```
<!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>proxy</title>
</head>
<body>
  <script>
    /*
,
,       ***** proxy *****
,       proxy는 filter처럼 생각할 수 있다.
,       그래서 이번엔 proxy를 filter로 선언해보겠다.
,
,       proxy는 두개의 argument를 가져오는데 하나는 target, 하나는 handler이다.
,
,       const userObj = {
,         username : "nico",
,         age : 12,
,         password : 1234
,       }
,       const userFilter = {};
,       const filteredUser = new Proxy(userObj, userFilter);
,
,       filteredUser는 보다시피 userObject의 모든것을 Reflection한다.
,       filteredUser.age / filteredUser.password 하면 값이 나옴.
,       따라서 기본적으로 내가 filteredUser를 호출하면 userObj를 얻음.
,
,       이론적으로 실제로 내가 filteredUser를 호출하면 첫번째로 userFilter가 호출되고,
,       그 다음에 userObject의 value가 return되거나 안될수도있다.
,
,       현재 userObj에 무언가 있다.
,       이것들 중 하나는 예를들어 get 이라고 부른다. ( 콘솔에 filteredUser를 쳐보면 쓸수있는 메소드들
이 나옴)
,       get은 단순히 properties를 얻을때 사용한다.
,       이것들을 나는 verb라고 부르는데 proxy world에서는 trap이라고 부른다.
,
,       mdn을 보면 많은 trap들이 있다. has, set, get ...
,       따라서 몇개의 trap들을 선언해보겠다
,
,       * trap
,       trap들은 proxy의 부분으로 handler에서 생성된다.
,
,       const userFilter = {
,         get : () => {
,           console.log("somebody is getting something")
,         },
,         set : () => {
,           console.log("Somebody wrote something")
,         }
,       };
,       이런식으로 만들어보고 filteredUser.password를 찍어보면
,       get 부분의 콘솔이 찍힐것이다. 무슨일이 벌어진것일까.
,
,       이 proxy는 Trap을 호출하고있다. 지금같은 경우에는 get
,       filteredUser.active = true;
,       만약 내가 하나를 이런식으로 만든다면 set쪽의 구문이 실행될것이다.
,       set으로 설정을 해준다고해서 실제 userObj에 생기는것은 아니다.
,
```

```
,       * 흥미로운점
,       우리는 filteredUser를 생성하고
,       proxy에게 targetObj를 주고 그 다음에 userFilter를 사용한다.
,       userFilter는 몇가지 properties를 가진다. get 그리고 set
,       이것들을 trap이라고 부르는데 우리가 기본적으로 여기서 하는것들은 userObj의 이벤트를 가로챈다.
,       userObj에 접근하도록 두지 않는다.
,       따라서 무엇을 하든 filteredUser의 password를 얻지 못할것이다
,
,       기본적으로 get 은 어떤 value도 주지 않음.
,       get 구문안에 return을 넣어보면
,       내가 어떤 것을 하던지 return쪽 구문이 리턴이 된다.
,       userFiltered는 return값을 바꿔서 전달한다.
,
,       이것이 proxy이다. object앞에 있는 filter
,
,       * 정리
,       콘솔에 filteredUser. 을 쳐보면 username, age, password에 대한 제안을 얻는다
,       기본적으로 filteredUser는 userObj이고, object이기 때문.
,       하지만 모든 동작은 가로채진다 userFilter에 의해서.
,
,       ***** handler trap *****
,
,       - get
,
,       get은 property value를 취하는것에 대한 trap이다.
,       get은 3개의 argument를 받는다. target,prop,receiver
,       이름은 임의로 지정가능. 각 argument를 콘솔에 찍어보면 알 수 있다.
,       target = target object를 출력해줌
,       prop = user가 물어본 prop을 반환해준다.
,       receiver = receiver는 Proxy가 될것이다.
,
,       * 순조롭게 진행하기 위해서
,       기본적으로 target을 취하고 []그 안에 property를 넣는다.
,       target[age]
,       이것이 누군가가 만약 age를 호출하면 target을 반환하는 이유다.
,
,       * 모든것을 가져올 수 있지만 원본 객체를 바꾸고 있지 않는다.
,       return prop === "password" ? `${"*".repeat(5)}` : target[prop];
,       이런식으로 누군가 password를 물어본다면 *****만 출력시켜줄수 있을것이다.
,       만약 password를 물어보지 않으면 user가 원하는것을 반환.
,
,       - delete
,
,       deleteProperty : (target, prop) => {
,         if ( prop === "password") {
,           return ;
,         } else {
,           target[prop] = "DELETED"
,         }
,       }
,
,       이런식으로 지우려고할때에 target[prop]를 덮어쓰는 코드로 만들어보았다.
,       delete filteredUser.username을 하면
,       username에 DELETED가 들어간 것을 볼 수 있다.
,
,       **** 정리
,
,
,
,
,     */
```

```
122
123     const userObj = {
124       username : "nico",
125       age : 12,
126       password : 1234
127     }
128
129     const userFilter = {
130       // 이름들은 내가 정하는것이아님.
131       get : (target, prop, receiver) => {
132         return prop === "password" ? `${"*".repeat(5)}` : target[prop];
133       },
134       set : () => {
135         console.log("Somebody wrote something")
136       },
137       deleteProperty : (target, prop) => {
138         if ( prop === "password") {
139           return ;
140         } else {
141           target[prop] = "DELETED"
142         }
143       }
144     };
145
146     const filteredUser = new Proxy(userObj, userFilter);
147
148   </script>
149 </body>
150 </html>
```