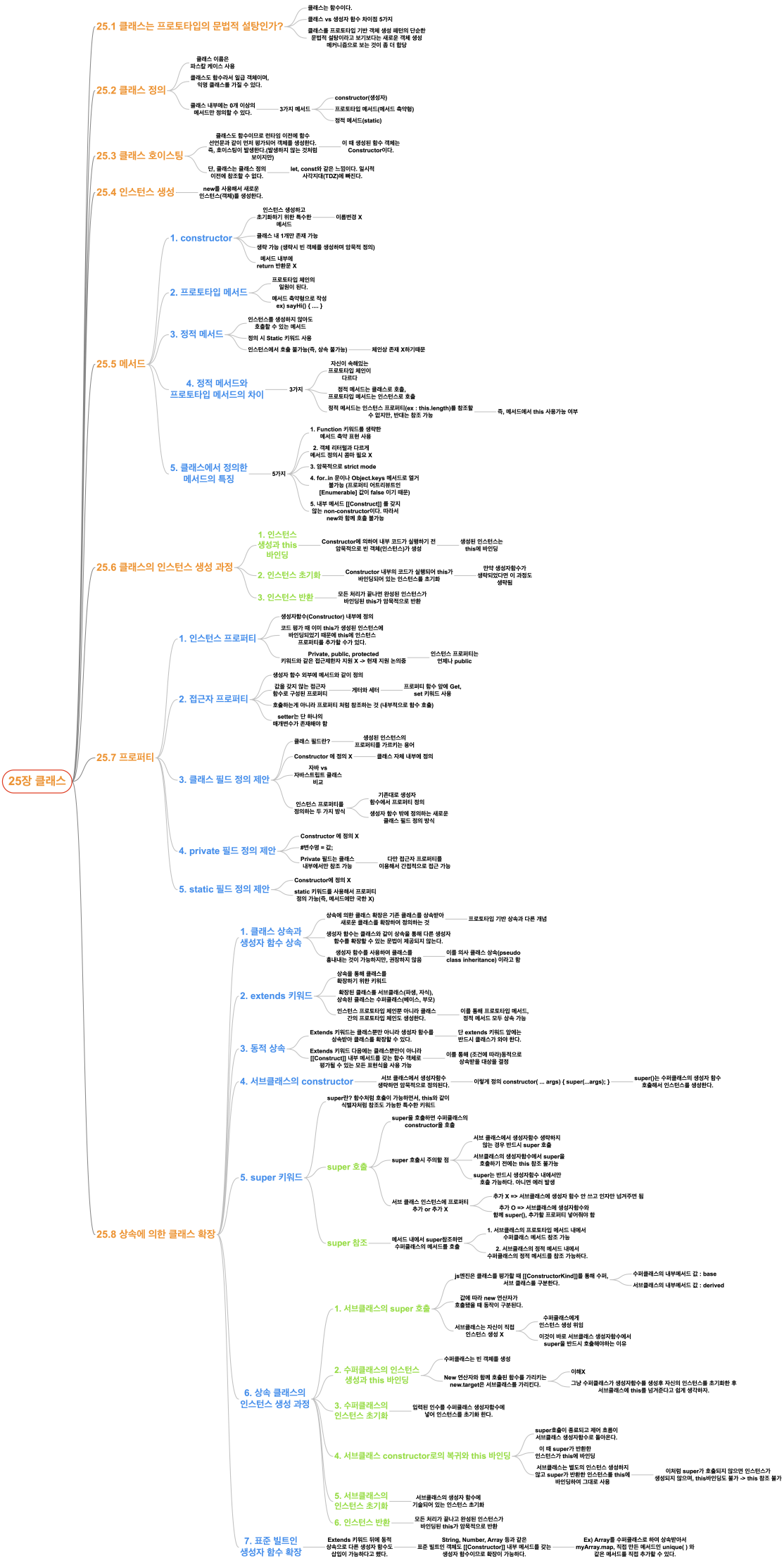
25장 클래스
25.1 클래스는 프로토타입의 문법적 설탕인가?
클래스는 함수이다.
클래스 vs 생성자 함수 차이점 5가지
클래스를 프로토타입 기반 객체 생성 패턴의 단순한 문법적 설탕이라고 보기보다는 새로운 객체 생성 메커니즘으로 보는 것이 좀 더 합당
25.2 클래스 정의
클래스 이름은 파스칼 케이스 사용
클래스도 함수라서 일급 객체이며, 익명 클래스를 가질 수 있다.
클래스 내부에는 0개 이상의 메서드만 정의할 수 있다.
3가지 메서드
constructor(생성자)
프로토타입 메서드(메서드 축약형)
정적 메서드(static)
25.3 클래스 호이스팅
클래스도 함수이므로 런타임 이전에 함수 선언문과 같이 먼저 평가되어 객체를 생성한다. 즉, 호이스팅이 발생한다.(발생하지 않는 것처럼 보이지만)
이 때 생성된 함수 객체는 Constructor이다.
