(슬라이드 1) 안녕하십니까 오늘 동기, 비동기 그리고 블로킹, 논블로킹에 대해 발표하게 된 소프트웨어학과 22학번 이이슬입니다.

(슬라이드 2) 먼저 동기, 비동기에 대해 설명하겠습니다.  
동기, synchronous 단어를 보자면 syn은 그리스어로 함께, chrono는 시간이라는 뜻인데요.  
즉 Synchronous는 작업 시간을 함께 맞춰서 실행한다 라는 뜻으로 해석됩니다.  
작업을 맞춰 실행한다는 말은 요청한 작업에 대해 완료 여부를 따져 순차대로 처리하는 것을 말합니다.

다음으로 비동기, Asynchronous는 synchronous앞에 A가 붙어 부정하는 형태인데요.  
그래서 동기와 반대로, 요청한 작업에 대해 완료 여부를 따지지 않고 자신의 다음 작업을 그대로 수행하게 됩니다.

(슬라이드 3) 이해를 돕기 위해 예시를 준비했는데요. 먼저 동기방식의 예로 A가 B에게 만원을 송금하는 경우를 들었습니다. 첫번째로 A의 계좌는 만원을 뺄 생각을 하고 있고,   
다음으로 A의 계좌가 B의 계좌에 만원을 송금합니다.   
B의 계좌는 만원을 받았다는 걸 인지하고, A의 계좌에 만원을 받았다고 전송한 후   
A, B 계좌 각각 증가와 차감이 동시에 발생하게 됩니다!!   
아까 설명한 개념과 동시에 생각해 보자면 2번 3번과 같이 요청과 응답으로 작업 완료 여부를 따진 후, 4번과 같이 같은 일이 동시에 진행된 것을 볼 수 있습니다.

다음으로 비동기 방식의 예로는 시험날의 학생과 선생님 경우를 들었는데요.   
학생은 시험문제를 풀고 시험문제를 모두 푼 학생은 선생님께 시험지를 제출하겠죠?   
그 후 선생님은 학생의 시험지를 채점한 뒤 채점된 시험지를 전달하고, 학생은 전달받은 시험지 채점 결과를 확인하게 됩니다.  
학생이 시험지를 제출한 뒤 결과가 나오기 전에 다음 과목 시험을 보거나 활동을 하겠죠? 이는 요청한 작업 완료 여부를 따지지 않고 자신의 다음 작업을 수행하는 것을 알 수 있고, 학생과 선생님은 시험지에 행하는 행위가 다르기 때문에 둘의 작업 처리 시간은 일치하지 않고, 일치하지 않아도 된다는 것을 알 수 있습니다.

(슬라이드 4) 좀 전에 설명한 비동기의 예제를 통해 블록과 논블록의 차이를 간략하게 설명하면, 학생이 시험지를 선생님께 제출한 후 돌려받기 전까지의 행동을 통해 구분할 수 있는데   
제출 후 가만히 앉아 채점이 끝나 시험지를 돌려 받기만을 기다린다면 학생은 블록 상태이고,   
제출 후 채점 완료 응답을 받기 전까지 다른 과목 공부나 게임 등 다른 일을 하게 되면 학생의 상태는 논 블록 상태입니다.

(슬라이드 5) 다음으로 블로킹과 논 블로킹인데요. 이전에 동기와 비동기는 작업 완료 여부의 차이였다면, 블로킹과 논 블로킹은 제어권을 어떻게 처리하는가에 따라 달라지는데요.   
먼저 제어권은 함수의 코드를 실행할 권리 같은 것인데요. 제어권을 가진 함수는 자신의 코드를 끝까지 실행한 후, 자신을 호출한 함수 에게 돌려줍니다.  
그리고 블로킹은 실행중인 함수가 다른 함수를 호출하면, 제어권을 호출한 함수 에게 넘겨주는 반면, 논 블로킹은 실행중인 함수가 다른 함수를 호출해도 제어권은 그대로 자신이 가집니다. 여기에 대해 좀 더 자세히 설명 드리겠습니다.

(슬라이드 6) 먼저 블로킹부터 보자면 A함수가 B함수를 호출하면 B에게 제어권을 넘깁니다.  
제어권을 넘겨받은 B는 열심히 함수를 실행하고, A는 B에게 제어권을 넘겨주었기 때문에 함수 실행을 잠시 멈춥니다. 이제 B함수의 실행이 끝나면 자신을 호출한 A에게 제어권을 돌려주게 됩니다.

다음으로 논 블로킹을 보면 A함수가 B함수를 호출하면, B함수는 실행되지만, 제어권은 A함수가 그대로 가지고 있습니다. A함수는 계속 제어권을 가지고 있기 때문에 B함수를 호출한 이후에도 자신의 코드를 계속 실행하게 됩니다.

(슬라이드 7) 다음으로 동기/비동기와 블로킹/논 블로킹 조합을 보겠습니다. 조합은 이와 같이 sync blocking, sync non-blocking, Async blocking, Async non-blocking이 있습니다.

(슬라이드 8) 먼저 sync blocking은 다른 작업이 진행되는 동안 자신의 작업을 처리하지 않고, 다른 작업의 완료 여부를 바로 받아 순차적으로 처리하는 방식입니다. 다른 작업의 결과가 자신의 작업에 영향을 주는 경우 활용할 수 있습니다.

