* Vòng đời activity onCreat -> onStart - > onResume - Run Activity  
    
  -> onPause -> onStop - > stopted
* -> onRestart - > onStart -> onResume -> Run Activity
* 4 viduj về tính chất OOP

Đóng gói : public, protected,private  
kế thừa , kế thừa thuộc tính và phương thức   
đa hình

* So sánh class với itf
* Abstract class với class
* Database room
* Broadcast Receiver

BroadcastReceiver là một thành phần trong hệ thống Android được sử dụng để nhận và phản ứng với các thông điệp (broadcasts) từ hệ thống hoặc các ứng dụng khác. Các thông điệp này có thể là các sự kiện hệ thống như khi thiết bị hoàn thành khởi động, khi pin yếu, khi có tin nhắn mới, hoặc các sự kiện tùy chỉnh từ các ứng dụng khác.

* Service

, OOP lấy ví dụ 4 tính chất , so sánh class vs itf , abstract class vs class , database room , broadcast receiver , service  
  
  
  
Interface và abstract class đều được sử dụng để định nghĩa các phương thức mà các lớp con phải triển khai, nhưng chúng có một số điểm khác biệt quan trọng:

### 1. Kế thừa (Inheritance)

- \*\*Abstract Class\*\*: 1 lớp chỉ có thể kế thừa 1 lớp abstract

- \*\*Interface\*\*: Một lớp con có thể triển khai nhiều interface.’

### 2. Phương thức (Methods)

- \*\*Abstract Class\*\*: Có thể chứa cả phương thức trừu tượng (không có phần thân) và phương thức thông thường (có phần thân).

- \*\*Interface\*\*: Chỉ chứa các phương thức trừu tượng (trước Kotlin 1.2). Từ Kotlin 1.2 trở đi, interface có thể chứa các phương thức mặc định (có phần thân).

### 3. Thuộc tính (Properties)

- \*\*Abstract Class\*\*: Có thể chứa các thuộc tính với các giá trị mặc định hoặc không.

- \*\*Interface\*\*: Chỉ chứa các thuộc tính trừu tượng (không có giá trị mặc định) trước Kotlin 1.2. Từ Kotlin 1.2 trở đi, interface có thể chứa các thuộc tính với các giá trị mặc định.

### 4. Khởi tạo (Initialization)

- \*\*Abstract Class\*\*: Có thể có constructor và có thể chứa logic khởi tạo.

- \*\*Interface\*\*: Không thể có constructor.

### Tóm lại

- \*\*Abstract Class\*\*: Dùng khi bạn muốn chia sẻ code giữa các lớp con, có thể chứa logic khởi tạo và các phương thức thông thường.

- \*\*Interface\*\*: Dùng khi bạn muốn định nghĩa một hợp đồng mà các lớp phải tuân theo, hỗ trợ đa kế thừa và có thể chứa các phương thức mặc định từ Kotlin 1.2 trở đi.

Trong Android, có bốn thành phần chính, mỗi thành phần có tác dụng riêng biệt:

 **Activity**:

* **Tác dụng**: Là thành phần đại diện cho một màn hình giao diện người dùng. Mỗi Activity thường tương ứng với một cửa sổ hoặc một màn hình trong ứng dụng. Nó quản lý giao diện người dùng và tương tác với người dùng.

 **Service**:

* **Tác dụng**: Là thành phần chạy ngầm để thực hiện các tác vụ dài hạn hoặc các tác vụ không yêu cầu giao diện người dùng. Ví dụ: phát nhạc, tải dữ liệu từ mạng, hoặc xử lý dữ liệu trong nền.

 **Broadcast Receiver**:

* **Tác dụng**: Là thành phần cho phép ứng dụng nhận và phản hồi các thông báo hệ thống hoặc các sự kiện phát sóng từ các ứng dụng khác. Ví dụ: nhận thông báo khi pin yếu, khi có tin nhắn mới, hoặc khi kết nối mạng thay đổi.

 **Content Provider**:

* **Tác dụng**: Là thành phần cho phép chia sẻ dữ liệu giữa các ứng dụng khác nhau. Nó cung cấp một giao diện chuẩn để truy cập và thao tác dữ liệu, chẳng hạn như danh bạ, hình ảnh, hoặc các dữ liệu khác được lưu trữ trong ứng dụng.

**1. Phân biệt JavaScript và Node.js**

* **JavaScript**:
  + **Định nghĩa**: JavaScript là một ngôn ngữ lập trình đa năng được sử dụng chủ yếu để tạo ra các trang web tương tác.
  + **Nền tảng**: Chạy trên trình duyệt web của người dùng (client-side).
  + **Sử dụng**: Tạo hiệu ứng động, xử lý sự kiện, xác thực dữ liệu trên form, thao tác với DOM, v.v.
* **Node.js**:
  + **Định nghĩa**: Node.js là một môi trường runtime cho phép bạn chạy JavaScript trên máy chủ (server-side).
  + **Nền tảng**: Chạy trên máy chủ.
  + **Sử dụng**: Xây dựng các ứng dụng phía máy chủ như API, server HTTP, ứng dụng thời gian thực, v.v.

