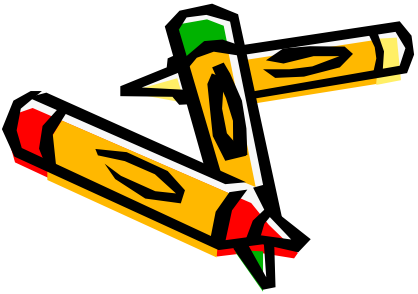
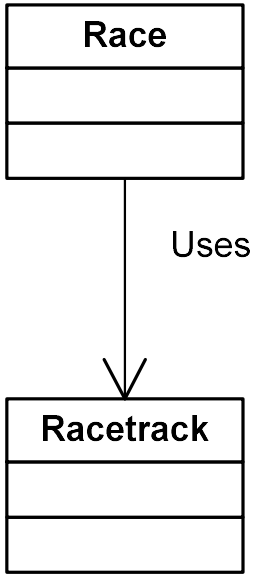
Uses

```
// Race.h

#include "Racetrack.h"

class Race
{
public:

    Race(Racetrack &);
    void go();

private:

    Racetrack & rt;

    // private data members and utility functions to
    // conduct the race simulation
};
```

```
// Race.cpp

Race::Race(Racetrack & r)
:rt(r)
{}

void Race::go()
{

    // some code that calls utility functions
    // and uses rt to run the race

};
```

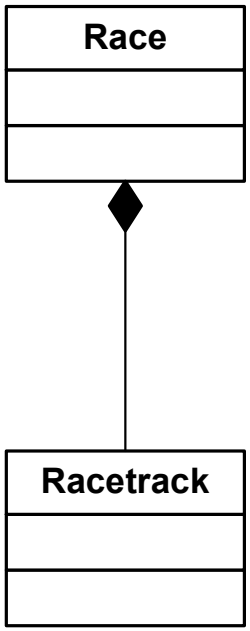
```
// main.cpp

#include <cstdlib>
#include "Racetrack.h"

void main()
{
    system("pause");
    Racetrack rt;
    Race r(rt);
    r.go();
}
```





Composition (has-a)

Strict Aggregation
Implementation as Concrete Data Member




```
// Race.h

#include "Racetrack.h"

class Race
{
public:
    void go();

private:
    Racetrack rt;

    // other private data members and utility functions to
    // conduct the race simulation. these utility functions
    // have access to the class-scoped rt object
};
```

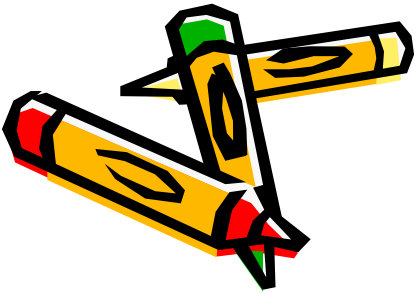
Since `Racetrack` is a Data Member of `Race`, the `Racetrack` object is created with `Race`, and is destroyed with `Race`. Thus "full" or "strict" aggregation.

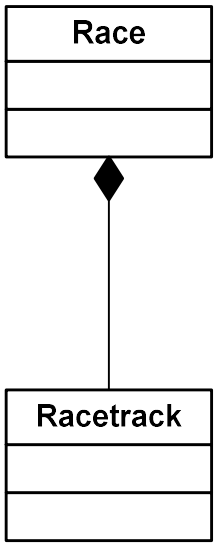
Composition (has-a)

Strict Aggregation
Implemented with Dynamic Memory Allocation

```
// Race.h

class Race
{
public:

    Race();
    ~Race();
    void go();

private:

    Racetrack *rt;
    // private data members and utility functions to
    // conduct the race simulation
};
```

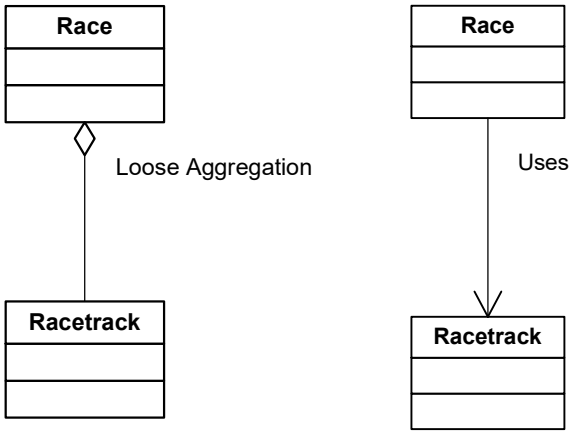
Since `Racetrack` is allocated in the `Race` constructor, the `Racetrack` object is created with `Race`, and is destroyed with `Race` in the `Race` destructor. Thus "full" or "strict" aggregation.






```
#include "Racetrack.h"
#include "Race.h"

Race::Race()
{
    rt = new Racetrack;
}

Race::~Race()
{
    if (rt) delete rt;
}

void Race::go()
{

    // some code that calls utility functions
    // these utility functions have access to the
    // class-scoped rt object

};
```

Loose Aggregation (has-a)



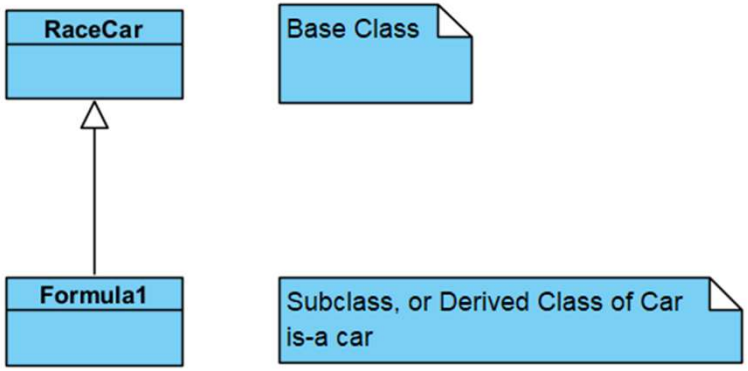

```
// Race.h

class Race
{
public:
    void go();

private:

    // private data members and utility functions to
    // conduct the race simulation
};
```

Since `Racetrack` is not created as part of the construction of `Race`, `Racetrack` has a different life span. Thus is not considered composition, rather loose aggregation.

```
// Race.cpp

#include "Racetrack.h"
#include "Race.h"

void Race::go()
{
    Racetrack rt;

    // some code that calls utility functions
    // rt object is passed as a parameter to these
    // utility functions because it has function scope,
    // not class scope.
};
```

Inheritance (is-a)

- A new class extends an existing class.
- The original class is the "base class", and the new class (or subclass) is the "derived" class.




```
// RaceCar.h

class RaceCar
{
public:

    RaceCar();
    ~RaceCar();
    void DriveFast;

protected:
    char[128] color;
};
```

```
// Formula1.h
#include RaceCar.h

class Formula1 : public RaceCar
{
public:

    Formula1();
    ~Formula1();

private:
    char[128] luverStyle;
};
```

In C++, the ONLY inheritance type that is an is-a relationship is public inheritance