이해를 돕기 위해 제가 예시를 생각해보았는데 일단 동기이므로 완료여부를 따지기 때문에 자료조사를 요청하고 결과값을 받아 ppt를 제작하는 상황을 만들고, 제어권을 노트북 가정한다면,  
“자 노트북 빌려 줄게 자료조사 부탁해”라고 요청을 하고, 이제 저는 노트북이 없으니까 하염없이 저 친구가 완료해서 결과값이랑 노트북을 줄 때까지 기다리게 될 것이고 받은 뒤에 ppt제작등 작업을 이어가겠죠??

(슬라이드 9) 다음으로는 sync non-blocking은 다른 작업이 진행되는 동안에도 자신의 작업을 처리하고, 다른 작업의 완료 여부를 바로 받아 순차적으로 처리하는 방식입니다.

이 조합의 예시도 동기이기 때문에 전 예시와 상황은 동일하나 제어권이 노트북이라고 하였을 때 논블록방식이기 때문에 노트북을 넘겨주지 않고 자신이 들고 있게 되는 거죠. 그러면서 뭐 미리 템플릿을 찾아보거나, 카톡으로 다 하셨나요? 라는 등의 완료 여부를 묻는 등 작업은 계속 할 수 있는 거죠.

(슬라이드 10) 다음으로는 Async non-blocking인데, Async non-blocking은 다른 작업이 진행되는 동안에도 자신의 작업을 처리하고, 다른 작업의 완료 여부를 따지지 않아 작업의 순서가 지켜지지 않는 방식입니다.

이 조합의 예시도 논블록이기에 전 예시처럼 노트북을 넘겨주지 않고 자신이 들고 있으면서, 비동기이기 때문에 상황은 팀장이 업무 지시하는 완료여부를 따지지 않는 상황을 만들었고, 그러면 팀장은 작업은 계속 하고 있다가 팀원들의 완료 응답을 받고 아 다 했구나 하는 거죠.

(슬라이드 11) 다음으로 Async blocking은 다른 작업이 진행되는 동안 자신의 작업을 멈추고 기다리면서, 다른 작업의 완료 여부를 따지지 않아 작업의 순서가 지켜지지 않는 방식입니다.

이 조합의 예시도 비동기이기 때문에 상황은 팀장이 업무 지시하는 상황을 만들고, 하지만 블로킹으로 이제 작업 요청을 할 때 노트북도 같이 전달해 주는 거죠. 그러니까 굳이 팀원의 결과값이 필요가 없지만, 노트북이 없어 작업을 못 하는 상황이죠.

(슬라이드 12) 이제 마지막으로 제가 동기 비동기를 눈으로 볼 수 있게 동기 프로그래밍, 비동기 프로그래밍 실습을 준비해 봤는데요.  
일단 제가 hello world를 출력할 예정인데요. Hello와 world사이에 2초가 걸리는 작업을 넣고, 2초가 걸리는 작업이 완료 후 완료라는 단어를 출력할 예정입니다.

먼저 동기 프로그래밍부터 해보겠습니다.

* 먼저 2초가 걸리는 작업을 longWork라는 이름의 함수로 만들어 주겠습니다.
* const now = new Date(); 명령어를 이용해 현재 날짜를 now에 가져오고,
* const milliseconds = now.getTime(); 명령어를 이용해 현재 시간을 밀리초로 가져오겠습니다.
* const afterTwoSeconds = milliseconds + 2\*1000; 다음으로는 milliseconds를 이용해, 코드 실행 시간에서 2초 더한 afterTwoSeconds 변수를 만들어 주겠습니다.
* while문을 현재 루프가 돌아가는 시간이 2초를 더한 시간 보다 작을 때 반복하도록 조건을 달아서 2초동안 반복문이 실행되도록 해줍니다.
* 이제 2초동안의 작업이 끝나고 완료가 출력되도록 해줍니다.
* 이제 Hello 출력 -> longWork -> World 순으로 실행해주면, hello, 완료, world 순으로 출력되는데요.
* 이걸 실행시키고 빨리 내려보면 2초라는 시간 이 지나고 완료 world 이렇게 출력이 됩니다.
* 다시 ppt를 가서 보자면 console.log(‘Hello’)가 실행된 뒤,   
  longWork()에서 2초동안 반복문이 실행될 동안 아무런 작업을 하지 못하고,   
  2초가 지난 후 완료를 출력하고, console.log(‘World’)가 실행되어서 아까 같은 결과가 나왔고, 이는 순차적으로 진행되는 것을 보여줍니다.

다음으로는 비동기 프로그래밍 해보겠습니다.

* 이번에는 longWork 함수에 비동기로 실행되는 함수인 setTimeout함수를 사용하겠습니다.
* setTimeout(함수, 시간)과 같은 형태로 시간 동안 기다렸다가 함수를 실행하는데요. 저는 2초 후에 완료를 출력하도록, setTimeout(()=>{console.log(‘완료’);},2000); 이렇게 적어 주겠습니다.
* 다음에는 동기 프로그래밍과 마찬가지로, console.log(‘Hello’)-> longWork() -> console.log(‘World’) 순으로 적어주고 실행하면,
* 동기 프로그래밍이랑은 다르게 Hello, world 하고 조금 있다가 완료가 출력 되는 것을 볼 수 있습니다.
* 이제 다시 ppt를 가서 보면 console.log(‘Hello’)가 실행된 뒤,   
  longWork()를 실행하면 setTimeout 함수가 실행되자마자 다른 작업이 가능해집니다. 왜냐면 setTimeout 함수는 비동기 함수이기 때문이죠. 다른 작업이 가능해지기 때문에 console.log(‘World’)를 실행시키고 2초 뒤에 완료를 출력하게 되는데  
  이는 순차적으로 실행되지 않는 비동기의 모습을 보여줍니다.

(슬라이드 13) 이상으로 발표를 마치겠습니다. 감사합니다.