

1. Lợi ích của Tính Đa hình (Polymorphism)

Cho phép xử lý các đối tượng Media như Book, CompactDisc, DigitalVideoDisc một cách thống nhất, không cần quan tâm cụ thể đó là loại đối tượng nào

Giúp dễ mở rộng: thêm một loại Media mới chỉ cần kế thừa Media mà không cần sửa code cũ.

Giảm phụ thuộc: chương trình chỉ cần thao tác với Media, không cần phụ thuộc vào lớp con cụ thể.

2. Kế thừa (Inheritance) giúp đạt được Đa hình (Polymorphism) như thế nào

Các lớp Book, Disc, CompactDisc đều kế thừa từ Media.

Vì kế thừa, các lớp con sẽ override phương thức (print(), play()...) theo cách riêng.

Khi ta gọi phương thức qua tham chiếu kiểu Media, Java sẽ tự động chọn phương thức phù hợp lúc runtime.

3. Phân biệt giữa Tính Đa hình và Kế thừa

- Mục đích

+ Kế thừa (Inheritance): Tái sử dụng code, mở rộng thêm tính năng.

+ Đa hình (Polymorphism): Cho phép một phương thức dùng cho nhiều đối tượng khác nhau.

- Khi nào diễn ra

+ Kế thừa (Inheritance): Khi thiết kế lớp (compile-time).

+ Đa hình (Polymorphism): Khi chương trình chạy (runtime).

- Mối quan hệ giữa lớp

+ Kế thừa (Inheritance): Quan hệ lớp con - lớp cha.

+ Đa hình (Polymorphism): Nhiều đối tượng khác nhau dùng chung một giao diện.

Ví dụ

+ Kế thừa (Inheritance): CompactDisc kế thừa Disc, Disc kế thừa Media.

+ Đa hình (Polymorphism): media.print() tự động gọi đúng Book.print() hoặc CD.print() tùy loại đối tượng.

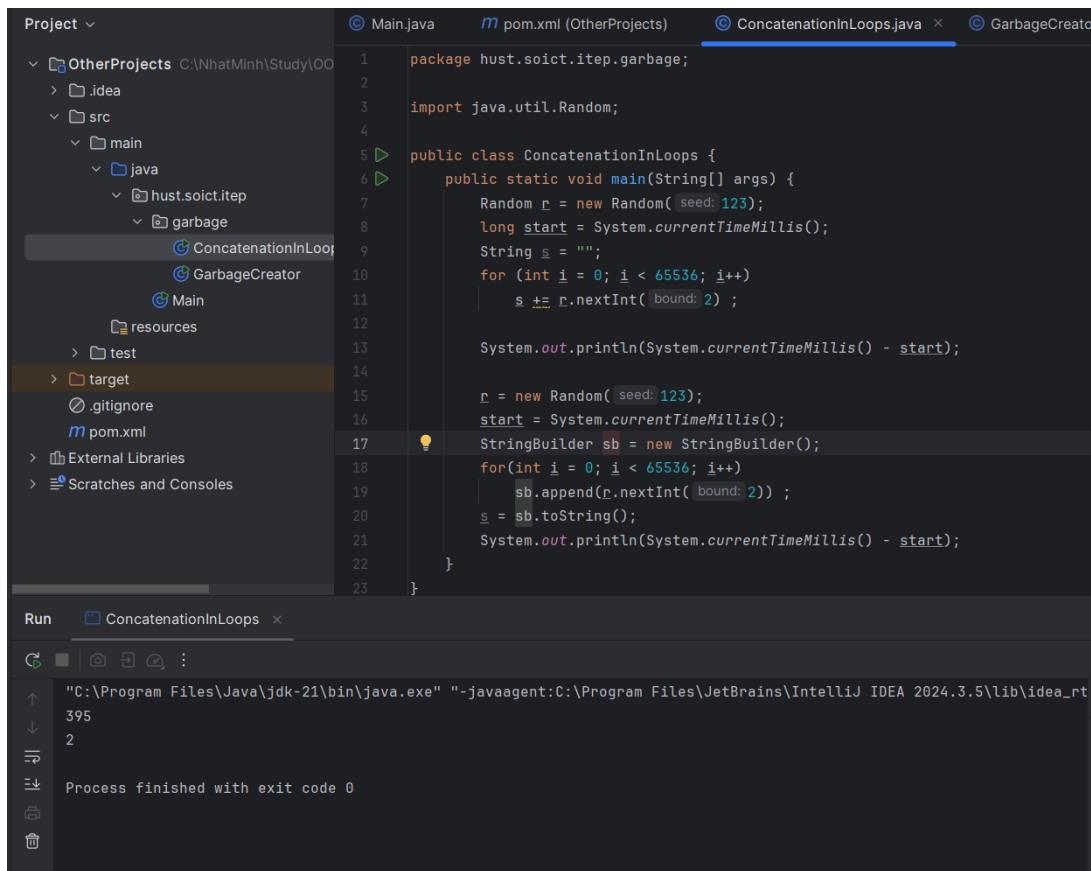
4. So sánh thời gian chạy

- Cấu hình:

