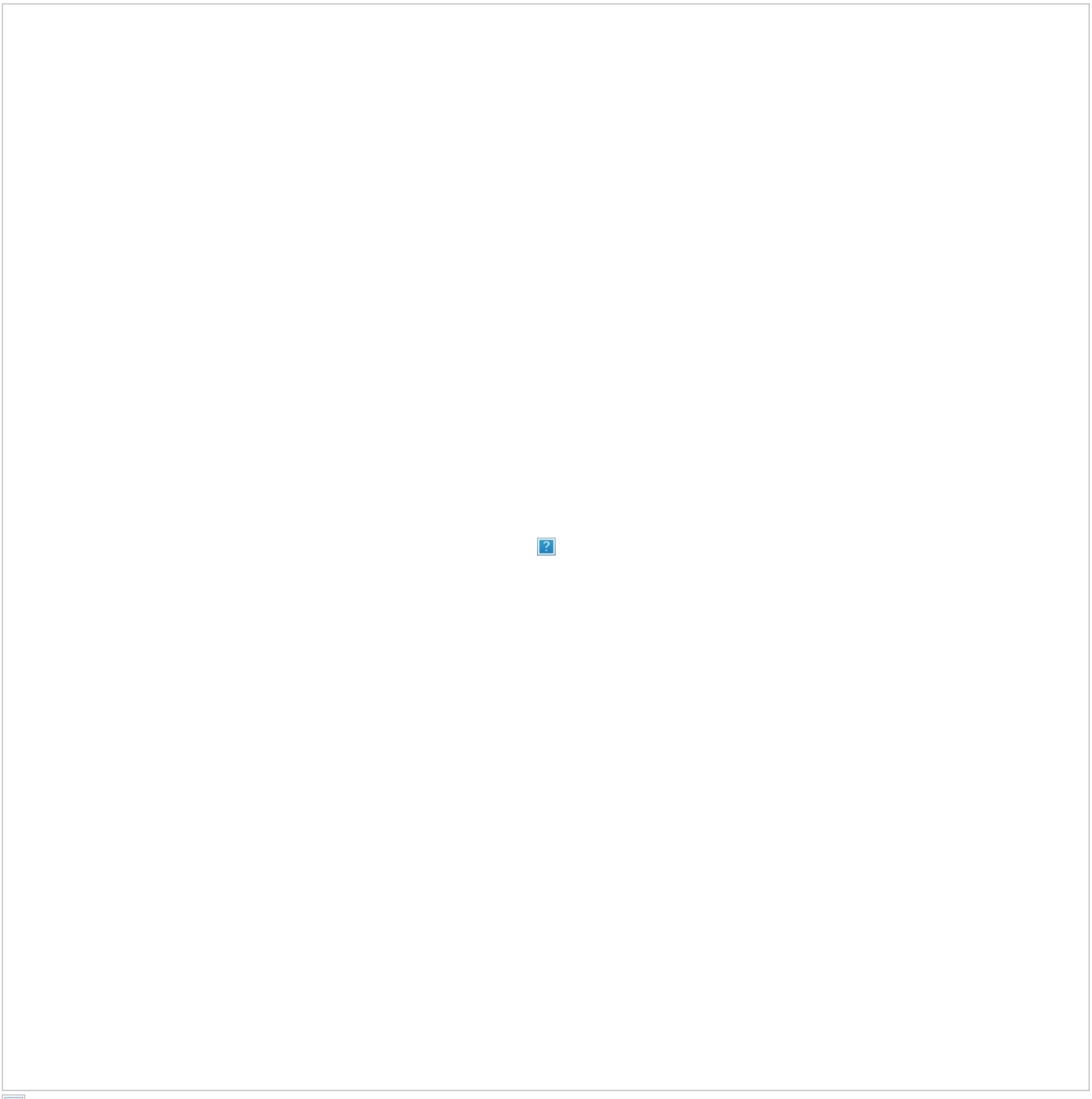
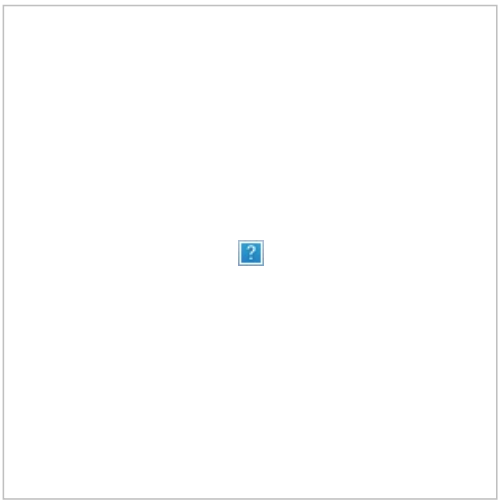