단, 클래스는 클래스 정의 이전에 참조할 수 없다.
let, const와 같은 느낌이다. 일시적 사각지대(TDZ)에 빠진다.
25.4 인스턴스 생성
new를 사용해서 새로운 인스턴스(객체)를 생성한다.
25.5 메서드
1. constructor
인스턴스 생성하고 초기화하기 위한 특수한 메서드
이름변경 X
클래스 내 1개만 존재 가능
생략 가능 (생략시 빈 객체를 생성하며 암묵적 정의)
메서드 내부에 return 반환문 X
2. 프로토타입 메서드
프로토타입 체인의 일원이 된다.
메서드 축약형으로 작성 ex) sayHi() { .... }
3. 정적 메서드
인스턴스를 생성하지 않아도 호출할 수 있는 메서드
정의 시 Static 키워드 사용
인스턴스에서 호출 불가능(즉, 상속 불가능)
제안상 존재 X하기때문
4. 정적 메서드와 프로토타입 메서드의 차이
3가지
자신이 속해있는 프로토타입 체인이 다르다
정적 메서드는 클래스로 호출, 프로토타입 메서드는 인스턴스로 호출
정적 메서드는 인스턴스 프로퍼티(ex : this.length)를 참조할 수 없지만, 반대는 참조 가능
즉, 메서드에서 this 사용가능 여부
5. 클래스에서 정의한 메서드의 특징
5가지
1. Function 키워드를 생략한 메서드 축약 표현 사용
2. 객체 리터럴과 다르게 메서드 정의시 콤마 필요 X
3. 암묵적으로 strict mode
4. for..in 문이나 Object.keys 메서드로 열거 불가능 (프로퍼티 어트리뷰트인 [[Enumerable]] 값이 false 이기 때문)
5. 내부 메서드 [[Construct]] 를 갖지 않는 non-constructor이다. 따라서 new와 함께 호출 불가능
25.6 클래스의 인스턴스 생성 과정
1. 인스턴스 생성과 this 바인딩
Constructor에 의하여 내부 코드가 실행하기 전 암묵적으로 빈 객체(인스턴스)가 생성
생성된 인스턴스는 this에 바인딩
2. 인스턴스 초기화
Constructor 내부의 코드가 실행되어 this가 바인딩되어 있는 인스턴스를 초기화
만약 생성자함수가 생략되었다면 이 과정도 생략됨
3. 인스턴스 반환
모든 처리가 끝나면 완성된 인스턴스가 바인딩된 this가 암묵적으로 반환
25.7 프로퍼티
1. 인스턴스 프로퍼티
생성자함수(Constructor) 내부에 정의
코드 평가 때 이미 this가 생성된 인스턴스에 바인딩되었기 때문에 this에 인스턴스 프로퍼티를 추가할 수가 있다.
Private, public, protected 키워드와 같은 접근제한자 지원 X -> 현재 지원 논의중
인스턴스 프로퍼티는 언제나 public
2. 접근자 프로퍼티
생성자 함수 외부에 메서드와 같이 정의
값을 갖지 않는 접근자 함수로 구성된 프로퍼티
게터와 세터
프로퍼티 함수 앞에 Get, set 키워드 사용
호출하는게 아니라 프로퍼티 처럼 참조하는 것 (내부적으로 함수 호출)
setter는 단 하나의 매개변수가 존재해야 함
3. 클래스 필드 정의 제안
클래스 필드란?
생성된 인스턴스의 프로퍼티를 가르키는 용어
Constructor 에 정의 X
클래스 자체 내부에 정의
자바 vs 자바스트립트 클래스 비교
인스턴스 프로퍼티를 정의하는 두 가지 방식
기존대로 생성자 함수에서 프로퍼티 정의
생성자 함수 밖에 정의하는 새로운 클래스 필드 정의 방식
4. private 필드 정의 제안
Constructor 에 정의 X
#변수명 = 값;
Private 필드는 클래스 내부에서만 참조 가능
다만 접근자 프로퍼티를 이용해서 간접적으로 접근 가능
5. static 필드 정의 제안
Constructor에 정의 X
static 키워드를 사용해서 프로퍼티 정의 가능(즉, 메서드에만 국한 X)
25.8 상속에 의한 클래스 확장
1. 클래스 상속과 생성자 함수 상속
상속에 의한 클래스 확장은 기존 클래스를 상속받아 새로운 클래스를 확장하여 정의하는 것
프로토타입 기반 상속과 다른 개념
생성자 함수는 클래스와 같이 상속을 통해 다른 생성자 함수를 확장할 수 있는 문법이 제공되지 않는다.
