1. Java SE, Standard Editor, thường đc dùng để lập trình những console application, UI application…
2. JRE (Java runtime environment) : giúp thực thi JVM.
3. JDK (Java development Kit) : Nó gồm JRE +công cụ phát triển.
4. 1 object là 1 instance của 1 lớp
5. Một class là một nhóm các object mà có các thuộc tính chung. Class là một Template hoặc bản thiết kế từ đó object được tạo. Một class trong Java có thể bao gồm:

* Thành viên dữ liệu
* Phương thức
* Constructor
* Block
* Lớp và Interface

1. Interface tương tự như class nhưng chỉ chứa các phương thức trừu tượng, thể hiện tính trừu tượng hoàn toàn và đa kế thừa trong Java. Khác với class ở các điểm :

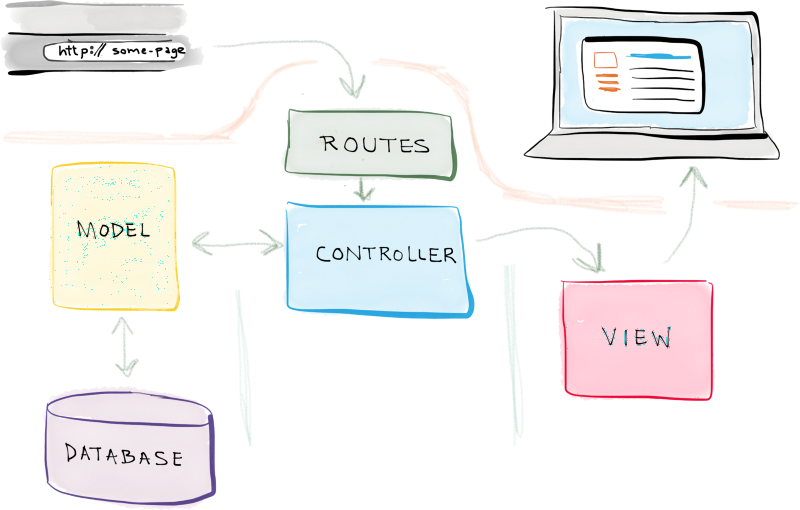
* Không thể khởi tạo một interface.
* Một interface không chứa bất cứ hàm contructor nào.
* Tất cả các phương thức của interface đều là abstract.
* Một interface không thể chứa một trường nào trừ các trường vừa static và final.
* Một interface không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
* Một interface có thể kế thừa từ nhiều interface khác.

1. The shadowing begins when the local variable is declared
2. Try-with-resource :

Cấu trúc try cho phép khai báo nhiều resource. Resource là 1 object mà phải được closed sau khi chương trình sử dụng xong resource đó.

Try-with-resource đảm bảo mỗi resource được closed tại cuối statement

1. MVC (Model – View – Controller) : frame work :

M : cấu trúc dữ liệu theo cách tin cậy và chuẩn bị dữ liệu theo lệnh của controller.

V : Hiển thị dữ liệu cho người dùng theo cách dễ hiểu dựa trên hành động của người dùng.

C : Nhận lệnh từ người dùng, gửi lệnh đến cho Model để cập nhập dữ liệu, truyền lệnh đến View để cập nhập giao diện hiển thị.

1. Spring FrameWork Overview

Spring là 1 framework giúp các nhà phát triển xây dựng những hệ thống và ứng dụng chạy trên JVM một cách đơn giản, nhanh chóng và mềm dẻo hơn.

**Spring Core** là thành phần cốt lõi của Spring Framework để xưng dựng nên các thành phần khác.

1. Dependency Invertion

Là 1 nguyên lý để thiết kế và viết code.

1. Invertion of Control (IOC)

Đây là một [design pattern](https://toidicodedao.com/2016/03/01/nhap-mon-design-pattern-phong-cach-kiem-hiep/) được tạo ra để code có thể tuân thủ nguyên lý Dependency Inversion. Có nhiều cách hiện thực pattern này: ServiceLocator, Event, [Delegate](https://toidicodedao.com/2015/02/10/series-c-hay-ho-callback-trong-c-delegate-action-predicate-func/), … Dependency Injection là một trong các cách đó.

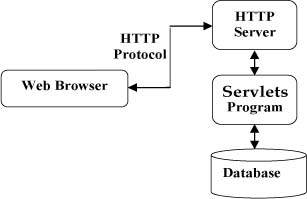
1. Dependency Injection (DI)

Đây là một cách để hiện thực Inversion of Control Pattern (Có thể coi nó là một design pattern riêng cũng được). Các **module phụ thuộc (dependency) sẽ được inject** vào module cấp cao.

https://toidicodedao.com/2015/11/03/dependency-injection-va-inversion-of-control-phan-1-dinh-nghia/

1. Thuật ngữ :

Servlet :



Servlet container :

Rest :

1. Maven:

<https://o7planning.org/vi/10131/huong-dan-su-dung-maven-cho-nguoi-moi-bat-dau>

1. React :

/\*Debug with VS Code\*/

https://www.youtube.com/watch?v=UI7dpnVoad8

React là 1 thư viện của Javascript, dùng để build user interfaces. Nó giúp ta sinh ra các khối UI, gọi là **components** (**React.Component**).

Chúng ta dùng **components** để thông báo với React về những gì ta muốn in ra màn hình.

let và const : ES6 : 2 new ways to declare variables.

let dùng { } làm containers

React.Component

|

Components and Props

|

State and Lifecycle

1. **Components**

Components giúp ta chia các UI thành các phần riêng biệt. Về mặt khái niệm, Components giống như các function của JavaScript, cũng nhận các input động (**props – properties**) và return React element – mô tả những gì hiện lên màn hình.

Khai báo 1 component theo 2 cách :

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

Với ES6 (ECMAScript 6, đây được coi là một tập hợp các kỹ thuật nâng cao của Javascript và là phiên bản mới nhất của chuẩn ECMAScript) thì ta có thể làm như dưới đây :

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

const element = <Welcome name="Sara" age = "21t"/>;

ReactDOM.render(element,document.getElementById("root"));

1. We call ReactDOM.render() with the <Welcome name="Sara" /> element.
2. React calls the Welcome component with {name: 'Sara'} as the props.
3. Our Welcome component returns a <h1>Hello, Sara</h1> element as the result.
4. React DOM efficiently updates the DOM to match <h1>Hello, Sara</h1>.
5. **State and LifeCycle**

**State :** Mỗi component đều có 1 state. Mỗi lần state đó thay đổi (**setState**), React sẽ tự động re-render() state mới đó.

**Vòng đời của 1 React Component :**

<https://viblo.asia/p/vong-doi-cua-mot-react-component-RQqKLMRzZ7z>

* **constructor(props)**

Được gọi khi một thể hiện của component được tạo ra.

Có thể dùng để khởi tạo state cho component.

Cũng có thể dùng để "bind" các hàm của component.

Nếu phải cài đặt hàm này thì phải khai báo 1 tham số props cho nó và phải gọi super(props) đầu tiên.

Nếu không làm gì thì không phải cài đặt hàm này.

**Adding LifeCycle to a Class** :

Mounting and unmounting

s

* **componentWillMount()**

Được gọi trước khi render().

Dùng để đăng ký các sự kiện toàn cục.

Dựa vào các props để tính toán và set lại state

* **render()**
* **componentDidMount()**
* **componentWillReceiveProps(nextProps)**
* **shouldComponentUpdate(nextProps, nextState)**
* **componentWillUpdate(nextProps, nextState)**
* **componentDidUpdate(prevProps, prevState)**
* **componentWillUnmount()**