10-2. 소유

창의적소프트웨어프로그래밍
2022년도 여름학기
Racin

이번 시간에는

- **Object**에 대한 **소유**권(ownership) 이야기를 구경해 봅니다.
  - **소유**의 의의, **소유**의 주체(**소유**권자), **object** 공유(또는, **이동**)

  - 그 과정에서 아래 내용을 함께 체크해 봅니다:
    - `new` / `delete` / `delete[]` **연산자**
    - **Reference**, `std::move<>()`
    - `auto` specifier

이번 시간에는

- 바로 시작해 봅시다.
  - CSP_10_2_yeshi.cpp를 탑재한 다음 열어주세요
    - 지난 시간에 본 유리수 또 들고 왔어요(이번엔 변경하는 버전이에요)

Object를 소유한다?

- 다시 한 번 짚어 둘께요.
  우리가 다루려는 **object**는 '**값** + 칸'을 의미하므로,
  기본적으로 모든 **object**들은 **runtime**에만 존재해요.
  - 코드 적는 시점이나 compile time에는 기본적으로 존재할 수 없어요
  - 모든 칸들은 기본적으로 메모리 위에 있으며
    '실행중인 프로그램'을 위한 메모리 공간은 **runtime**에만 존재하므로 이는 당연해요
  - 그렇긴 하지만, C++ 컴파일러는 우리가 적은 코드 내용에 따라 **object**들이 적당한 시점에 **생성-파괴**되도록 컴파일해 줄 것이기에, 우리는 눈 앞의 .cpp 파일을 보며 **runtime**에 어떤 **object**가 어떤 삶을 갖게 될 지 유추하거나 의도할 수 있어요

Object를 소유한다?

- 아래 코드를 봅시다:

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

Object를 소유한다?

- 아래 코드를 봅시다:
  - 이 코드만 보았을 때, 이걸 실행하면 유리수 **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

Object를 소유한다?

- 아래 코드를 봅시다:
  - 이 코드만 보았을 때, 이걸 실행하면 유리수 **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

new **연산자**가 하나도 안 보이니 반자동 방식은 잴 필요 없고,
여기에 적절한 **선언** 하나가 존재하니
'한 개'라 말하면 정답일 듯!

Object를 소유한다?

- 아래 코드를 봅시다:
  - 이 코드만 보았을 때, 이걸 실행하면 int **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

Object를 소유한다?

- 아래 코드를 봅시다:
  - 이 코드만 보았을 때, 이걸 실행하면 int **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Mu      (RationalNumber other)
    {
        boon              boonja;
        boonm             nmo;
    }
};
```

여기에 선언이 있긴 한데, 얘는 **멤버 선언**이에요.
따라서 이것만 가지고는 int **object**가 특정되지 않아요.

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

Object를 소유한다?

- 아래 코드를 봅시다:
  - 이 코드만 보았을 때, 이걸 실행하면 int **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

맞아요. 이 **선언**에 의해 유리수 **object** 하나가 **생성**되면서,
해당 **object**에 **포함**되는 int **object** 두 개가 **생성**돼요.
각각을 의도하기 위한 서로 다른 두 **수식**을 적을 수 있음!

Object를 소유한다?

- 아래 코드를 봅시다:
  - 이 코드만 보았을 때, 이걸 실행하면 int **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

사실 int **형식**은 어제보다 더 이전 시점에 마련되었으니,
우리 눈에 안 보이는 다른 어딘가에
int **object**를 다루는 다른 코드가 적혀 있을 수 있긴 해요.

Object를 소유한다?

- 아래 코드를 봅시다:
  - 이 코드만 보았을 때, 이걸 실행하면 int **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(
    {
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;
```

유리수 **형식**은 그렇지 않아요.
Class **정의**가 어제 시점에 작성되었으므로,
그보다 이전 시점에 '유리수 **object**를 다루는 코드'를 적어 놓았을 가능성은 전혀 없어요.
(물론 동일한 **이름**의 다른 무언가를 정했을 순 있지만, 아무튼 걘 얘가 아니에요)

Object를 소유한다?

- 이제 문제를 내 볼까요?
  - 이 프로그램 실행 도중 하나밖에 존재하지 않는 유리수 **object**의 주인은 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

Object를 소유한다?

- 이제 문제를 내 볼까요?
  - 이 프로그램 실행 도중 하나밖에 존재하지 않는 유리수 **object**의 주인은 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int b
    int b

        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

어제 유리수 **형식**을 창시한 김 모 프로그래머?

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

오늘 유리수 **변수 선언**을 적은 이 모 프로그래머?

Object를 소유한다?