+ JDK 21.0.6 phần mềm chạy IntelliJ Idea

- Phần cứng: 24GB-526GB, chip i7-13620H

- So sánh:



```
1 package hust.soict.itcp.garbage;
2
3 import java.util.Random;
4
5 public class ConcatenationInLoops {
6     public static void main(String[] args) {
7         Random r = new Random( seed: 123);
8         long start = System.currentTimeMillis();
9         String s = "";
10        for (int i = 0; i < 65536; i++)
11            s += r.nextInt( bound: 2) ;
12
13        System.out.println(System.currentTimeMillis() - start);
14
15        r = new Random( seed: 123);
16        start = System.currentTimeMillis();
17        StringBuilder sb = new StringBuilder();
18        for(int i = 0; i < 65536; i++)
19            sb.append(r.nextInt( bound: 2)) ;
20        s = sb.toString();
21        System.out.println(System.currentTimeMillis() - start);
22    }
23 }
```

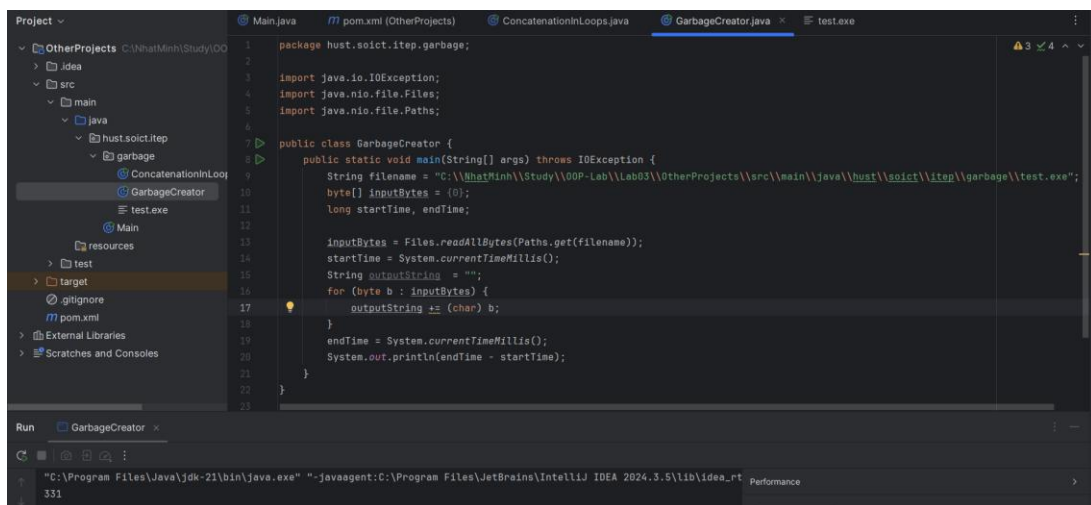
Run ConcatenationInLoops x

"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3.5\lib\idea_rt" 395 2

Process finished with exit code 0

Khi để s là String thì thời gian chạy là 395 ms, khi chuyển sang StringBuilder thì mất 2 ms

- Với 1 file test.exe

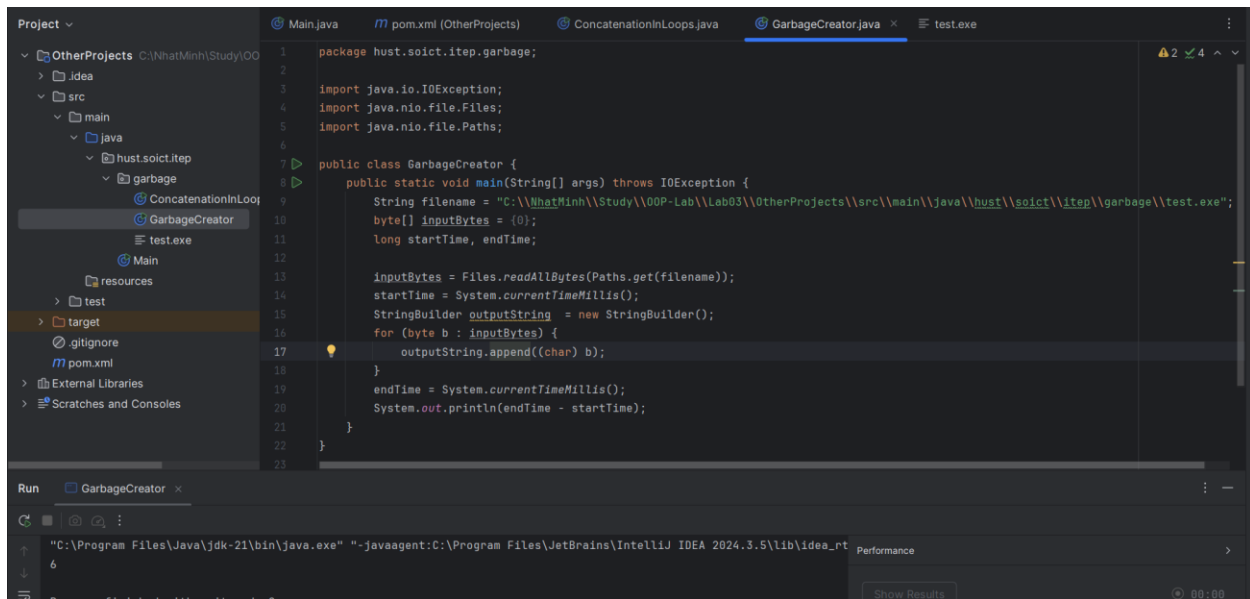


```
1 package hust.soict.itcp.garbage;
2
3 import java.io.IOException;
4 import java.nio.file.Files;
5 import java.nio.file.Paths;
6
7 public class GarbageCreator {
8     public static void main(String[] args) throws IOException {
9         String filename = "C:\\\\hatMinh\\Study\\00P-Lab\\Lab03\\OtherProjects\\src\\main\\java\\hust\\soict\\itcp\\garbage\\test.exe";
10        byte[] inputBytes = Files.readAllBytes(Paths.get(filename));
11        long startTime, endTime;
12
13        inputBytes = Files.readAllBytes(Paths.get(filename));
14        startTime = System.currentTimeMillis();
15        String outputString = "";
16        for (byte b : inputBytes) {
17            outputString += (char) b;
18        }
19        endTime = System.currentTimeMillis();
20        System.out.println(endTime - startTime);
21    }
22 }
23 }
```