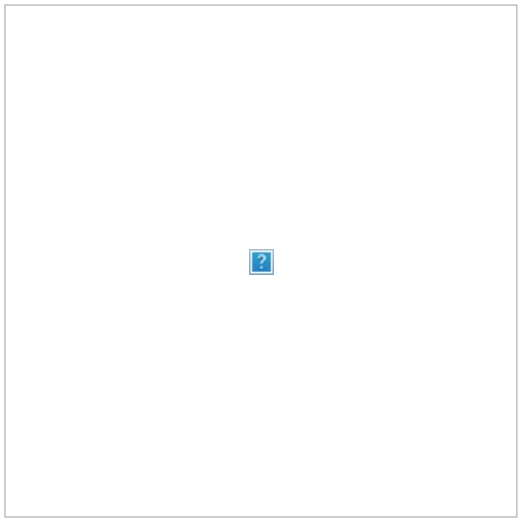
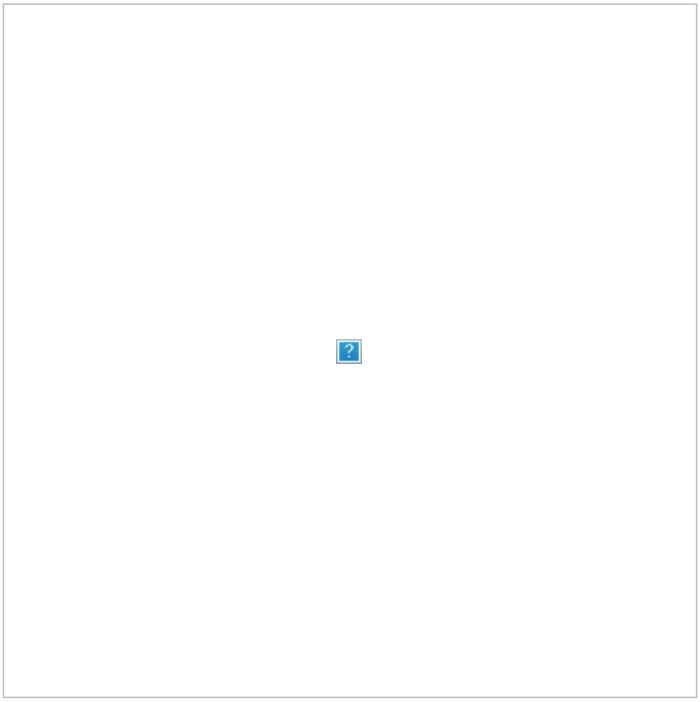
0314 수정