- 이제 문제를 내 볼까요?
  - 이 프로그램 실행 도중 하나밖에 존재하지 않는 유리수 **object**의 주인은 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    return 0;
}
```

맞아요. 이 **선언**이 아니었으면 그 **object**는 존재할 수 없었어요.
그러므로 이 모 프로그래머가 주인이라 말할 수 있을 거예요.
절대 강사가 이씨라서 손을 들어 주는 게 아님!

Object를 소유한다?

- 이번엔 코드를 약간 바꾸어 보았어요.

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
}
```

Object를 소유한다?

- 이번엔 코드를 약간 바꾸어 보았어요.
  - 이 코드만 보았을 때, 이걸 실행하면 유리수 **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
}
```

Object를 소유한다?

- 이번엔 코드를 약간 바꾸어 보았어요.
  - 이 코드만 보았을 때, 이걸 실행하면 유리수 **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

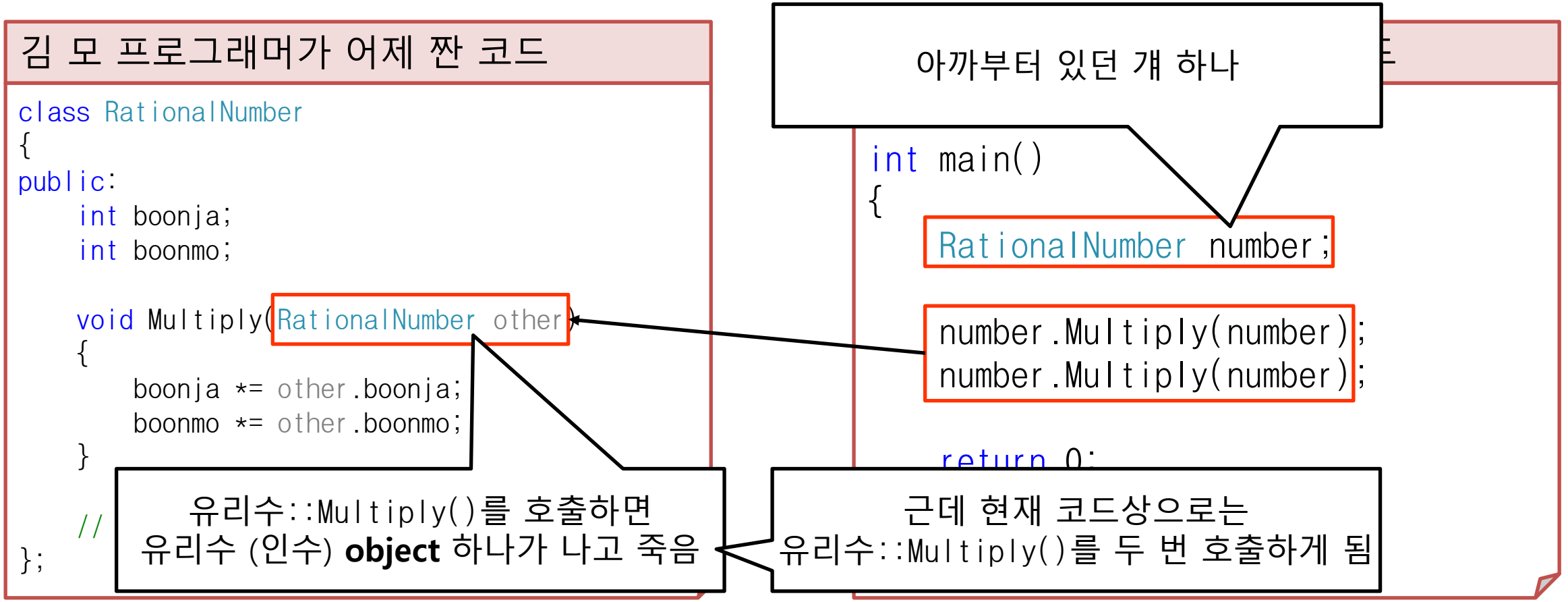
이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
```

답은 '세 개'예요.
왜 세 개가 되었냐를 따진다면...

Object를 소유한다?

- 이번엔 코드를 약간 바꾸어 보았어요.
  - 이 코드만 보았을 때, 이걸 실행하면 유리수 **object**가 몇 개 살다 죽게 될까요?



Object를 소유한다?

- 이번엔 코드를 약간 바꾸어 보았어요.
  - 이 코드만 보았을 때, 이걸 실행하면 유리수 **object**가 몇 개 살다 죽게 될까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.b
        boonmo *= other.b
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number                    ber);
}
```

좀 더 정밀하게 본다면 이 **선언** 또한
'main()이 호출될 때마다 하나'를 의미한다는 것을 간파할 수 있을 거예요!
다만 main()이 매 프로그램 실행마다 단 한 번만 호출되고 있을 뿐임!

Object를 소유한다?

- 이제 또 문제예요.
  - 현재 이 프로그램에 세 개 있는 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
}
```

Object를 소유한다?