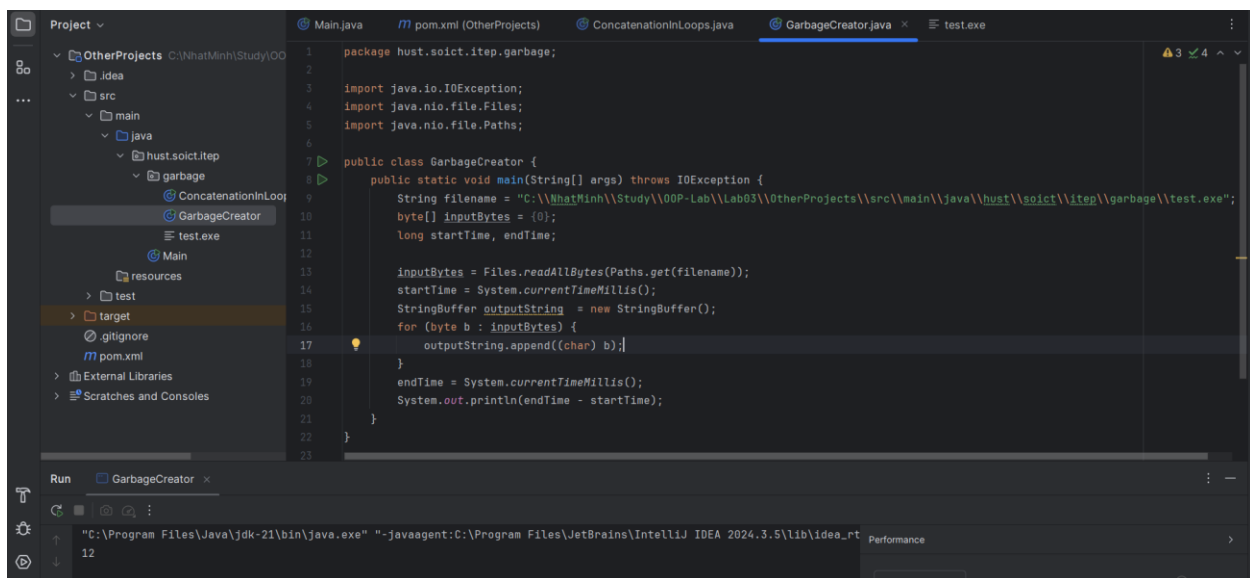
Run GarbageCreator x

"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3.5\lib\idea_rt" Performance 331

- Dùng String mất 331 ms



- Dùng StringBuilder mất 6 ms



- Dùng StringBuffer mất 12ms

Giải thích:

- String là immutable: Mỗi lần thay đổi chuỗi, Java tạo một đối tượng mới → tốn bộ nhớ và chậm nếu lặp nhiều lần.
- StringBuffer – có thể thay đổi và thread-safe
 - + Là mutable: Có thể thay đổi nội dung mà không tạo đối tượng mới.
 - + Thread-safe: Các phương thức được đồng bộ hóa (synchronized), nên dùng được trong môi trường đa luồng.

- `StringBuilder` – giống `StringBuffer` nhưng không thread-safe
- + Cũng mutable, hiệu suất tốt hơn `StringBuffer` khi không cần đồng bộ, nhanh hơn trong luồng đơn