용어정리

용어	의미
Environment	컴퓨터 시스템이나 프로그램이 동작하는 데 필요한 모든 하드웨어, 소프트웨어 및 시스템 구성 요소//
Closure	프로그래밍에서 함수와 그 함수가 선언된 언어 환경(Lexical Environment)의 조합
Scope	변수나 함수의 유효 범위
Dynamic	프로그래밍에서 실행 시간(Runtime)에 데이터 타입이나 변수의 값 등이 결정되는 것
Static type	프로그래밍에서 변수에 미리 정해진 데이터 타입을 지정하여 사용하는 방식
Symbol table	컴파일러나 인터프리터 등에서 변수나 함수와 같은 식별자(Identifier)에 대한 정보를 저장하는 자료 구조
Tree Structure	계층적인 구조를 나타내는 것으로, 상위 요소와 하위 요소의 관계를 트리(Tree) 구조로 표현
Binding Time	변수나 함수와 같은 이름이나 값이 실제 메모리상에서 할당되는 시점을 의미
Container	프로그램 실행환경에서 다른 프로세스나 모듈을 포함하는 구조를 의미
Thread	프로세스 내에서 실행되는 실행 단위로, 하나의 프로세스는 여러 개의 스레드를 가질 수 있습니다. 스레드는 하나의 프로세스 내에서 공유되는 자원, 즉 메모리 공간을 사용하므로, 스레드 간의 자원 공유 및 동기화 문제를 고려
Thread Safe	스레드 세이프(Thread Safe)란 멀티스레드 환경에서 여러 스레드가 동시에 공유하는 자원(변수, 함수, 객체 등)에 대한 접근을 제어하여, 스레드 간의 경쟁 상황으로 인한 오류를 방지하는 것
Context	프로그램이 실행되는 동안 해당 프로그램의 실행 상태를 나타내는 정보입니다. 이 정보에는 프로그램 카운터, 레지스터 값, 메모리 할당 정보 등이 포함
Dependency Injection	객체 지향 프로그래밍에서 객체간의 결합도(Coupling)를 낮추기 위해 사용하는 설계 패턴입니다. 디펜던시 인젝션은 객체가 필요로 하는 의존성(서비스, 객체 등)을 외부에서 제공받아 객체 내부에서 생성하지 않고 주입해주는 것
언어	무한집합
레귤러 익스프레션	언어를 정의하는 수학적 언어
유한 오토마타	레귤러 익스프레션에 대한 머신
정규표현	포말(형식을 갖춘)랭귀지- 형식언어를 4개의 카테고리로 분류 //레귤러랭귀지를 포함한 언어를컨텍스트
Chomsky 4계층	0. 재귀적 열거 언어 : 튜링 기계로 인식할 수 있는 언어, 가장 바깥에서 기계어를 알아들수 있는 말로 번역 1. 프리 랭귀지 2.컨텍트 센서티브 랭귀지

