What are the advantages of Polymorphism?

Polymorphism – hay còn gọi là **tính đa hình**, là khả năng cho phép một đối tượng thực thi nhiều hành vi khác nhau, tùy thuộc vào ngữ cảnh. Dưới đây là những lợi ích nổi bật:

- **Tăng tính linh hoạt và khả năng mở rộng**: Code có thể xử lý các đối tượng có cùng kiểu cha mà không cần biết chính xác kiểu con là gì.
- **Giảm độ phức tạp, tăng khả năng tái sử dụng**: Một hàm có thể hoạt động với nhiều loại đối tượng khác nhau, giúp tiết kiệm thời gian và công sức khi mở rộng hệ thống.
- **Dễ bảo trì, dễ nâng cấp**: Khi thay đổi hoặc thêm hành vi, chỉ cần chỉnh sửa trong lớp con, không ảnh hưởng đến lớp cha hay các phần khác.
- **Tối ưu hóa thiết kế hướng đối tượng**: Kết hợp chặt chẽ với inheritance (kế thừa) và interface (giao diện), giúp thiết kế phần mềm theo hướng mở rộng chứ không sửa đổi (Open/Closed Principle).

- How is Inheritance useful to achieve Polymorphism in Java?

Inheritance (kế thừa) là yếu tố nền tảng để đạt được **runtime polymorphism** trong Java.

- Khi một lớp con kế thừa từ một lớp cha, nó **có quyền ghi đè (override)** các phương thức của lớp cha.
- Ta có thể tạo biến kiểu lớp cha nhưng gán cho đối tượng kiểu lớp con. Khi gọi phương thức, **Java sẽ gọi phương thức đúng của lớp con tại thời điểm chạy** (runtime), không phải tại thời điểm biên dịch (compile time).

Ví dụ:

```
class Animal {
    void sound() { System.out.println("Animal makes a sound"); }
}


class Dog extends Animal {
    void sound() { System.out.println("Dog barks"); }
```

}

Animal a = new Dog();

a.sound(); // In ra: Dog barks

=> Đây chính là tính đa hình động (dynamic polymorphism) – cốt lõi của đa hình trong Java, và kế thừa là cầu nối giúp đạt được điều đó.

- What are the differences between Polymorphism and Inheritance in Java?

Tiêu chí	**Polymorphism (Đa hình)**	**Inheritance (Kế thừa)**
Khái niệm	Một đối tượng có thể có nhiều hình thái khác nhau.	Một lớp có thể kế thừa thuộc tính/phương thức từ lớp khác.
Mục đích	Cho phép thực hiện các hành vi khác nhau qua cùng một giao diện.	Tái sử dụng code và mở rộng chức năng của lớp hiện có.
Loại	Có thể là **compile-time** (overloading) hoặc **runtime** (overriding).	Là một mối quan hệ cha-con giữa các lớp.
Liên quan	Thường sử dụng cùng với inheritance và interface.	Cần thiết để đạt được runtime polymorphism.
Ví dụ	Gọi draw() trên một mảng các đối tượng Shape (Circle, Square, v.v.).	Lớp Dog kế thừa từ Animal.
Mức độ linh hoạt	Rất linh hoạt trong thiết kế – dễ mở rộng hành vi.	Dễ tái sử dụng và tổ chức lại logic chung.