- 이제 또 문제예요.
  - 현재 이 프로그램에 세 개 있는 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    n       .Multiply(number);
            .Multiply(number);
```

이 **선언**에 의해 특정되는 **object** 하나의 주인은,
이 **선언**을 적은 이 모 프로그래머였어요.

Object를 소유한다?

- 이제 또 문제예요.
  - 현재 이 프로그램에 세 개 있는 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);
```

같은 논리로 본다면, 이 인수 **선언**에 의해 특정되는 **object** 두 개의 주인은...

Object를 소유한다?

- 이제 또 문제예요.
  - 현재 이 프로그램에 세 개 있는 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);
```

같은 논리로 본다면, 이 인수 **선언**에 의해 특정되는 **object** 두 개의 주인은...
비록 그 **object**가 이 모 프로그래머의 의도에 의해 복사 생성된다 하더라도
여전히 '실제로 이 **선언**을 적은' 김 모 프로그래머가 맞다고 할 수 있어요!

소유의 주체

- 잠시 정리하면...
  - **Object**의 주인은 기본적으로 '그 **object**를 생성하는 코드를 적은 프로그래머'인 듯 해요
    - 사실 반자동 방식에서도 'new 적은 사람'이 주인이라 생각할 수 있을 듯?
  - 어떤 **object**의 주인은 기본적으로 그 **object**에 **포함**된 멤버 **object**들의 주인이기도 해요

소유의 주체

- 잠시 정리하면...
  - **Object**의 주인은 기본적으로 '그 **object**를 생성하는 코드를 적은 프로그래머'인 듯 해요
    - 사실 반자동 방식에서도 'new 적은 사람'이 주인이라 생각할 수 있을 듯?
  - 어떤 **object**의 주인은 기본적으로 그 **object**에 **포함**된 멤버 **object**들의 주인이기도 해요
  - 사실 지금은 둘이 각각 **함수 정의**를 하나씩만 적었으니 이렇게 볼 수 있을 것 같아요.
    약간'만' 더 정밀하게 말한다면, **object**의 주인은 기본적으로
    '그 **object**를 **생성**하는 코드가 내용물 안에 적힌 **함수**(의 이번 호출)'라 볼 수 있어요!
    - 방금 본 코드를 혼자 다 만들었다 하더라도,
      `main()` **정의** 안에서 **이름** `other`를 직접 사용하여 그로 인해 특정되는 **object**를 다룰 순 없어요!
    - 어렵게 적긴 했지만 뭐 이미 다 알고 있는 내용이지요?

Object를 소유하지 않는다?

- 여기서, 코드를 약간 고쳐 봅시다.
  - 유리수::Multiply()의 인수 **선언**을 아래와 같이 고쳐 보세요. (예시 코드 155줄)

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &other)
    {
        boonja *= other.boonj
        boonmo *= other.boo
    }

    // 이외에도
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
```

Declarator에 & 한 글자를 추가하면 돼요.
고쳤으면, 바로 실행해 봅시다.

Object를 소유하지 않는다?

- 우리 논리대로라면 이제 이 코드는 유리수 **object**를 하나만 다루고 있어요!
  - 그 **object**의 주인이 누구인지는 굳이 짚지 않아도 되겠지요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
}
```

Object를 소유하지 않는다?

- 우리 논리대로라면 이제 이 코드는 유리수 **object**를 하나만 다루고 있어요!
  - 그 **object**의 주인이 누구인지는 굳이 짚지 않아도 되겠지요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
```

이제 이 인수 **선언**은 더 이상 '이 **함수**가 주인인' 유리수 **object**를 특정하지 않아요.
그럼에도 출력 결과가 동일하다는 점을 생각해 보면...

Object를 소유하지 않는다?

- 우리 논리대로라면 이제 이 코드는 유리수 **object**를 하나만 다루고 있어요!
  - 그 **object**의 주인이 누구인지는 굳이 짚지 않아도 되겠지요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(number);

    return 0;
```

이제 이 인수 **선언**은 더 이상 '이 **함수**가 주인인' 유리수 **object**를 특정하지 않아요.
그럼에도 출력 결과가 동일하다는 점을 생각해 보면...
인수 **이름** other는 '호출자가 주인인' 유리수 **object**를 특정한다고 볼 수 있어요!

Object를 소유하지 않는다?

- 여기서 코드를 조금만 더 바꾸어 볼까요?
  - main() **정의** 안 **멤버 함수** 호출식을 아래와 같이 고쳐 봅시다. (예시 코드 283줄 근처)

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(3);

    return 0;
}
```

Object를 소유하지 않는다?