용어	의미
대부분의 프로그래밍 언어는 프리랭귀지를 벗어나지 못함.	
머신	전산학에선 자동으로 동작할수 있는 수학적 계체를 의미 = 오토마타(jvm는 자바란 컨텍스트 프리 랭귀지를 사용하는 머신)
레귤러 랭귀지	
컨텍트 센서티비 랭귀지	리뉴얼 바운디드 오토마타
튜어링 랭귀지	컴퓨터로 계산 가능한 랭귀지
컴퓨터 사이언스	수학의 일부다 튜어링 머신으로 계산못하면 컴퓨터 분야가 아니다.수학적으로 증명된 사실이다.
정규 표현	정규표현 운영 분야 1. 어노테이션에 매핑 달 때(*.do에서 ,*은 무슨 의미일까)
집합	1. 공집합 : 이걸 표현할 래귤러 랭귀지표현(아무것도 안쓰면 된다...)
정규 표현	문자열의 규칙(ex)전화번호)을 정의하면 특정 파이날 오토마타를 넣을 수 있다.
정규 표현식(regular expression)	은 문자열의 패턴을 정의하는데 사용되는 간결한 문법입니다. 정규 표현식은 특정 규칙(예: 전화번호, 이메일 주소 등)을 따르는 문자열을 찾거나 검증하는 데 사용됩니다. 여기서 언급한 "파이날 오토마타"라는 용어는 일반적으로 사용되지 않으며, "유한 오토마타(finite automata)"를 의미하는 것으로 추정됩니다.
유한 오토마타는 정규 언어	를 인식하는 기계로, 정규 표현식과 같은 패턴을 처리하는데 사용됩니다. 유한 오토마타는 상태, 입력 심볼, 상태 전환 함수, 초기 상태, 그리고 종료 상태 집합으로 구성됩니다. 유한 오토마타는 문자열을 읽으면서 상태를 전환하고, 최종적으로 종료 상태에 도달하면 해당 문자열이 정규 언어에 속한다고 판단합니다.
정규 표현식을 사용	하면 문자열의 규칙을 정의할 수 있으며, 이 규칙에 따라 유한 오토마타를 구성할 수 있습니다. 유한 오토마타는 정규 표현식에 정의된 패턴을 처리하고 검증하는 데 사용되므로, 특정 문자열이 주어진 정규 표현식에 부합하는지 확인할 수 있습니다. 이를 통해 문자열 검색, 치환, 검증 등의 작업을 수행할 수 있습니다.
알파벳	[a-z]이걸 쓸수 있다는 건 a~z까지 있다는 정의가 있고, 0자 이상 와야 한다. 0번 이상 온다는 정의를 ' * '
정규표현모르면	학장님왈 VI못쓴다 챗 지피티 정규 표현식(regular expression)에 익숙하지 않아도 VI 편집기를 사용할 수는 있습니다. 하지만, 정규 표현식을 알고 있다면 VI 편집기에서 훨씬 더 효과적으로 작업을 수행할 수 있습니다 VI 편집기는 리눅스와 유닉스 환경에서 널리 사용되는 강력한 텍스트 편집기입니다. 정규 표현식을 사용하면 VI에서 다양한 문자열 검색, 치환, 삭제 등의 작업을 쉽게 수행할 수 있습니다

용어	의미
Inversion Of Controller(IC)	
Dependency Injection(DI)	
디자인 패턴	오랜 시간동안 검증된 코딩 기법들을 모아놓은 것
프레임워크	어떤 형태에 맞춰서 코드를 짜라. 양식에 맞춰서 짜라
Bean과 Object	둘은 동네같은 개념이다.
Spring 프레임워크	빈 같은 녀석들을 만들수 있도록 하는 프레임웍
개발자는	Bean동네에 들어가기 위한 몇가지를 준비해야한다.
Spring 동네용어	wire,composition,wiring
Spring 동네에서 쓰는 것	Dependency Injection, IoC(제어의 역전)
IoC	제어의 역전
스프링 컨테이너	
cohesion,coupling	
동작하는 오브젝트의 라이프사이클 시멘스틱이 다르다	컨텍스트, 환경
Spring Bean	만드는 방식, 관리하는방식, 라이프사이클 등 자바와 다르다
팩토리 메소드	제어의 역전, Dependency Injection
개체를 사용자가 만들때	물체를 만들때 물체에 대한 정보를 개발자가 정확히 알아야한다는 문제,coupling을 때야 cohesion이 올라감
밀집도가 높은 상태	외부와 소통할 방법은 남아야함 => 인터페이스
	의존성을 스프링컨테이너로 넘김
	wiring을 스프링 컨테이너로 넘김
reflection	
information hiding	컴퓨터를 예로, 새로운 컴퓨터 왔다고, 컴퓨터를 뜯어서, 덧셈 연산을 시킨다고 조작하면 위험하다. 그래서 인포메이션 하이딩으로 안전을 지킨다.
정보은닉	불필요한 정보를 노출하지않아서, 정보의 밀집도와 정보의 뭐라고???
추상화	타입정의의 개념으로 쓰면 안된다
abstract class, interface	타입정의는 인터페이스, 필요한걸 끌어쓰려면 abstract class