생성자 함수를 사용하여 클래스를 흉내내는 것이 가능하지만, 권장하지 않음
이를 의사 클래스 상속(pseudo class inheritance) 이라고 함
2. extends 키워드
상속을 통해 클래스를 확장하기 위한 키워드
확장된 클래스를 서브클래스(파생, 자식), 상속된 클래스는 수퍼클래스(베이스, 부모)
인스턴스 프로토타입 체인뿐 아니라 클래스 간의 프로토타입 체인도 생성한다.
이를 통해 프로토타입 메서드, 정적 메서드 모두 상속 가능
3. 동적 상속
Extends 키워드는 클래스뿐만 아니라 생성자 함수를 상속받아 클래스를 확장할 수 있다.
단 extends 키워드 앞에는 반드시 클래스가 와야 한다.
Extends 키워드 다음에는 클래스뿐만이 아니라 [[Construct]] 내부 메서드를 갖는 함수 객체로 평가될 수 있는 모든 표현식을 사용 가능
이를 통해 (조건에 따라)동적으로 상속받을 대상을 결정
4. 서브클래스의 constructor
서브 클래스에서 생성자함수 생략하면 암묵적으로 정의된다.
이렇게 정의 constructor(... args) { super(...args); }
super()는 수퍼클래스의 생성자 함수 호출해서 인스턴스를 생성한다.
5. super 키워드
super란? 함수처럼 호출이 가능하면서, this와 같이 식별자처럼 참조도 가능한 특수한 키워드
super 호출
super을 호출하면 수퍼클래스의 constructor을 호출
super 호출시 주의할 점
서브 클래스에서 생성자함수 생략하지 않는 경우 반드시 super 호출
서브클래스의 생성자함수에서 super을 호출하기 전에는 this 참조 불가능
super는 반드시 생성자함수 내에서만 호출 가능하다. 아니면 에러 발생
서브 클래스 인스턴스에 프로퍼티 추가 or 추가 X
추가 X => 서브클래스에 생성자 함수 안 쓰고 인자만 넘겨주면 됨
추가 O => 서브클래스에 생성자함수와 함께 super(), 추가할 프로퍼티 넣어줘야 함
super 참조
메서드 내에서 super참조하면 수퍼클래스의 메서드를 호출
1. 서브클래스의 프로토타입 메서드 내에서 수퍼클래스 메서드 참조 가능
2. 서브클래스의 정적 메서드 내에서 수퍼클래스의 정적 메서드를 참조 가능하다.
6. 상속 클래스의 인스턴스 생성 과정
1. 서브클래스의 super 호출
js엔진은 클래스를 평가할 때 [[ConstructorKind]]을 통해 수퍼, 서브 클래스를 구분한다.
수퍼클래스의 내부메서드 값 : base
서브클래스의 내부메서드 값 : derived
값에 따라 new 연산자가 호출됐을 때 동작이 구분된다.
서브클래스는 자신이 직접 인스턴스 생성 X
수퍼클래스에게 인스턴스 생성 위임
이것이 바로 서브클래스 생성자함수에서 super을 반드시 호출해야하는 이유
2. 수퍼클래스의 인스턴스 생성과 this 바인딩
수퍼클래스는 빈 객체를 생성
New 연산자와 함께 호출된 함수를 가리키는 new.target은 서브클래스를 가리킨다.
이해X
그냥 수퍼클래스가 생성자함수를 생성후 자신의 인스턴스를 초기화한 후 서브클래스에 this를 넘겨준다고 쉽게 생각하자.
3. 수퍼클래스의 인스턴스 초기화
입력된 인수를 수퍼클래스 생성자함수에 넘어 인스턴스를 초기화 한다.
4. 서브클래스 constructor로의 복귀와 this 바인딩
super호출이 종료되고 제어 흐름이 서브클래스 생성자함수로 돌아온다.
이 때 super가 반환한 인스턴스가 this에 바인딩
서브클래스는 별도의 인스턴스 생성하지 않고 super가 반환한 인스턴스를 this에 바인딩하여 그대로 사용
이처럼 super가 호출되지 않으면 인스턴스가 생성되지 않으며, this바인딩도 불가 -> this 참조 불가
5. 서브클래스의 인스턴스 초기화
서브클래스의 생성자 함수에 기술되어 있는 인스턴스 초기화
6. 인스턴스 반환
모든 처리가 끝나고 완성된 인스턴스가 바인딩된 this가 암묵적으로 반환
7. 표준 빌트인 생성자 함수 확장
Extends 키워드 뒤에 동적 상속으로 다른 생성자 함수도 삽입이 가능하다고 봤다.
String, Number, Array 등과 같은 표준 빌트인 객체도 [[Construct]] 내부 메서드를 갖는 생성자 함수이므로 확장이 가능하다.
Ex) Array를 수퍼클래스로 하여 상속받아서 myArray.map, 직접 만든 메서드인 unique( ) 와 같은 메서드를 직접 추가할 수 있다.