- 옹... 빨간 줄이 뜨는군요.
  - **수식** 3은 '호출자가 주인인 유리수 **object**'를 특정할 수 없는 **수식**이므로 그런 듯 해요

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(3);

    return 0;
}
```

Object를 소유한다?

- 여기서 아까 유리수::Multiply()에 살짝 적어 놨던 &을 다시 지우면?
  - (예시 코드 155줄)

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(3);

    return 0;
}
```

Object를 소유한다?

- 오... 이번에는 무리 없이 잘 실행되고 있습니다.
  - **수식** 3을 가지고 '인수 other에 의해 특정되는 **object**'의 (일반)**생성자**를 호출하고 있어요!

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(3);

    return 0;
}
```

Object를 소유한다?

- 자, 한 번만 더 유리수 **class**를 괴롭혀 봅시다.
  - 이번에는 유리수::Multiply()의 인수 **선언**에 &&을 붙여 보세요! (예시 코드 155줄)

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(3);

    return 0;
}
```

Object를 소유한다?

- 오? 매우 오묘한 반응을 보이고 있어요.
  - 이번에는 거꾸로 '호출자가 주인인 유리수 **object**'를 받지 않으려 하고 있어요

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(number);
    number.Multiply(3);

    return 0;
}
```

Object를 소유하지 않는다?

- 오류 나는 줄을 아래와 같이 고친 다음 한 번 실행해 봅시다.
  - (예시 코드 283줄 근처)

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);
```

인수 자리에 number를 적어 std::move()를 호출하고
그 결과를 유리수::Multiply()에 담고 있어요.

Object를 소유하지 않는다?

- 오? 이번에는 유리수 **object**가 두 개만 살다 가고 있습니다.
  - std::move() 어쩌구 적은 **멤버 함수** 호출식은 새 유리수 **object**를 생성하지 않나봐요

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);

    return 0;
}
```

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);

    return 0;
}
```

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);
```

일단 **수식** 3을 통해 **생성**하는 **object**는
(비록 **수식** 3을 main()에서 적기는 했지만)
명백히 유리수::Multiply()가 주인이라고 말할 수 있을 거예요.

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(
```

문제는 이 친구예요.
얘는 주인이 누구라고 말해야 할까요?

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(2);
```

정답 비슷한 것을 말해버리면, 이 **object**는
(이번 호출에 대한) '유리수::Multiply() 내용물이 **실행**되는 동안은
유리수::Multiply() 것'이 돼요!

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);
```

첫 호출에서 인수 other에 의해 특정되는 유리수 **object**는
잠시 유리수::Multiply() 것처럼 쓰이다가
함수 정의 내용물 **실행**이 끝나면 다시 main() 것으로 간주돼요.
(그래서 그 **object**는 main() **정의** 내용물 **실행** 끝나는 시점 즈음에 알아서 **파괴**돼요)

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);
```

두 번째 호출에서 인수 other에 의해 특정되는 유리수 **object**는
잠시 유리수::Multiply() 것처럼 쓰이다가
함수 정의 내용물 **실행**이 끝나면 '주인이 없기 때문에' 파괴돼요.
(그 **object**는 **멤버 함수** 호출식이 포함된 **수식 계산**이 끝나는 타이밍 즈음에 알아서 파괴돼요)

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);
```

두 경우 모두,
유리수::Multiply()의 입장에서는 '내 것'이라 말할 수 있어요.

Object를 소유하지 않는다?

- 자, 이제 마지막 문제예요.
  - 관찰되고 있는 두 유리수 **object**들의 주인은 각각 누구일까요?

김 모 프로그래머가 어제 짠 코드

```
class RationalNumber
{
public:
    int boonja;
    int boonmo;

    void Multiply(RationalNumber &&other)
    {
        boonja *= other.boonja;
        boonmo *= other.boonmo;
    }

    // 이외에도 이것저것 다 짜 둠
};
```

이 모 프로그래머가 오늘 짠 코드

```
int main()
{
    RationalNumber number;

    number.Multiply(std::move(number));
    number.Multiply(3);
```

main()의 입장에서는,
첫 호출에서는 자신이 **선언**해서 만든 유리수 **object**를 잠시 '양도'하며,
두 번째 호출에서는 유리수 **object** 없이 호출하고 있어요.

소유의 주체: 함수

- 좋아요. 우리는 이제까지 몇 가지 문제를 생각해 보면서,
  함수를 **object** 소유의 주체로 놓고 보았을 때
  인수의 **형식**에 따라 **object**가, 또는 **object**의 소유권이 어떻게 간주되는지
  살짝 구경해 보았어요.
  - 마지막에 본 '양도' 부분을 제외하면 그럭저럭 납득할 수 있었음!