용어	의미
delegation	의존성이 준다.
Critical Section	스레드는 콘쿼런트, 특정한 코드가 하나만 싱글 스레드로 돌땐 시퀀설하다. 여러개의 스레드가 특정 코드를 돌아가며 쓰면, 시퀀설이 깨진다. 뭐를 크리티컬 섹션을 지정하는데, 싱크로나이즈 블럭, 싱크로나이즈 메서드, 어떤 경우엔 세마포어를 못쓴다.
모니터	세마포어보다 추상화가 올라간다.
량데뷰	
비지터 패턴	비지터 패턴 : a,b,c,d 함수가 다 적용됨. a를 받아 a가 진짜 a야? 그럼 afa를 출력해if a야 너 진짜 a야? ab RTTI(런타임 타입 인포메이션) 자바는 타입안에 다른 타입을 섬밋하고있다. 디스패치 온 타입을 디스패치한다. 새로운 레포가 생기면 레포를 위한 ff를 만드러야하고 코드를 고쳐야하는데, 고칠수없는 코드면 새 프로젝트를 만들어야한다. 파괴적, 그럼 비파괴적으로 확장해야함. => overiding,a가 실제 실행될때 actual 타입을 보고 진짜를 적용시켜줘 다이나믹 메서드

챗 지피티 정리 본

항목	설명
Chomsky 4계층	
0. 재귀적 열거 언어	튜링 기계로 인식할 수 있는 언어, 기계어를 알아들을 수 있는 말로 번역
1. 프리 랭귀지	문맥에 상관없이 문법 구조를 따르는 언어. 대부분의 프로그래밍 언어가 이 범주에 속함
2. 컨텍스트 센서티브 랭귀지	문맥에 따라 문법 구조가 달라지는 언어, 리뉴얼 바운디드 오토마타를 사용
머신	전산학에서 자동으로 동작할 수 있는 수학적 계체를 의미, 오토마타 (예: JVM은 컨텍스트 프리 랭귀지인 자바를 사용하는 머신)
레귤러 랭귀지	정규 언어라고도 하며, 정규 표현식을 사용해 인식할 수 있는 언어
컨텍스트 센서티브 랭귀지	리뉴얼 바운디드 오토마타를 사용하는 언어
튜어링 랭귀지	튜링 기계로 계산 가능한 모든 언어
컴퓨터 사이언스	수학의 일부로, 튜링 머신으로 계산할 수 없는 문제는 컴퓨터 분야로 보지 않음. 이는 수학적으로 증명된 사실이다.
정규 표현	정규 표현식은 문자열에 대한 패턴을 표현하는데 사용되며, 어노테이션에 매핑 시 사용됨 (예: *.do에서, *는 모든 문자를 의미)
집합	
1. 공집합	래귤러 랭귀지 표현: 아무것도 없는 문자열