#3월#Java#예외처리
오늘 배운 것들의 목록이다
정적바인딩
- 코드가 컴파일 되기 전 상태가 보인다 .
동적바인딩
-코드가 실행 되면서 각 자식 클래스에서 Override된 값이 출력 된다.



정적 바인딩 상태임으로 오버라이딩 안된 기본적으로 선언된 Car의 위치만 나온다.
하지만 실행 시

각각의 Car / Tesla / Spark 에 Override 각각의 toString이 출력 된다.

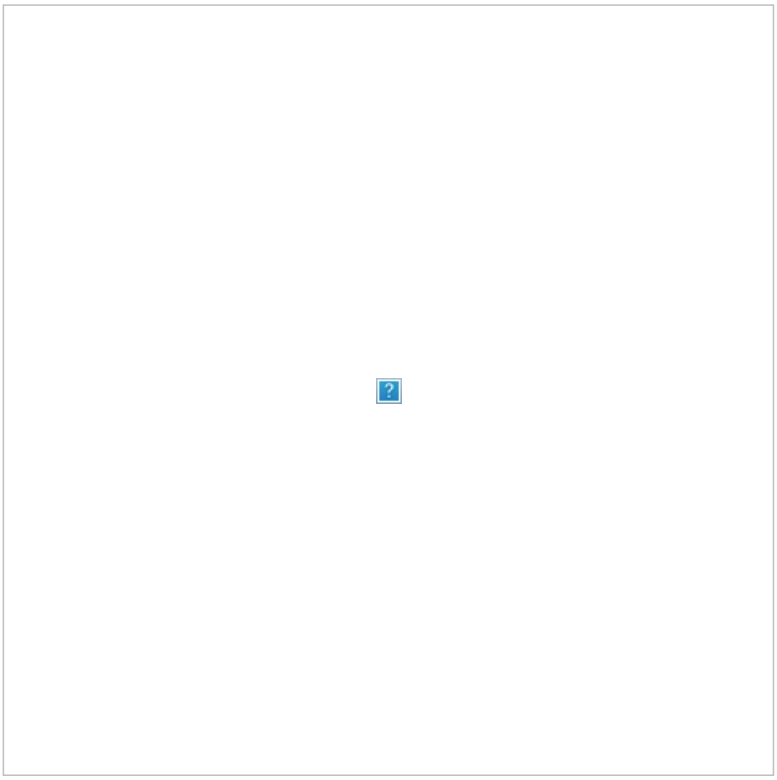
❖상속 관계로 이루어져 다형성이 적용된 경우 메소드 오버라이딩이 되어있으면

정적으로 바인딩 된 메소드 코드 보다 오버라이딩된 메소드 코드를 우선적으로 실행 한다.

---

instanceof 연산자

- 현재 참조형 변수가 어떤 클래스 형의 객체 주소를 참조하고 있는지 확인 할 때 사용 클래스 타입이 맞으면 **true** 아니면 **false**반환.



추상 클래스 (extends)

- 대략적으로 구성된 메소드를 포함한 클래스
- **abstract** 예약어를 사용해야 한다.
    - [접근제한자] **abstract class** 클래스명 **{}**

추상 메소드

- 구체적인 값이 없는 메소드 상속 시 반드시 구현되어야 한다.
- 추상 메소드의 선언부에 **abstract** 예약어를 사용해야 한다.
    - **[접근제한자] abstract** 반환형 메소드명 **(**자료형 변수명**);** 구체화되지 않으니 **{}** 대괄호를 쓰지 않는다.