소유자의 권리(와 의무)

- 좋아요. 우리는 이제까지 몇 가지 문제를 생각해 보면서,
  함수를 **object** 소유의 주체로 놓고 보았을 때
  인수의 **형식**에 따라 **object**가, 또는 **object**의 소유권이 어떻게 간주되는지
  살짝 구경해 보았어요.
  - 마지막에 본 '양도' 부분을 제외하면 그럭저럭 납득할 수 있었음!

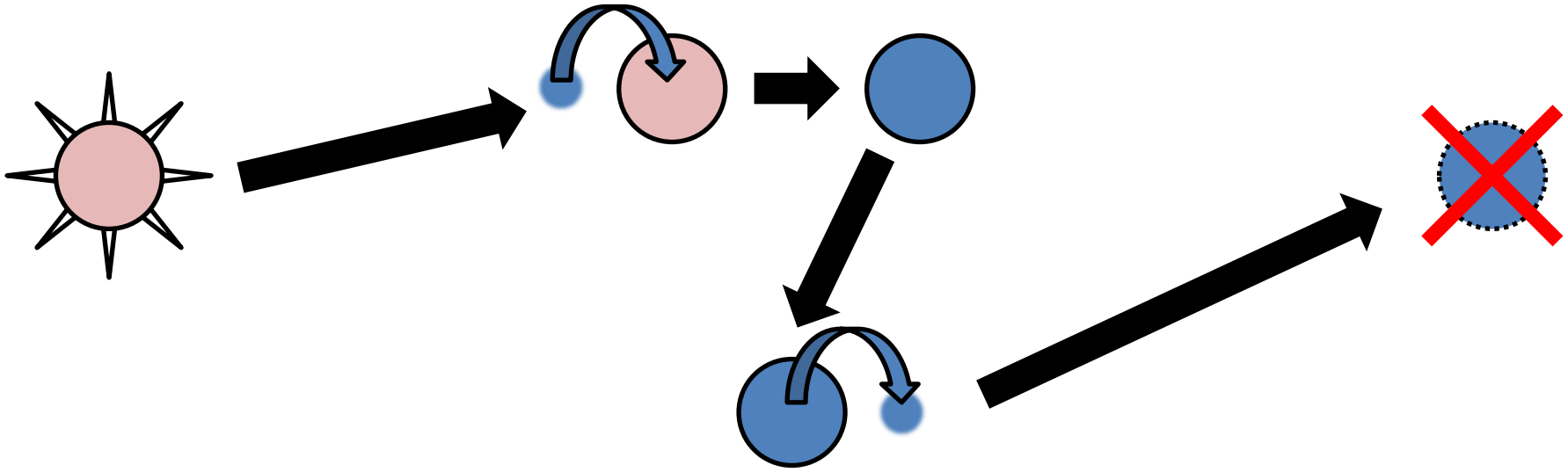
- 이쯤 오면 근본적인 질문을 갖게 된 친구들도 있을 듯 해요.
  - 아니, 어차피 **함수** 호출 끝나면 인수고 뭐고 다 날아갈 건데
    뭐 굳이 이걸 구분해서 볼 필요가 있나요?
    - 사실 이런 거 전혀 생각 안 하고 코드 짜도 과제같은 거 할 때 전혀 지장 없었잖아요
    - 사실 우리를 괴롭혔던 건 *이었는데, 정작 얘는 아직 한 번도 등장하지 않았잖아요

소유자의 권리(와 의무)