항목	설명
정규 표현	문자열의 규칙(예: 전화번호)을 정의하며, 특정 유한 오토마타로 처리할 수 있다.
정규 표현식	문자열 패턴을 정의하는 간결한 문법으로, 특정 규칙(예: 전화번호, 이메일 주소 등)을 따르는 문자열을 찾거나 검증하는 데 사용됨
유한 오토마타	정규 언어를 인식하는 기계로, 정규 표현식과 같은 패턴을 처리하는데 사용됨. 상태, 입력 심볼, 상태 전환 함수, 초기 상태, 종료 상태 집합으로 구성됨.
유한 오토마타와 정규 언어	유한 오토마타는 문자열을 읽으면서 상태를 전환하고, 최종적으로 종료 상태에 도달하면 해당 문자열이 정규 언어에 속한다고 판단합니다.
정규 표현식 사용	문자열의 규칙을 정의하고, 유한 오토마타를 구성해 패턴 처리 및 검증에 사용. 문자열 검색, 치환, 검증 등의 작업을 수행할 수 있음
알파벳	[a-z]: a~z까지의 문자가 있다는 정의, 0자 이상 올 수 있음. 0번 이상 오는 것을 '*'로 표현
정규 표현식과 VI 편집기	정규 표현식에 익숙해지면 VI 편집기에서 더 효과적으로 작업 수행 가능. VI에서 문자열 검색, 치환, 삭제 등의 작업을 쉽게 수행할 수 있음
Inversion Of Control (IoC)	객체의 생성, 조립, 관리를 외부에서 담당함으로써 모듈 간의 결합도를 낮추고 유연성을 높이는 디자인 원칙
Dependency Injection (DI)	객체 간의 의존 관계를 외부에서 설정해주는 것으로, Inversion of Control의 한 형태로 모듈 간의 결합도를 낮추고 유연성을 높이는 방법
디자인 패턴	오랜 시간 동안 검증된 코딩 기법들을 모아놓은 것으로, 효율적인 소프트웨어 설계를 돕는 일련의 공식화된 솔루션
프레임워크	특정 형태나 양식에 맞춰 코드를 작성할 수 있도록 미리 정의된 구조와 규칙을 제공하는 소프트웨어 개발 도구. 개발자의 생산성과 코드의 재사용성을 높이는데 도움을 줌
Bean과 Object	둘은 비슷한 개념으로, Bean은 스프링 컨테이너가 관리하는 객체(Object)를 의미합니다
Spring 프레임워크	Bean 객체를 생성하고 관리할 수 있도록 도와주는 프레임워크로, 개발자의 생산성과 코드의 유연성을 향상시킵니다
개발자는	스프링 프레임워크에서 Bean을 사용하기 위해 구성 정보를 준비하고 관리해야 합니다
Spring 동네용어	wire, composition, wiring은 스프링에서 Bean 간의 의존성을 설정하고 구성하는 과정을 의미합니다
Spring 동네에서 쓰는 것	Dependency Injection (DI)와 Inversion of Control (IoC)는 스프링 프레임워크에서 핵심적으로 사용되는 디자인 패턴입니다. 이를 통해 코드의 유연성과 확장성이 향상됩니다.
IoC (Inversion of Control)	제어의 역전이란 프로그램의 흐름을 개발자가 아닌 프레임워크가 관리하는 것을 의미합니다. 이를 통해 개발자는 비즈니스 로직에 집중할 수 있으며, 코드의 유지보수가 용이해집니다. 스프링에서는 IoC를 통해 객체의 생성, 구성 및 관리를 스프링 컨테이너가 담당합니다.
스프링 컨테이너	객체의 생성, 구성 및 관리를 담당하는 스프링 프레임워크의 핵심 구성 요소. IoC와 DI 패턴을 구현합니다

