Object Oriented Programming

Lecture 9

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Composition and Aggregation

Lets Recall

OOP: “its all about code reuse”

Lets Recall

OOP: “its all about code reuse”

One way is to

Use object of **one class** in **another class**

Lets Recall

OOP: “its all about code reuse”

One way is to

Use object of **one class** in **another class**

Recall !!!!

Lets Recall

OOP: “its all about code reuse”

One way is to

Use object of **one class** in **another class**

Composition

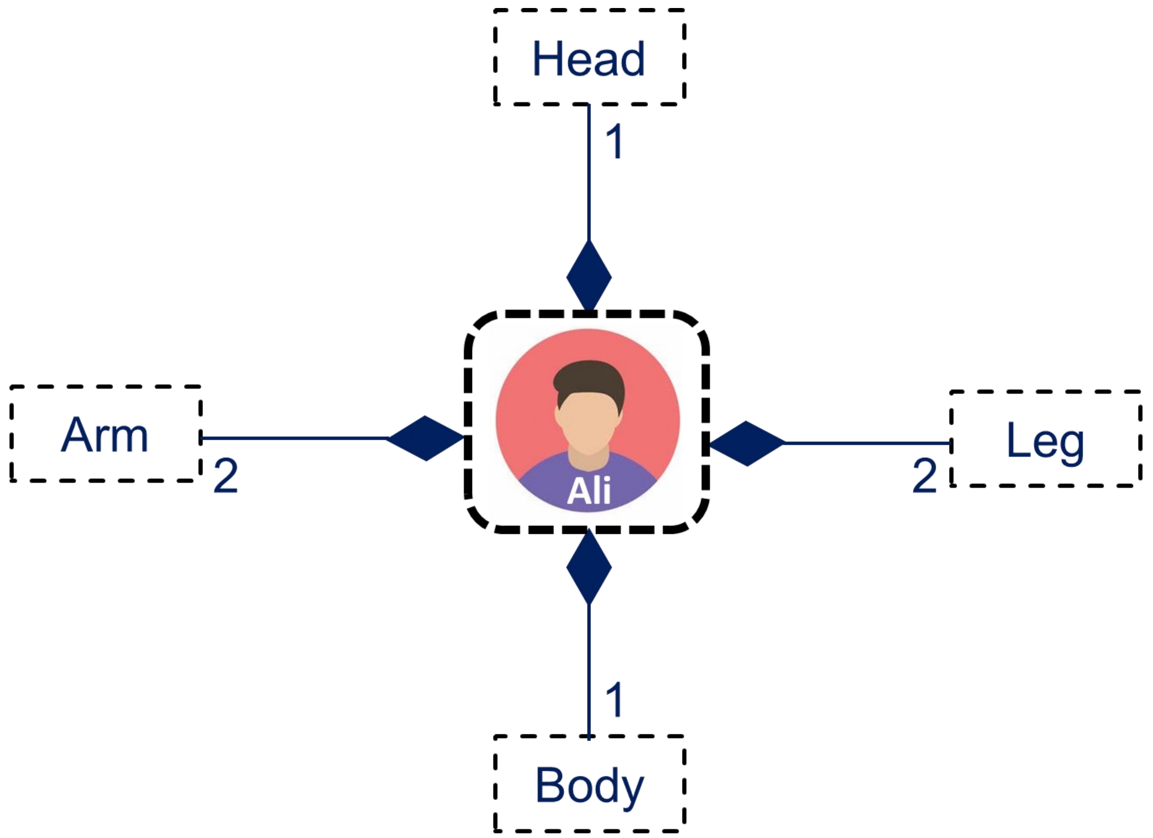
Aggregation

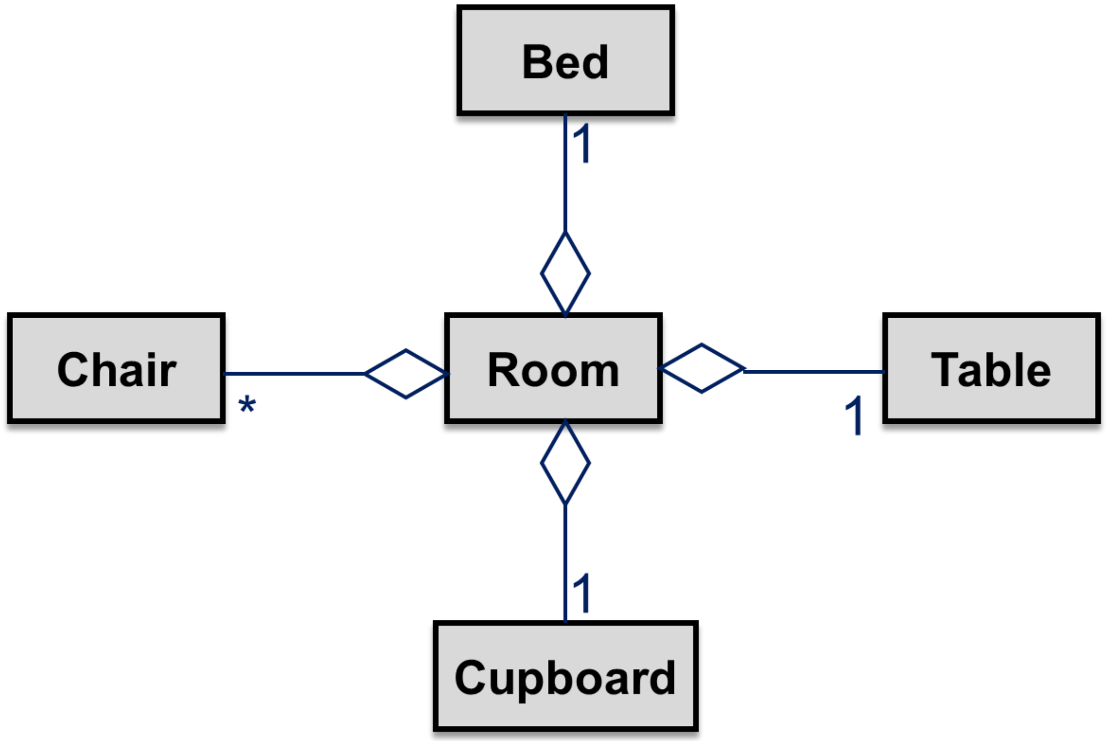
Lets Recall

OOP: “its all about code reuse”