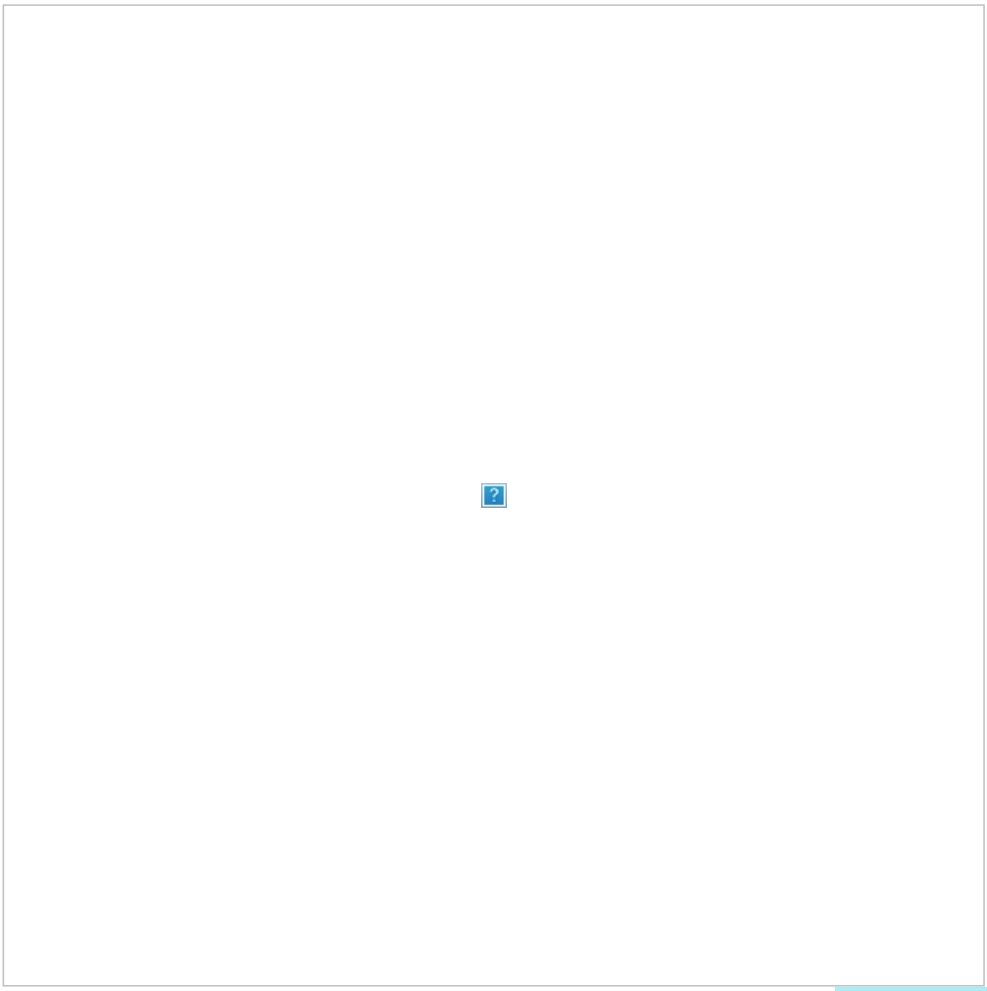
추상 클래스의 특징

1. 미완성 클래스다 **(abstract** 키워드 사용**):** 자체적으로 객체 생성 불가 **- >** 반드시 상속하여 상속해야 한다
2. **abstract** 메소드가 포함된 클래스는 반드시 **abstract** 클래스이다 **(abstract)** 가 없어도
3. **abstract** 클래스여도 필드, 일반 생성자,메소드를 포함해도 괜찮다.
4. 스스로 객체 생성은 안되지만 공통된 부분이 있다면 자식 객체를 통해 참조형 변수로 사용 가능하다

장점

- 상속 받은 자식에게 공통된 멤버 제공
- 일부 기능의 구현을 강제화 (공통적으로 반드시 구현되어야 되지만 자식마다 재정의 방식이 다른 경우)

인터페이스 : implements



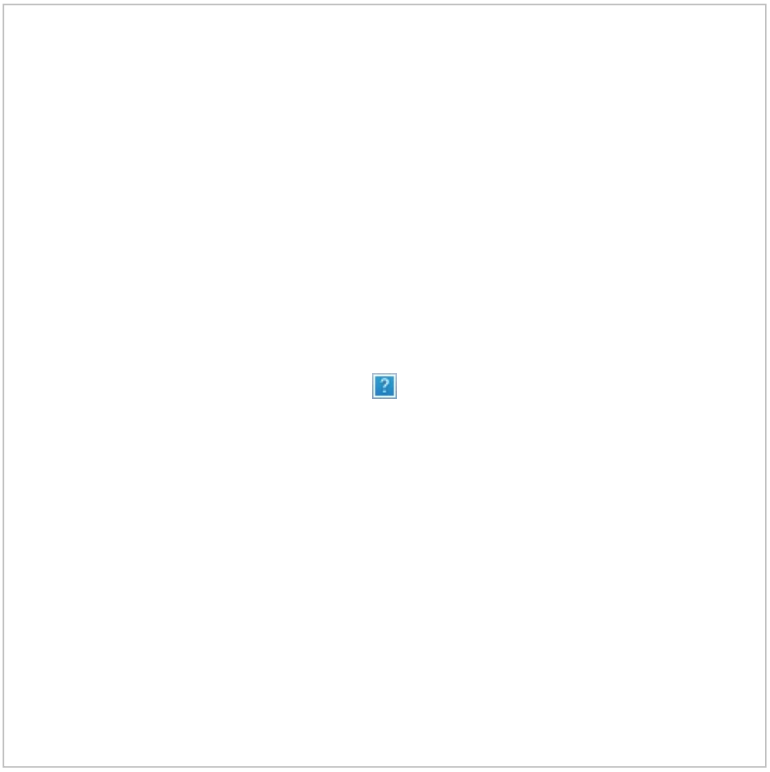
- 추상 클래스가 추상 메소드를 0개 이상만 포함 하면 되지만 인터페이스는 모든 클래스가 abstract이여야 한다.
- 상수형 필드와 추상 메소드만이 작성 가능하다

특징

1. 모든 메소드는 묵시적으로 ***public abstract***
2. 변수 생성은 묵시적으로 ***public static fianl***
3. 객체 생성은 안되나 참조형 변수로 사용 가능하다(다형성)
    1. 부모 참조 변수로 자식 참조 변수를 를 참조하는 것.