항목	설명
Cohesion	소프트웨어 설계에서의 응집력은 한 모듈 내부의 요소들이 얼마나 밀접하게 관련되어 있는지를 나타냅니다. 높은 응집력은 각 모듈이 명확한 역할과 책임을 가지며, 이는 유지보수와 확장성을 향상시키는 데 도움이 됩니다.
Coupling	결합도는 소프트웨어 설계에서 두 모듈 간의 상호 의존성을 나타냅니다. 낮은 결합도는 각 모듈이 독립적으로 동작할 수 있어 유지보수와 확장성에 이점을 줍니다. Inversion of Control (IoC) 및 Dependency Injection (DI) 패턴을 사용하면 결합도를 낮출 수 있습니다.
동작하는 오브젝트의 라이프사이클 시멘스틱이 다르다	컨텍스트와 환경에 따라 객체의 라이프사이클이 다를 수 있습니다. 이는 객체가 생성, 사용, 소멸되는 방식에 영향을 줍니다.
Spring Bean	스프링 Bean은 스프링 컨테이너가 관리하는 객체로, 생성 방식, 관리 방식, 라이프사이클 등이 일반 자바 객체와 다릅니다. 이를 통해 객체 간의 의존성이 낮아지고 코드의 유연성이 향상됩니다.
팩토리 메소드	팩토리 메소드는 객체 생성의 책임을 분리하고, 제어의 역전 및 Dependency Injection을 구현하는 데 사용되는 디자인 패턴입니다. 이를 통해 객체 생성과 사용이 느슨하게 결합되어 코드의 유지보수가 용이해집니다.
개체를 사용자가 만들 때	객체를 만들 때 개발자가 객체에 대한 정보를 정확히 알아야 하는 문제가 발생할 수 있으며, 이는 결합도를 높일 수 있습니다. 결합도를 낮추고 응집도를 높이려면 의존성 주입 및 제어의 역전 같은 기술을 사용하여 객체 간의 관계를 최소화해야 합니다.
밀집도가 높은 상태	외부와 소통할 방법은 남겨두어야 합니다. 이를 해결하기 위해 인터페이스를 활용하는 것이 좋습니다.
의존성을 스프링 컨테이너로 넘김	스프링 프레임워크는 IoC와 DI를 이용하여 객체 간의 의존성을 낮춥니다. 의존성을 스프링 컨테이너로 넘겨 객체의 생성, 구성, 관리 등을 스프링 컨테이너가 담당합니다.
wiring을 스프링 컨테이너로 넘김	스프링 컨테이너는 XML 또는 어노테이션 기반으로 wiring 작업을 수행하며, 객체 간의 관계를 설정합니다. wiring을 스프링 컨테이너로 넘겨 객체 간의 결합도를 낮출 수 있습니다.
reflection	객체의 정보를 동적으로 가져오고 수정할 수 있는 기능을 제공합니다. Java에서는 Class 클래스를 활용하여 reflection을 구현합니다.
information hiding	객체 지향 프로그래밍에서 객체의 내부 데이터나 구현 정보를 외부로부터 감추는 것을 의미합니다. 예를 들어, 새로운 컴퓨터가 있다고 할 때, 컴퓨터를 뜯어서 덧셈 연산을 실행하면 위험할 수 있습니다. 그래서 정보 은닉을 통해 안전을 지킵니다.
정보은닉	불필요한 정보를 노출하지 않아서, 정보의 밀집도와 정보의 응집도를 감소시켜 유지보수성을 향상시킵니다.
추상화	객체 지향 프로그래밍에서 추상화는 개체의 공통적인 특징을 뽑아내어 공통의 추상 클래스를 정의하는 것을 말합니다. 이는 객체 간의 결합도를 낮추고, 코드의 재사용성을 높이는 데 기여합니다. 그러나 추상화를 타입 정의의 개념으로 쓰면 개념이 헷갈리게 되어 프로그래밍의 이해가 어려워질 수 있기 때문에 추상화를 타입 정의의 개념으로 쓰지 않는 것이 좋습니다.
abstract class	객체 지향 프로그래밍에서 abstract class는 공통의 추상 클래스를 정의하는 것을 말합니다. abstract class에서는 구현되지 않은 메서드가 존재할 수 있습니다.

항목	설명
interface	객체 지향 프로그래밍에서 interface는 개체에서 구현해야할 메서드의 원형을 정의하는 것을 말합니다. 구현되지 않은 메서드를 가지고 있습니다.
delegation	객체 지향 프로그래밍에서 delegation은 객체가 다른 객체의 동작을 위임하는 것을 말합니다. 이를 통해 의존성을 줄일 수 있습니다.
Critical Section	Critical Section은 멀티 스레드 환경에서 특정 코드 블럭을 싱글 스레드처럼 실행하기 위해 사용하는 개념입니다. 여러 스레드가 특정 코드 블럭을 동시에 접근하면 시퀀스가 깨지고 데이터 손상의 위험이 있어, Critical Section으로 지정된 코드 블럭은 제한된 스레드만 접근할 수 있도록 합니다. 이를 위해 싱크로나이즈 블럭, 싱크로나이즈 메서드, 세마포어 등을 사용할 수 있습니다.
Monitor	Monitor는 Critical Section 개념의 개선 버전으로, 멀티 스레드 환경에서 특정 코드 블럭을 싱글 스레드처럼 실행하기 위해 사용하는 개념입니다. Monitor는 싱크로나이즈 블럭, 싱크로나이즈 메서드, 세마포어 등보다 높은 추상화 개념으로, 코드 블럭의 접근을 제한하고 특정 스레드가 코드 블럭에 접근할 때까지 다른 스레드는 대기하는 방식으로 데이터 손상의 위험을 줄입니다.
랑데뷰 (Rendezvous)	랑데뷰는 두 프로세스 간에 데이터를 주고받는 동기화 기술입니다. 두 프로세스 중 하나가 데이터를 생성하고 다른 하나가 그 데이터를 사용할 수 있도록 하는 것입니다.
CSP (Communicating Sequential Processes)	CSP는 병렬 프로세스의 제어와 통신을 기술하는 모델로, 각 프로세스가 서로 간의 순서를 기술하지 않고 통신하는 방식을 기술합니다.
시리얼라이저 (Serializer)	시리얼라이저는 동시에 실행되는 멀티 스레드 환경에서 한 개의 스레드만이 특정 리소스에 접근할 수 있도록 하는 동기화 기술입니다. 이를 위해 락을 걸어 특정 리소스에 접근할 수 있는 스레드를 제한하고, 다른 스레드는 대기하도록 합니다.