- 일단 진정해요.
  - 이전에 등장했던 개념들을 추가로 들고 와서 문제를 더 복잡하게 만들어 볼께요

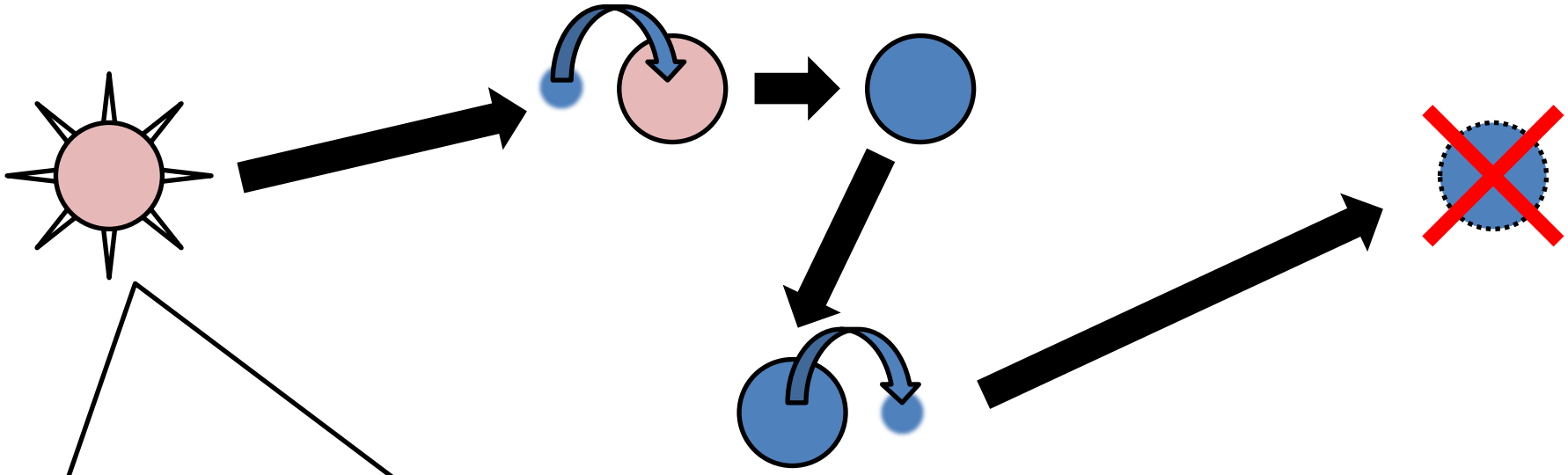
소유자의 권리(와 의무)

- 일단, **object**의 일생에 대한 그림을 다시 가져왔어요:



소유자의 권리(와 의무)

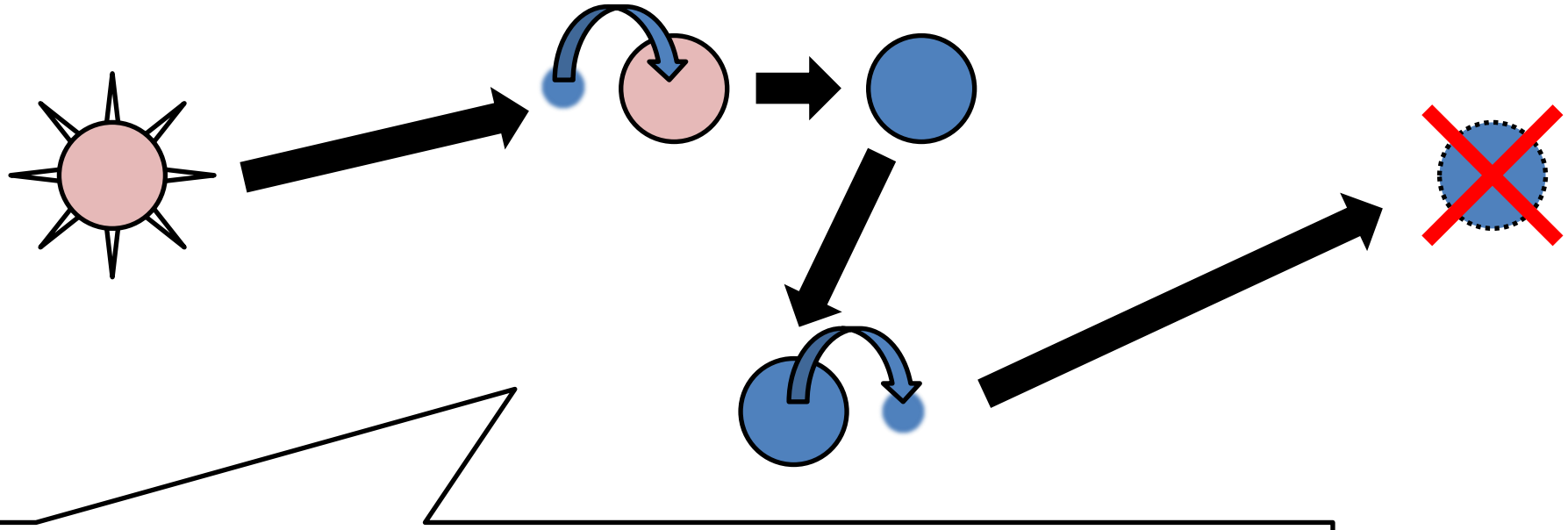
- 일단, **object**의 일생에 대한 그림을 다시 가져왔어요:



이제까지 우리가 다룬 것을 요약하면,
'칸을 형성시킨(**object**를 **생성**한) 자가 그 **object**의 주인이다'
...정도가 돼요.