장점

- 다형성을 이용해서 상위 타입 역할(자식 객체 간 연관성이 없어도 연결 시킴)
- 인터페이스 구현 객체 공통된 기능 구현 강제화 한다
- 공동 작업을 위한 인터페이스 제공한다



언제 어떤 상속 기술을 사용하는가
상속 : 재사용 + 확장
추상 클래스 : abstract 외에 동적 필드 , 기본 생성자를 활용할 수 있다
인터페이스 : 무조건 고정되어 있어야 한다 (상수 필드 , static 메소드)
추상 클래스는 단일 상속만 되지만
인터페이스는 다중 상속이 지원되는 점이 큰 차이점이라는 거 같다
implements : 키워드 느낌으로 인터페이스에 정의된 정의된 메소드를 각 클래스에 목적에 맞게 기능을 구현하는 느낌이다
abstract : 자신의 기능을 하위 클래스로 확장 하는 느낌이다
예외 / 에러 /예외 처리
예외의 종류

- 컴파일 에러
  - 프로그램의 실행을 막는 소스 코드 상의 문법 에러 (소스 코드 수정으로 해결 가능)
- 런타임 에러
  - 프로그램을 실행하는 중 발생 하는 에러 (주로 if문 사용 등으로 배열을 벗어나거나 할 경우 계산식의 오류)
- 시스템 에러
  - 컴퓨터의 물리적인 오작동으로 인한 에러 (소스 코드만으로는 해결 불가(

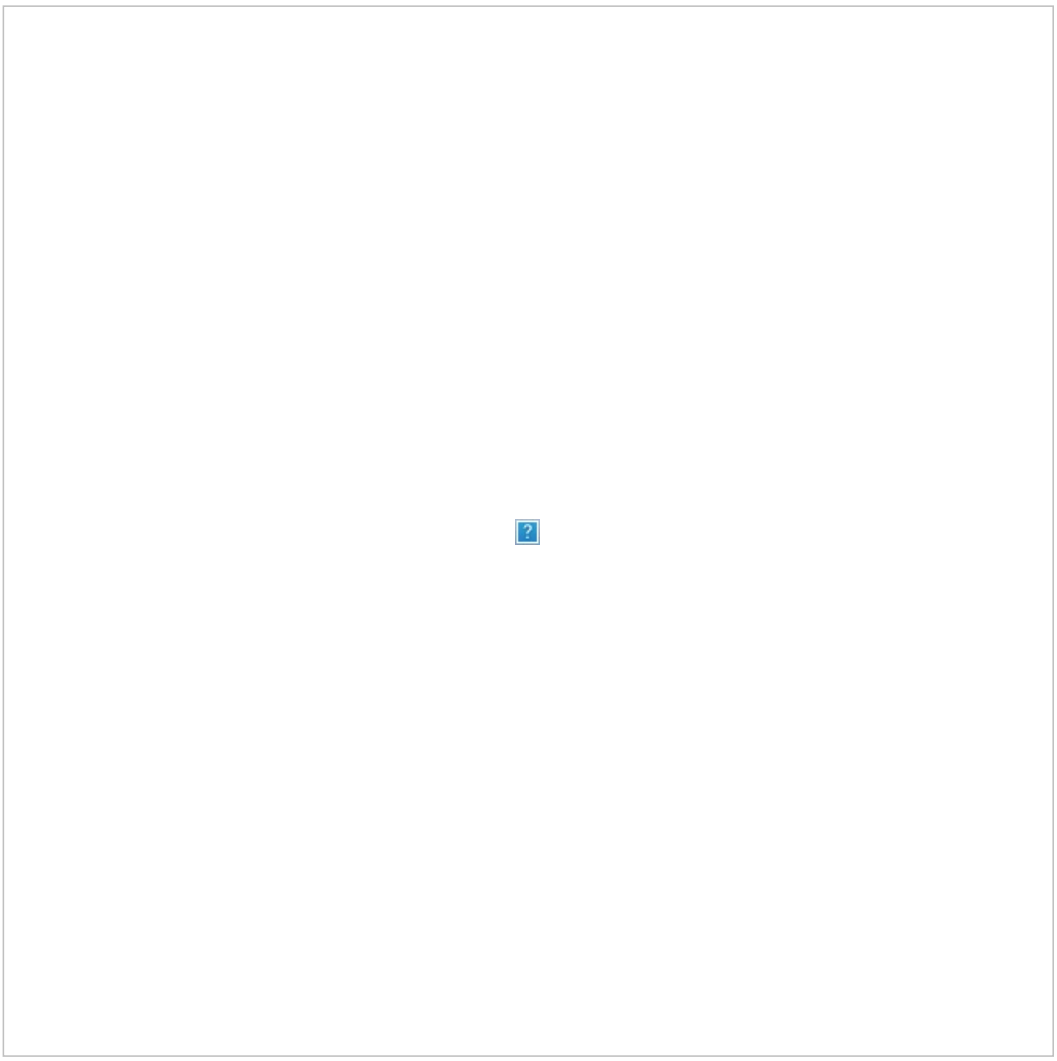
소스 코드 수정으로 해결 가능한 에러를 예외(Exception)이라 한다