Container는 밖으로 나가는걸 막는 용도

Beans

스프링에서 "bean"이란 스프링 컨테이너가 관리하는 객체(인스턴스)를 의미합니다. 스프링 컨테이너는 XML 파일, Java Configuration 클래스, 어노테이션 등을 통해 빈을 정의하고 생성, 관리합니다. 스프링의 핵심 기능 중 하나인 DI(Dependency Injection)를 통해 빈들 간의 의존성을 자동으로 관리하므로 개발자는 객체 생성 및 의존성 관리에 대한 로직을 작성할 필요가 없어집니다.

스프링에서 빈을 등록하고 관리하는 방법에는 다음과 같은 방식들이 있습니다.

XML 파일에 빈 정의하기
스프링에서 가장 오래된 방법 중 하나로, XML 파일을 통해 빈을 등록하고 관리하는 방식입니다. XML 파일에 <bean> 태그를 사용하여 빈을 정의하고, 스프링 컨테이너에서 해당 XML 파일을 로딩하여 빈을 생성하고 관리합니다.

Java Configuration 클래스를 사용하여 빈 정의하기
자바 코드를 통해 빈을 정의하고 관리하는 방식입니다. 스프링 3부터 지원되는 방식으로, XML 파일을 사용하지 않고 자바 코드로 빈을 생성하고 의존성을 주입할 수 있습니다.

어노테이션을 사용하여 빈 정의하기

자바 클래스나 메서드에 특정 어노테이션을 사용하여 빈을 정의하고 관리하는 방식입니다. 주로 @Component, @Service, @Repository, @Controller 등의 어노테이션을 사용하여 빈을 등록합니다.

스프링의 빈은 싱글톤(Singleton)으로 생성되어, 여러 번 요청해도 하나의 인스턴스만 생성되어 사용됩니다. 이는 스프링의 메모리 관리 및 성능 향상을 위해 적용되는 기본적인 방식 중 하나입니다.

Dependency Injection

의존성 주입(Dependency Injection)은 객체 지향 프로그래밍에서 객체간의 결합도(Coupling)를 낮추기 위해 사용하는 설계 패턴입니다. 디펜던시 인젝션은 객체가 필요로 하는 의존성(서비스, 객체 등)을 외부에서 제공받아 객체 내부에서 생성하지 않고 주입해주는 것입니다.

스프링 프레임워크에서는 IoC(Inversion of Control) 컨테이너를 사용하여 디펜던시 인젝션을 구현합니다. IoC 컨테이너는 객체 생성, 관리, 의존성 주입 등을 자동으로 처리하며, 개발자는 객체 생성 및 관리를 위한 코드를 작성하지 않고도 IoC 컨테이너를 통해 객체를 사용할 수 있습니다.

스프링에서는 대표적으로 3가지 방식으로 의존성을 주입할 수 있습니다.

생성자 주입(Constructor Injection)
생성자를 통해 의존성을 주입하는 방식입니다. 생성자를 통해 의존성을 주입받으면 해당 객체가 생성될 때 필수적으로 주입받아야 하는 의존성을 명확히 나타낼 수 있습니다.

세터 주입(Setter Injection)
Setter 메서드를 통해 의존성을 주입하는 방식입니다. Setter 메서드를 통해 주입하는 방식이므로, 의존성이 선택적인 경우에 사용됩니다.

필드 주입(Field Injection)
필드에 직접 의존성을 주입하는 방식입니다. 이 방식은 코드가 간결하며 사용하기 쉽다는 장점이 있지만, 외부에서 해당 객체의 의존성을 주입받는 것이 어렵다는 단점이 있습니다.

의존성 주입은 객체지향 설계의 핵심 원칙인 SOLID 원칙 중 DIP(Dependency Inversion Principle)를 준수하기 위해 사용되며, 객체간의 결합도를 낮추어 코드의 유지보수성, 확장성을 향상시키는데 도움을 줍니다.