소유자의 권리(와 의무)

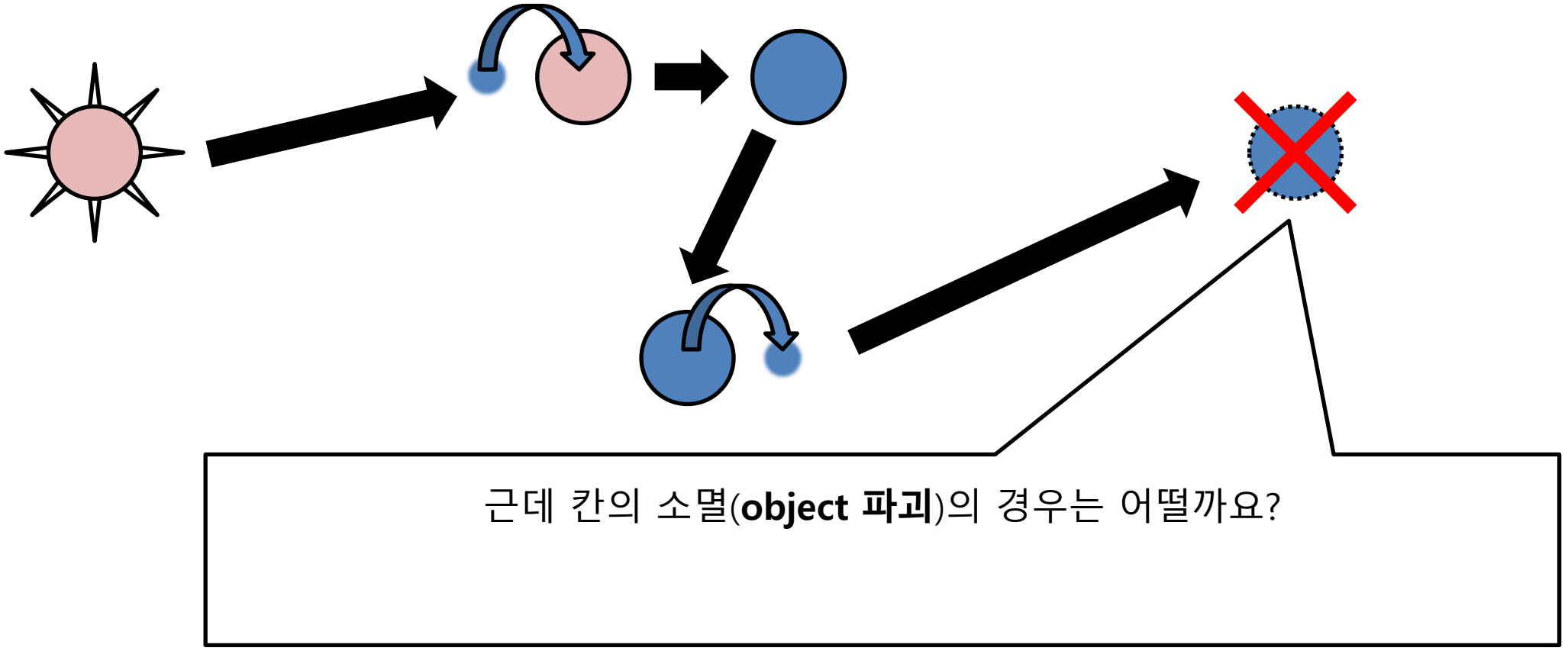
- 일단, **object**의 일생에 대한 그림을 다시 가져왔어요:



예시 코드의 유리수::Multiply()를 생각해 보면,
object를 **소유**하고 있는지 여부는
그 **object**를 사용할 수 있는지 여부와는 일치하지 않아요.
(인수에 & 붙였을 때도 우리는 main() **소유**의 **object**를 사용할 수 있었어요)