**2. Node.js là gì?**

Node.js là một môi trường runtime dựa trên V8 JavaScript engine của Chrome, cho phép chạy JavaScript trên máy chủ. Nó sử dụng mô hình event-driven và không chặn (non-blocking), rất hiệu quả cho các ứng dụng I/O mạnh mẽ và thời gian thực.

**3. Liệt kê những lợi ích chính của việc sử dụng Node.js**

* **Hiệu suất cao**: Do sử dụng V8 engine và mô hình không đồng bộ, Node.js có tốc độ xử lý nhanh.
* **Không chặn (Non-blocking)**: Cho phép xử lý nhiều yêu cầu cùng lúc mà không bị chặn.
* **Khả năng mở rộng**: Dễ dàng mở rộng ứng dụng bằng cách thêm vào các module.
* **Hệ sinh thái phong phú**: NPM cung cấp hàng nghìn thư viện và module có sẵn.
* **Cộng đồng lớn**: Có nhiều tài liệu, hỗ trợ và ví dụ từ cộng đồng lập trình viên.

**4. Sự khác biệt giữa React.js/Vue.js và Node.js là gì?**

* **React.js/Vue.js**:
  + **Định nghĩa**: Là các thư viện/framework JavaScript cho việc xây dựng giao diện người dùng (UI) phía khách hàng.
  + **Nền tảng**: Chạy trên trình duyệt.
  + **Sử dụng**: Tạo ra các ứng dụng web tương tác và động.
* **Node.js**:
  + **Định nghĩa**: Là một môi trường runtime để chạy JavaScript trên máy chủ.
  + **Nền tảng**: Chạy trên máy chủ.
  + **Sử dụng**: Xây dựng backend cho các ứng dụng web, xử lý yêu cầu API, v.v.

**5. Node.js đơn hay đa luồng?**

Node.js là đơn luồng (single-threaded).

**6. Tại sao Node.js lại đơn luồng?**

Node.js sử dụng mô hình đơn luồng để xử lý các yêu cầu I/O không đồng bộ và không chặn (non-blocking). Mô hình này cho phép Node.js quản lý các yêu cầu một cách hiệu quả mà không phải tạo ra nhiều luồng (threads) cho mỗi yêu cầu, giúp tiết kiệm tài nguyên hệ thống.

**7. Node.js hoạt động như thế nào?**

Node.js sử dụng một vòng lặp sự kiện (event loop) để quản lý và xử lý các yêu cầu. Khi một yêu cầu đến, nó sẽ được đưa vào hàng đợi sự kiện và được xử lý khi tài nguyên sẵn sàng, mà không chặn các yêu cầu khác.

**8. Node.js có thể được sử dụng ở đâu?**

* **API Server**: Xây dựng các dịch vụ web và RESTful API.
* **Ứng dụng thời gian thực**: Chat, game, thông báo tức thì.
* **Server-side Rendering**: Render trang web trên máy chủ.
* **Ứng dụng IoT**: Kết nối và quản lý các thiết bị IoT.
* **Microservices**: Tạo và quản lý các microservices.

**9. Có bao nhiêu function API trong Node.js?**

Node.js có hàng trăm hàm API, tổ chức trong nhiều module khác nhau như fs, http, crypto, path, events, v.v. Số lượng chính xác có thể thay đổi theo từng phiên bản Node.js.

**10. Sự khác biệt giữa không đồng bộ và không chặn là gì?**

* **Không đồng bộ (Asynchronous)**: Là khi một tác vụ có thể bắt đầu và hoàn thành mà không cần phải chờ các tác vụ khác. Kết quả của tác vụ không đồng bộ sẽ được xử lý sau khi tác vụ hoàn thành.
* **Không chặn (Non-blocking)**: Là khi một tác vụ không ngăn cản hoặc chặn các tác vụ khác. Nó cho phép các tác vụ khác tiếp tục mà không phải chờ tác vụ hiện tại hoàn thành.

**11. package.json là gì?**

package.json là một tệp cấu hình trong Node.js chứa thông tin về dự án, bao gồm các dependencies, scripts, phiên bản, tên dự án, tác giả, v.v. Nó cho phép quản lý và cài đặt các gói npm một cách dễ dàng.

**12. Bạn hiểu thế nào về lập trình hướng sự kiện (event driven programming)?**

Lập trình hướng sự kiện là một mô hình lập trình trong đó luồng điều khiển của chương trình được xác định bởi các sự kiện như đầu vào từ người dùng, tín hiệu từ phần cứng, hoặc thông báo từ các dịch vụ khác. Các hàm callback hoặc listeners được đăng ký để xử lý các sự kiện khi chúng xảy ra.

**13. Vòng lặp sự kiện (event loop) trong Node.js nó hoạt động như thế nào?**

Vòng lặp sự kiện (event loop) là trung tâm của Node.js, quản lý và xử lý các sự kiện và yêu cầu I/O không đồng bộ. Khi một yêu cầu đến, nó được đưa vào hàng đợi sự kiện. Vòng lặp sự kiện sẽ kiểm tra hàng đợi và xử lý từng yêu cầu một cách không chặn, chuyển tiếp các tác vụ I/O cho hệ điều hành và xử lý kết quả khi sẵn sàng, giúp Node.js duy trì hiệu suất cao và khả năng xử lý đồng thời nhiều yêu cầu.

Jpa   
  
controller -=> service = > repository