이러한 예외(Exception)상황을 (예측 가능한 에러) 구문을 처리하는 방법인 예외 처리를 통해 해결

API문서를 조회에서 해당 클래스에 대한 생성자나 메소드를 검색하면
그 메소드가 어떤 Exception을 발생 시킬 가능성이 있는지 확인 가능하다.

- 발생하는 예외를 미리 확인하여 상황에 따른 예외 처리 코드를 작성할 수 있음

예외클래스 계층 구조



Unchecked Excception 은 주로 프로그래머의 부주의로 인한 오류인 경우가 많기 때문에

예외처리 보다는 코드를 수정해야 하는 경우가 많음

ArithmeticException	0으로 나누는 경우 발생 if문으로 나누는 수가 0인지 검사

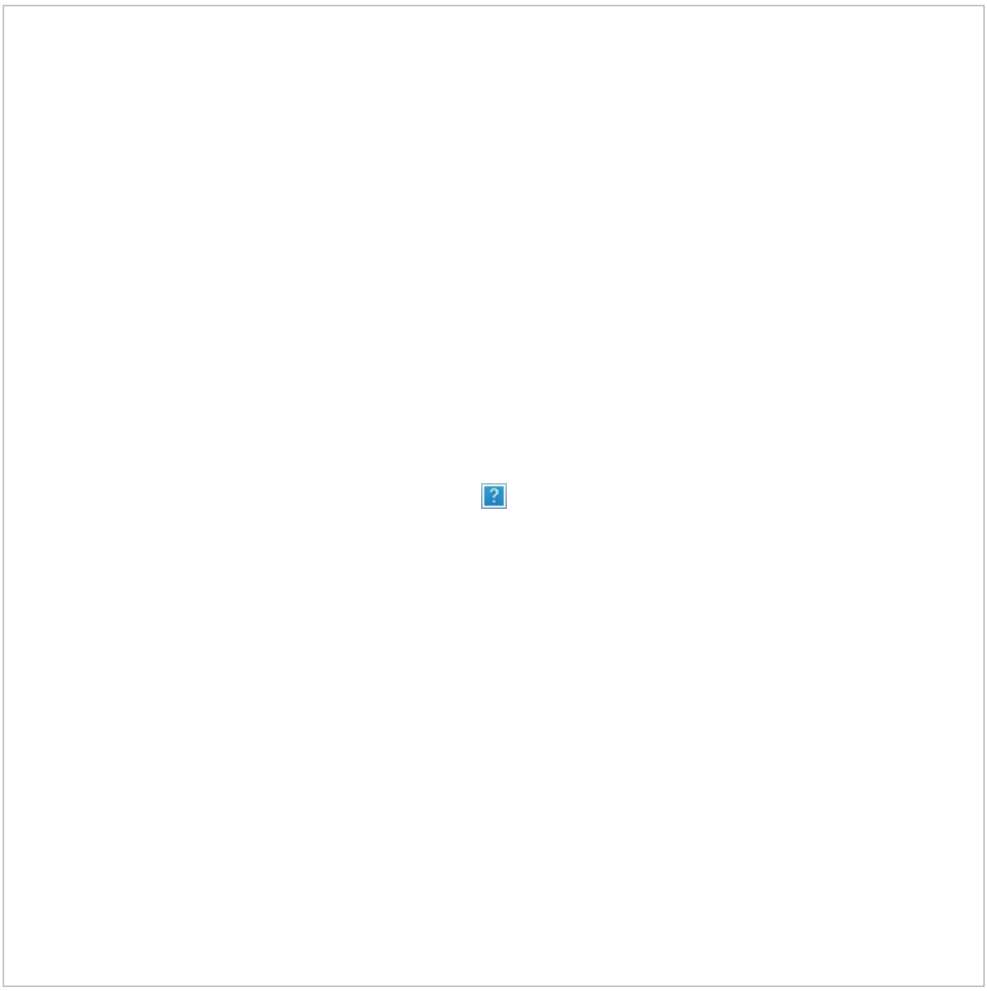
NullPointerException Null인 참조 변수로 객체 멤버 참조 시도 시 발생 객체 사용 전에 참조 변수가 null인지 확인

