โดยที่คลาสลูกสามารถนำสิ่งที่ได้มาพัฒนาหรือปรับปรุงเพิ่มเติมได้

```
class Dog extends Animal {
    @Override
    public void sound() {
        System.out.println("barks");
    }
}
class Cat extends Animal {
    @Override
    public void sound() {
        System.out.println("meows");
    }
}
```

6. Polymorphism

นิยาม: Polymorphism หมายถึงความสามารถในการใช้เมธอดเดียวกันในออบเจกต์ที่มีลักษณะแตกต่างกัน หรือคลาสที่แตกต่างกัน โดยทำให้การเรียกใช้งานเมธอดในซับคลาสมีผลลัพธ์ต่างจากคลาสแม่

```
Animal a = new Dog();
a.sound();  // จะพิมพ์ "barks"

Animal b = new Cat();
b.sound();  // จะพิมพ์ "meows"
```

III. abstract class

นิยามของ Abstract Class:

-Abstract Class คือคลาสที่ไม่สามารถสร้างอินสแตนซ์ (instance) ได้โดยตรง

-ใช้เป็นแม่แบบ (blueprint) สำหรับคลาสลูก (subclasses) ที่ต้องการสืบทอดคุณสมบัติและพฤติกรรมจากคลาสแม่

-สามารถมี abstract methods (เมธอดที่ไม่มีการดำเนินการภายในคลาสแม่) ที่ต้องถูก override ในคลาสลูก

-นอกจากนี้ยังสามารถมีเมธอดที่มีการดำเนินการ (concrete methods) ได้เช่นกัน

ในตัวอย่างโค้ดข้างต้นในข้อ I ไม่มี class ใดเลยที่เป็น abstract class แต่ถ้าจะแก้ไขให้เป็น abstract class ก็สามารถเปลี่ยนได้โดยวิธีดังนี้

1.ประกาศคลาส Animal เป็น abstract:

```
public abstract class Animal {
```

2.ประกาศเมธอด sound() เป็น abstract:

```
public abstract void sound();
```

3.การปรับปรุงคลาสลูก (Dog และ Cat):

คลาสลูกต้อง implement เมธอด sound() ที่เป็น abstract ในคลาสแม่:

```
@Override
public void sound(){
    System.out.println("barks");
}
```

และ

```
@Override
public void sound(){
    System.out.println("meows");
}
```

ข้อดีของการใช้ Abstract Class:

-การบังคับใช้งาน: บังคับให้คลาสลูกต้อง implement เมธอดที่เป็น abstract ซึ่งช่วยให้แน่ใจว่าทุกคลาสลูกมีพฤติกรรมที่ต้องการ

-การจัดโครงสร้าง: ช่วยให้โครงสร้างของโปรแกรมชัดเจนและเป็นระบบ โดยมีคลาสแม่ที่กำหนดรูปแบบพื้นฐานและคลาสลูกที่มีการดำเนินการเฉพาะตัว

-การขยายระบบ: ทำให้การเพิ่มคลาสลูกใหม่ทำได้ง่ายและมีความยืดหยุ่นในการขยายระบบ