Context

컨텍스트는 일반적으로 프로세스나 스레드와 관련이 있습니다. 예를 들어, 하나의 프로세스가 여러 개의 스레드를 가지고 있을 때, 각 스레드는 공유되는 컨텍스트와 각각의 독립적인 스택을 가지고 있습니다. 스레드 간에 컨텍스트를 전환하면, 해당 스레드의 실행 상태가 저장되고 다음 스레드의 실행 상태가 로드됩니다. 이를 컨텍스트 전환(Context Switching)이라고 합니다.

컨텍스트는 시스템 호출, 인터럽트 처리, 멀티태스킹 등에 중요한 역할을 합니다. 이러한 작업들은 컨텍스트 전환을 필요로 하며, 이를 효율적으로 수행하는 것이 시스템의 성능 향상에 큰 영향을 미칩니다. 따라서, 컨텍스트 전환은 운영체제의 핵심 기능 중 하나입니다.

Thread

전산학에서 스레드(Thread)는 프로세스 내에서 실행되는 실행 단위로, 하나의 프로세스는 여러 개의 스레드를 가질 수 있습니다. 스레드는 하나의 프로세스 내에서 공유되는 자원, 즉 메모리 공간을 사용하므로, 스레드 간의 자원 공유 및 동기화 문제를 고려해야 합니다.

스레드는 다중 처리 기능을 활용하여, 병렬 처리 및 동시성 작업을 수행하는 데에 유용하게 사용됩니다. 예를 들어, 웹 서버에서 클라이언트 요청을 처리하는 데에는 다수의 스레드를 활용하여 동시에 처리할 수 있습니다. 또한, 그래픽 애니메이션과 같은 대규모 계산 작업에서도 스레드를 활용하여 병렬 처리를 수행할 수 있습니다.

스레드는 각각의 실행 경로를 갖기 때문에, 다른 스레드에 비해 상대적으로 경량화되어 있어, 생성 및 소멸 비용이 낮고, 스위칭 비용이 적어 효율적인 프로그래밍이 가능합니다. 따라서 스레드를 적극적으로 활용하는 것이 성능 향상에 큰 도움이 됩니다.

스레드를 사용하는 언어로는 C++, Java, Python 등이 있으며, 이들 언어에서는 스레드 관련 라이브러리를 제공하여 스레드 프로그래밍을 보다 쉽게 할 수 있도록 도와줍니다.

Thread Safety

스레드 세이프한 코드는 여러 스레드가 동시에 접근해도 안정적으로 동작해야 합니다. 이를 위해서는 여러 스레드가 공유하는 자원에 대한 접근을 제한하는 방법이 필요합니다. 대표적인 방법으로는 뮤텍스(Mutex), 세마포어(Semaphore), 경쟁 상황을 방지하는 알고리즘 등이 있습니다.

뮤텍스는 하나의 자원에 대한 접근을 한 번에 하나의 스레드만 가능하도록 제어합니다. 다른 스레드가 접근하려고 할 때는 뮤텍스를 획득할 수 없으므로 대기 상태에 들어갑니다. 뮤텍스를 해제하면 다음 스레드가 뮤텍스를 획득할 수 있게 됩니다.

세마포어는 뮤텍스와 유사하나, 동시에 접근 가능한 스레드의 수를 설정할 수 있습니다. 이를 통해 자원의 사용량을 제한하거나, 우선순위를 설정하여 더 중요한 스레드가 자원을 우선적으로 사용할 수 있도록 할 수 있습니다.

스레드 세이프한 코드는 멀티스레드 환경에서 안정적인 동작을 보장하므로, 대규모 시스템에서는 필수적인 기술입니다. 따라서 스레드 세이프한 코드를 작성하는 것은 중요한 개발 스킬 중 하나입니다.

Container

전산학에서 컨테이너(Container)는 프로그램 실행환경에서 다른 프로세스나 모듈을 포함하는 구조를 의미합니다. 일반적으로 운영체제 레벨에서 프로세스를 격리하고, 각 프로세스마다 독립된 메모리 공간을 할당하여 안전하게 실행되도록 보장하기 위해 사용됩니다.

