```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Classes</title>
</head>

<body>

    <script>
        /*
,            ***** Classes ****
,
,            Classes는 화려한 object이다.
,            대부분 나는 Classes를 스스로 만들어 쓰지는 않을것이다.
,
,            Class는 많은 사람들이 라이브러리나 리액트 같은것을 만들 때
,            Classes를 export해서 사용한다.
,
,            예를들어, 내가 엄청 많은 코드를 가지고 있고 이것을 구조화하기를 원할때,
,            Class를 이용한다면 매우 유용할수있다. 재사용이 가능하기 때문
,
,            Class는 기본적으로 blueprint(청사진)이다.
,            blueprint는 설계도같은것. 비행기를 가정하고 비행기는 아니지만
,            비행기가 어떻게 생겼는지를 알려주는것.
,
,            비행기를 예를들면, 이런것들을 실제로 운행되지는 않지만
,            어떻게 만들어야하는지 모든 specification을 가지고있다.
,
,            * Class는 constructor(생성자)를 안에 가지고 있다.
,            Constructor는 Class를 말 그대로 construct(구성)한다는것.
,            Class안에는 뭔가 이상한 this로 불리는것이 있다.
,
,            class User {
,                constructor(){
,                    this.username = 'bibi';
,                }
,            }
,
,            console.log(User.username)
,            이렇게 찍어보면 undefined가 뜬다. 왜그럴까.
,            constructor안에 단지 두기만했는데.
,            차이점은 현재 만든것은 그냥 blueprint이다.
,
,            내가해주어야할것은 이 blueprint로 Class를 생성해주어야한다.
,            그래서 이 Class를 가져다가 선언을하는것이다. 이 blueprint로 클래스를 만들도록.
,
,            class sexyUser = new User();
,            sexyUser는 이제 이 Class의 instance다.
,            instance는 살아있는 Class를 의미한다.
,            class User부분은 죽어있는 blueprint이고 sexyUser가 살아있는 Class이다.
,
,            console.log(sexyUser.username)이런식으로 쓸 수 있는것.
,
,
,            * 또 우리는 Class에 함수도 만들 수 있다.
,
,            얼마나 많은 instance들을 class가 가질수있을까. 원하는만큼 가능.
```

```
,
,
,            const baseObject = {
,                username : "bibimon",
,                sayHello : function(){
,                    console.log("bibimon hello")
,                }
,            }
,            const sexyUser = baseObject;
,            const uglyUser = baseObject;
,
,            sexyUser.sayHello();
,            uglyUser.sayHello();
,
,            이런 방식으로도 가능하다. 여기 왜 Class가 필요하냐에 대한 의문을 가지게된다.
,
,            차이점은
,            * 이미 object를 하나 만들어버리는것에 있다.
,            그리고 그것을 단지 sexy,ugly User에 넣은것뿐이다.
,            Class로 햇을때는 실제로 만들어내지않았다.
,
,            class의 constructor에 argument를 주고, 그리고 이것은 name이 들어갈
,            첫번째 argument이다. 그리고 construction을 통해서 name을 정해주어야한다.
,
,            class User {
,                constructor(name){
,                    this.username = name;
,                }
,                sayHello(){
,                    console.log("hello i'm bibi")
,                }
,            }
,            const sexyUser = new User("bibi");
,            console.log(sexyUser.username)
,            이것은 object로는 하지못한다. object는 단지 object일 뿐
,
,            ** Class는 기본적으로 object의 공장이다.
,            우리가 baseObject로 만든것은 심플하고, 정의를 해줘야했고, 특징도 없고,
,            뭐든 넣어도 되는 username같은 변수적인 특징도 없었다.
,            sayHello(){
,                console.log(`Hello, my name is ${this.username}`)
,            }
,            여기서의 this가 동작하는 이유는 나중에 설명.
,
,            많은 Classes를 가지면 내 코드는 아마 엄청 구조적으로 변할것이다.
,            counstructor를 가지고 또 함수를 가지면 엄청 구조적으로 될것.
,
,            sayHello()를 그냥 찍어보면 존재하지않는다. scope의 유효범위가 다르기때문
,            sayHello는 오직 Class의 *instance* 안에 존재하는것이다.
,         */

        class User {
            constructor(name){
                this.username = name;
            }
            sayHello(){
                console.log(`Hello, my name is ${this.username}`)
            }
        }
        const sexyUser = new User("bibi");
        const uglyUser = new User("bucky");
```

```
123
124         console.log(sexyUser.username)
125         setTimeout(sexyUser.sayHello, 4000)
126
127         console.log(uglyUser.username)
128         setTimeout(uglyUser.sayHello, 4000)
129     </script>
130 </body>
131
132 </html>
```