NegativeArraySizeException 배열 크기를 음수로 지정한 경우 발생 배열 크기를 0보다 크게 지정해야 함

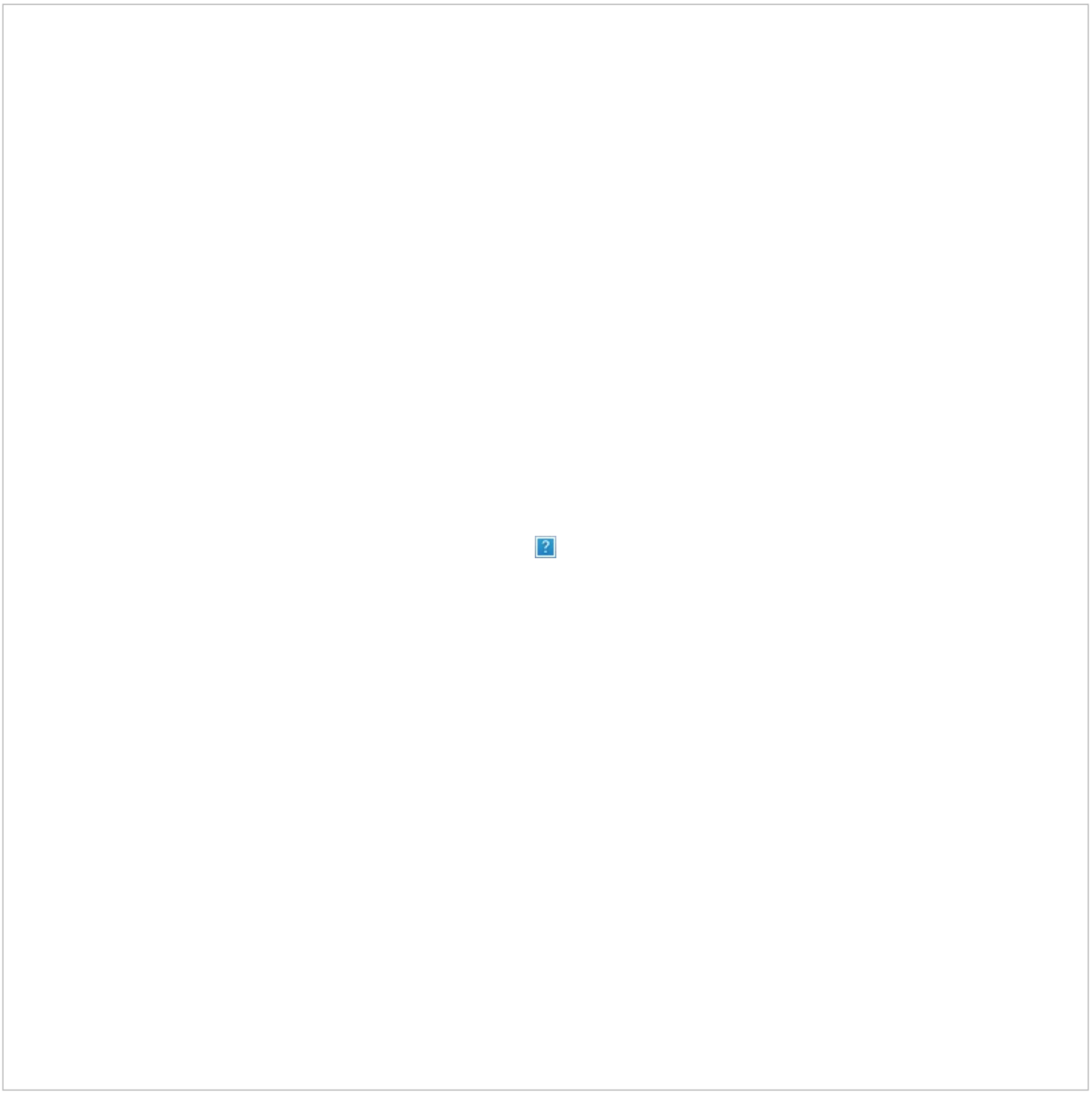
ArrayIndexOutOfBoundsException 배열의 index범위를 넘어서 참조하는 경우 배열명.length를 사용하여 배열의 범위 확인