컨테이너 기술은 프로그램의 이식성과 확장성을 높일 수 있으며, 동일한 프로그램이 다양한 환경에서 실행될 수 있도록 지원합니다. 가상화 기술을 이용하여, 컨테이너는 호스트 운영체제에서 격리된 가상 환경을 제공하며, 컨테이너 안에서는 프로세스와 모듈을 자유롭게 실행할 수 있습니다.

대표적으로 Docker와 Kubernetes가 컨테이너 기술을 이용한 대표적인 플랫폼입니다. Docker는 컨테이너 이미지를 생성하고 배포하는 기능을 제공하며, Kubernetes는 컨테이너를 자동으로 배치하고 관리하는 기능을 제공합니다. 이러한 컨테이너 기술은 대규모 분산 시스템과 클라우드 환경에서 주로 사용되며, DevOps와 같은 개발 방법론과 연계하여 사용됩니다.

Tree Structure와 Scope

스코프(Scope)와 나무 구조(Tree Structure)는 비슷한 개념입니다. 스코프는 변수, 함수, 객체 등이 유효한 범위를 나타내는 것으로, 해당 범위 내에서만 변수나 함수, 객체 등에 접근할 수 있습니다. 나무 구조는 계층적인 구조를 나타내는 것으로, 상위 요소와 하위 요소의 관계를 트리(Tree) 구조로 표현합니다.

스코프와 나무 구조는 비슷한 개념이므로, 스코프도 트리 구조로 표현될 수 있습니다. 전역 스코프는 트리 구조에서 뿌리(Root)에 해당하며, 각각의 함수는 자신의 부모 스코프(Parent Scope)와 자식 스코프(Child Scope)를 가지고 있습니다. 자식 스코프는 부모 스코프의 하위 요소이므로, 부모 스코프의 변수나 함수에 접근할 수 있습니다. 하지만 부모 스코프는 자식 스코프의 변수나 함수에 접근할 수 없습니다.

스코프는 나무 구조와 같은 계층적인 구조를 가지고 있으며, 이를 이용하여 변수나 함수 등의 이름 충돌을 방지하고, 코드의 유지보수성을 높일 수 있습니다. 또한, 스코프 체인(Scope Chain)을 이용하여 클로저(Closure)와 같은 고급 기능을 구현할 수 있습니다. 따라서, 스코프와 나무 구조는 프로그래밍에서 중요한 개념이며, 이를 잘 이해하고 활용하는 것이 좋습니다.

바인딩 타임

바인딩 타임(Binding Time)은 변수나 함수와 같은 이름이나 값이 실제 메모리상에서 할당되는 시점을 의미합니다. 즉, 이름이나 값이 언제 어디서 결정되는지를 나타내는 개념입니다.

바인딩 타임은 크게 컴파일 타임(Compile Time)과 런타임(Runtime)으로 나뉩니다. 컴파일 타임에서는 변수나 함수의 이름이나 타입 등이 결정되며, 코드가 실행되기 전에 이미 결정이 끝납니다. 컴파일 타임에서 결정되는 정보는 실행 중에 변경할 수 없으며, 프로그램의 성능과 안정성을 높이는 데에 중요한 역할을 합니다.

반면, 런타임에서는 변수나 함수의 값이나 메모리상의 위치 등이 결정됩니다. 런타임에서 결정되는 정보는 실행 중에 변경될 수 있으며, 동적으로 메모리를 할당하거나 객체를 생성하는 등의 작업에서 바인딩 타임이 일어납니다.

바인딩 타임은 프로그램의 동작 방식을 결정하는 중요한 요소 중 하나입니다. 따라서, 프로그래밍 언어에서는 컴파일 타임과 런타임의 개념을 잘 이해하고, 이를 활용하여 효율적인 프로그램을 작성할 수 있도록 지원합니다.

1. 다음에서 클로저를 분석하시오

```
function outerFunction() {
  var outerVar = 10;
```

```
  function innerFunction() {
    var innerVar = 20;
    console.log(outerVar + innerVar); // 30
  }

  return innerFunction;
}

var innerFunc = outerFunction();
innerFunc();
```