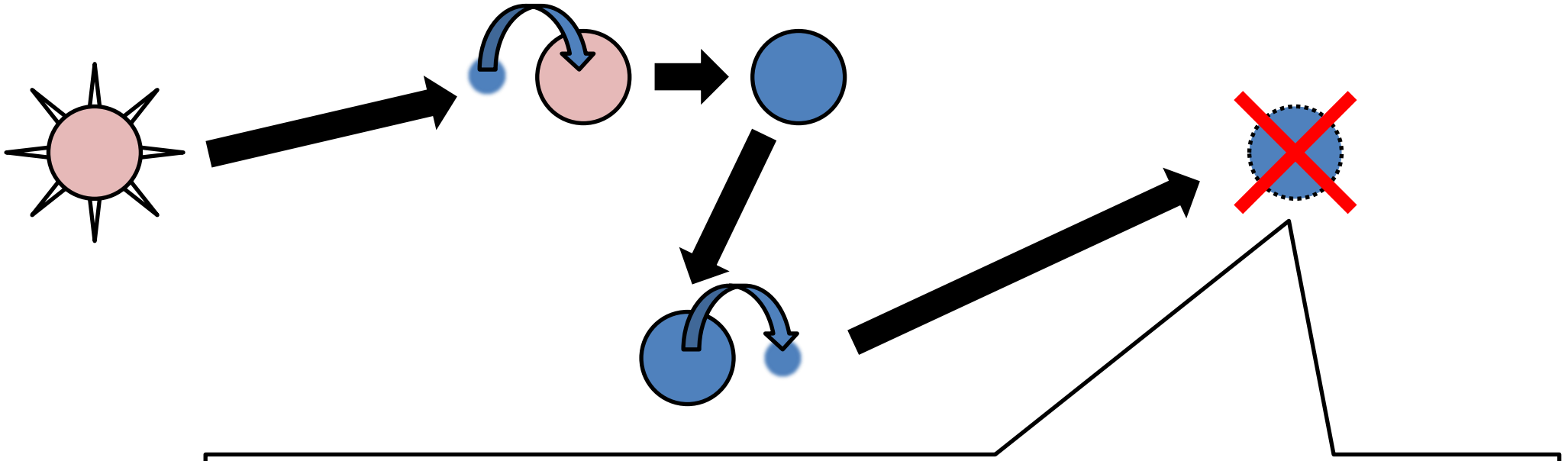
소유자의 권리(와 의무)

- 일단, **object**의 일생에 대한 그림을 다시 가져왔어요:



소유자의 권리(와 의무)

- 일단, **object**의 일생에 대한 그림을 다시 가져왔어요:



근데 칸의 소멸(**object 파괴**)의 경우는 어떨까요?
소유권을 가지지 않음에도 불구하고 '남의 **object**를 무턱대고 **파괴**'하는 것은
현실에서도 하면 안 되는 나쁜 행동인 것 같아요.

소유자의 권리(와 의무)

- 칸권 중 하나인 '소멸할 권리'를 생각한다면,
  특히, 반자동 방식으로 새 **object**를 만들 예정이라면,
  두 **함수**(호출자, 호출받는 자)들 중 '누가 **파괴**할 것인지'를 결정하는 것은
  생각보다 중요한 문제가 될 수 있어요.
  - 기본적으로는 '**생성**한 쪽에서 **파괴**'가 원칙이겠지만,
    C 할 때 본 New_Game() 같은 것을 생각해 보면 늘 그게 가능한 건 아닌 듯 해요

소유자의 권리(와 의무)

- 칸권 중 하나인 '소멸할 권리'를 생각한다면,
  특히, 반자동 방식으로 새 **object**를 만들 예정이라면,
  두 **함수**(호출자, 호출받는 자)들 중 '누가 **파괴**할 것인지'를 결정하는 것은
  생각보다 중요한 문제가 될 수 있어요.
  - 기본적으로는 '**생성**한 쪽에서 **파괴**'가 원칙이겠지만,
    C 할 때 본 `New_Game()` 같은 것을 생각해 보면 늘 그게 가능한 건 아닌 듯 해요

- 그래서 우리는 단어 **소유**(ownership),
  구체적으로 **object**에 대한 **소유**를 다시 규정해 놓고 넘어가려 해요:
  - **Object**의 **소유자**(owner)는 그 **object**를 **파괴**할 권리(와 의무)를 가져요!
    - 방금 예시에서는 automatic **object**들만을 다루느라 크게 고민하지 않아도 되었을 뿐임!

소유권을 넘긴다는 것의 의미

- 이렇게 놓고 보면, 방금 슬쩍 등장한 '소유권 넘기기'의 의미 또한 자연스럽게 따라오게 될 거예요.

소유권을 넘긴다는 것의 의미

- 이렇게 놓고 보면, 방금 슬쩍 등장한 '소유권 넘기기'의 의미 또한 자연스럽게 따라오게 될 거예요.
  - 비록 내가 **생성**한(또는 누군가에게 받은) **object**지만, 이를 **파괴**할 권리(와 의무)를 다른 이에게 넘기는 것!
  - 어렵지 않지요?
    C++가 칸권을 정교하게 신경 쓰는 프로그래밍 언어라는 점을 생각하면
    지금 우리가 이 부분을 나름 중요하게 바라볼 필요가 있다는 것도 납득할 수 있을 거예요!

조금 더 구경해 보기

- 이제 다시 코드로 돌아가서, 강사와 함께 몇 가지 실험을 해 보고 올께요.
  - 방금 간단히 적어 본 &, &&에 대한 실험
  - new / delete / delete[] **연산자**에 대한 실험
  - 중간에 (C++의) auto specifier도 슬쩍 사용해 볼 예정이에요
- 오늘은 구경만 하고, 정리는 다음 시간에 해 볼께요.

마무리