ClassCastException Cast 연산자 사용 시 타입 오류 instanceof 연산자로 객체타입 확인 후 cast연산

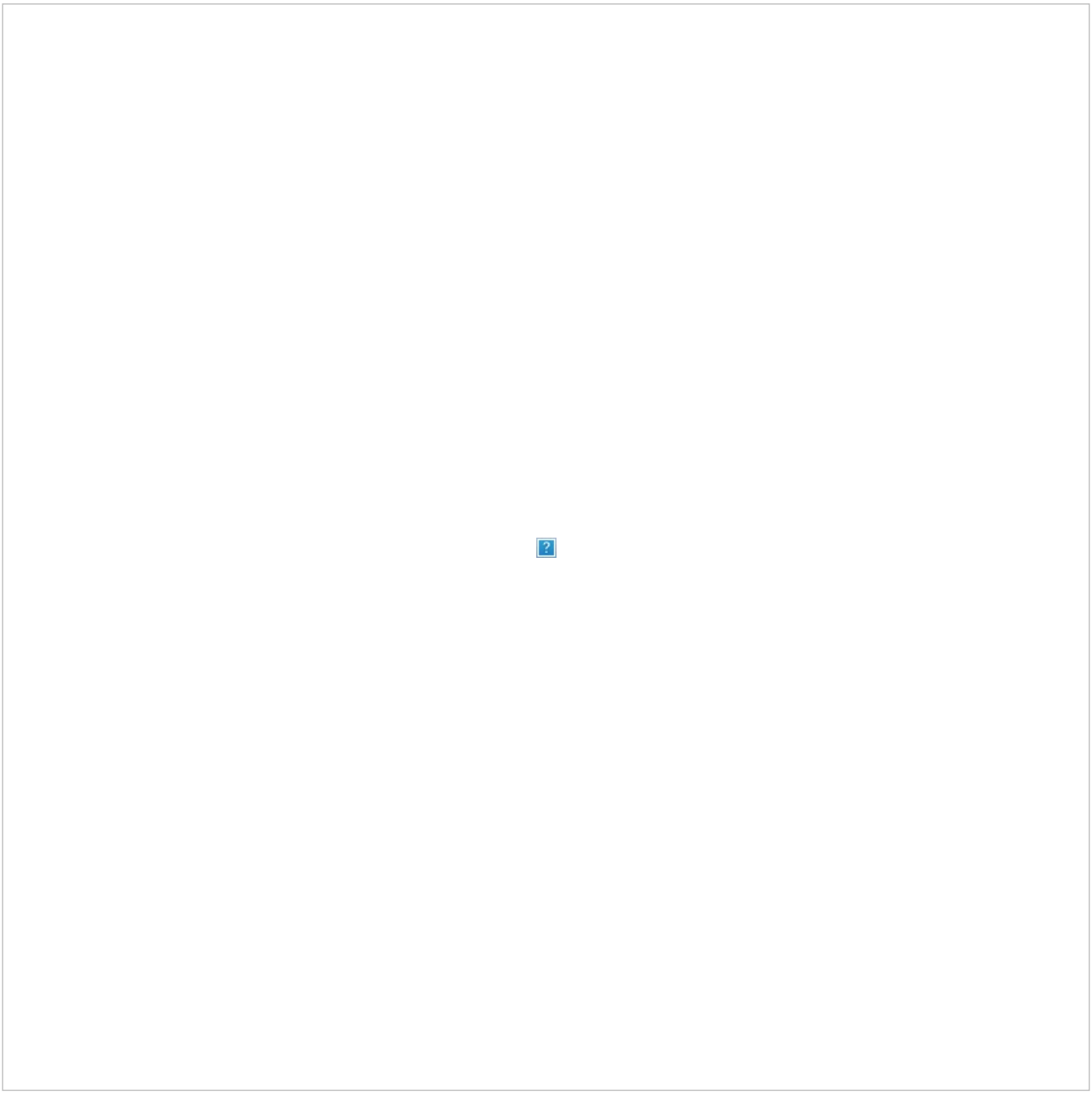
InputMismatchException Scanner를 사용하여 데이터 입력 시 입력 받는 자료형이 불일치할 경우 발생



try ~ catch



오버라이딩 시 **throws**하는 **Exception**의 개수와 상관없이 처리 범위가 같거나 후손이여야 한다



+다형성 / 입력 버퍼 예외처리