Composition

Aggregation

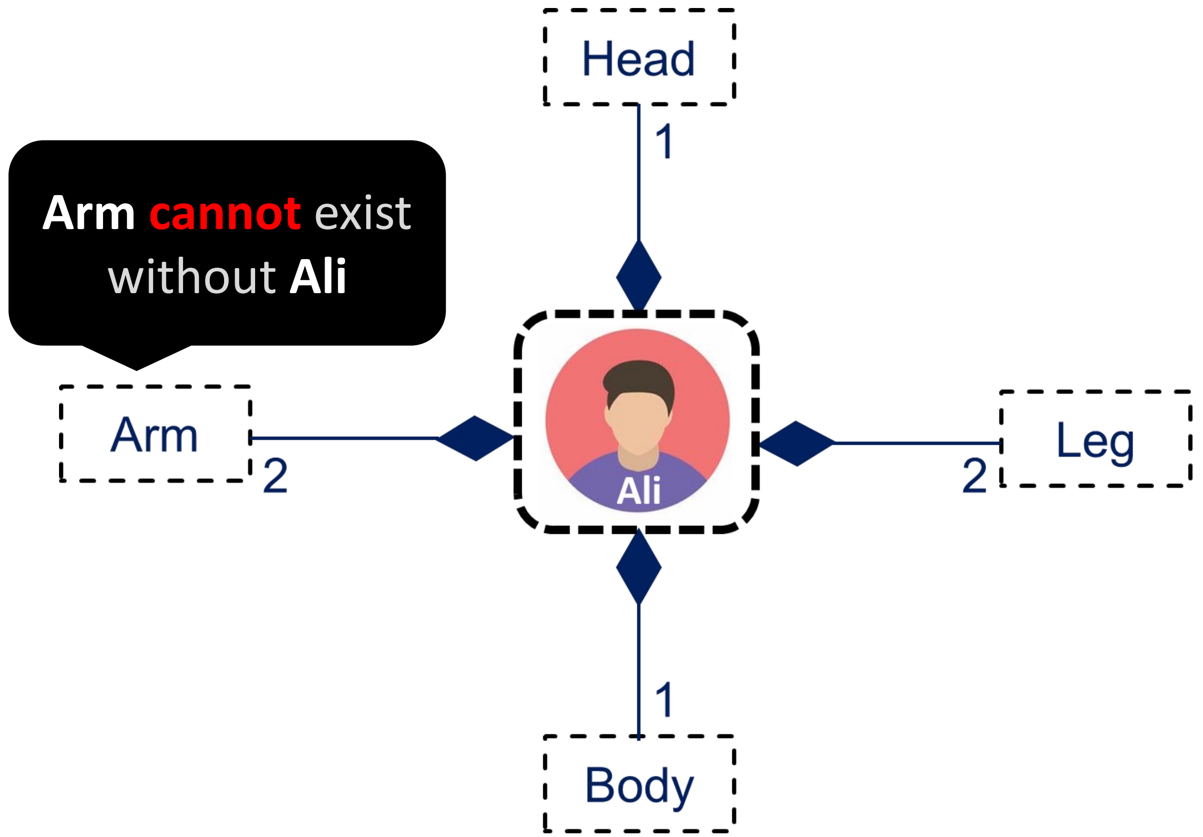




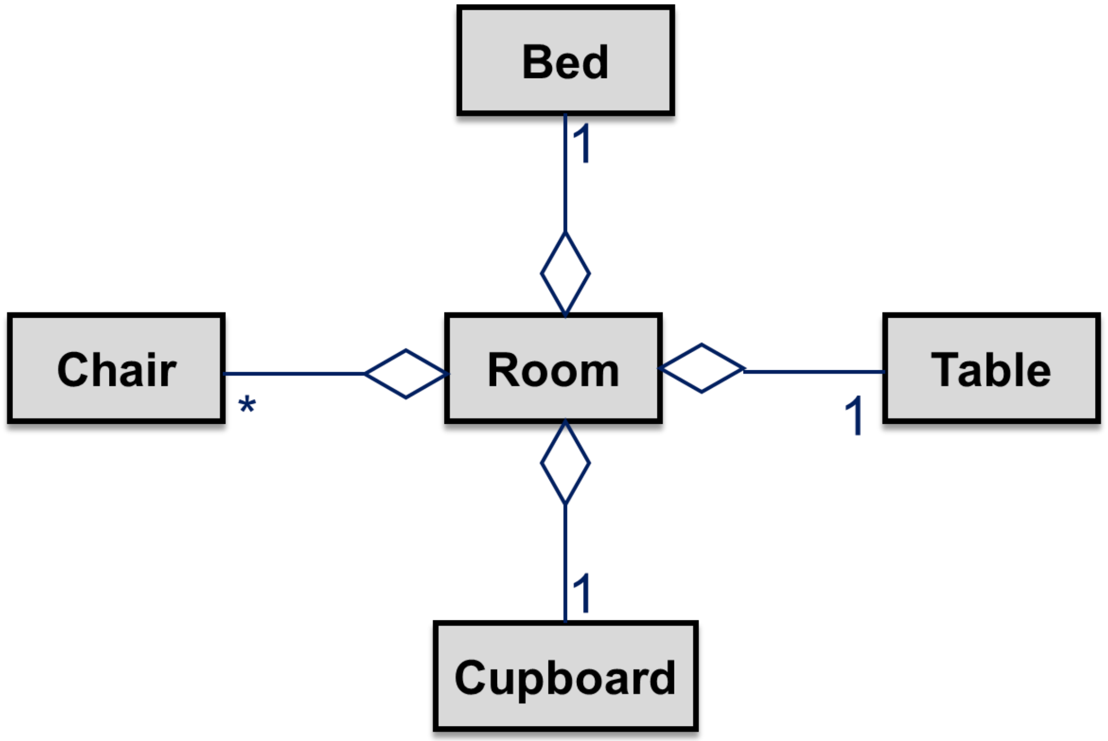
Lets Recall

OOP: “its all about code reuse”

Composition



Aggregation

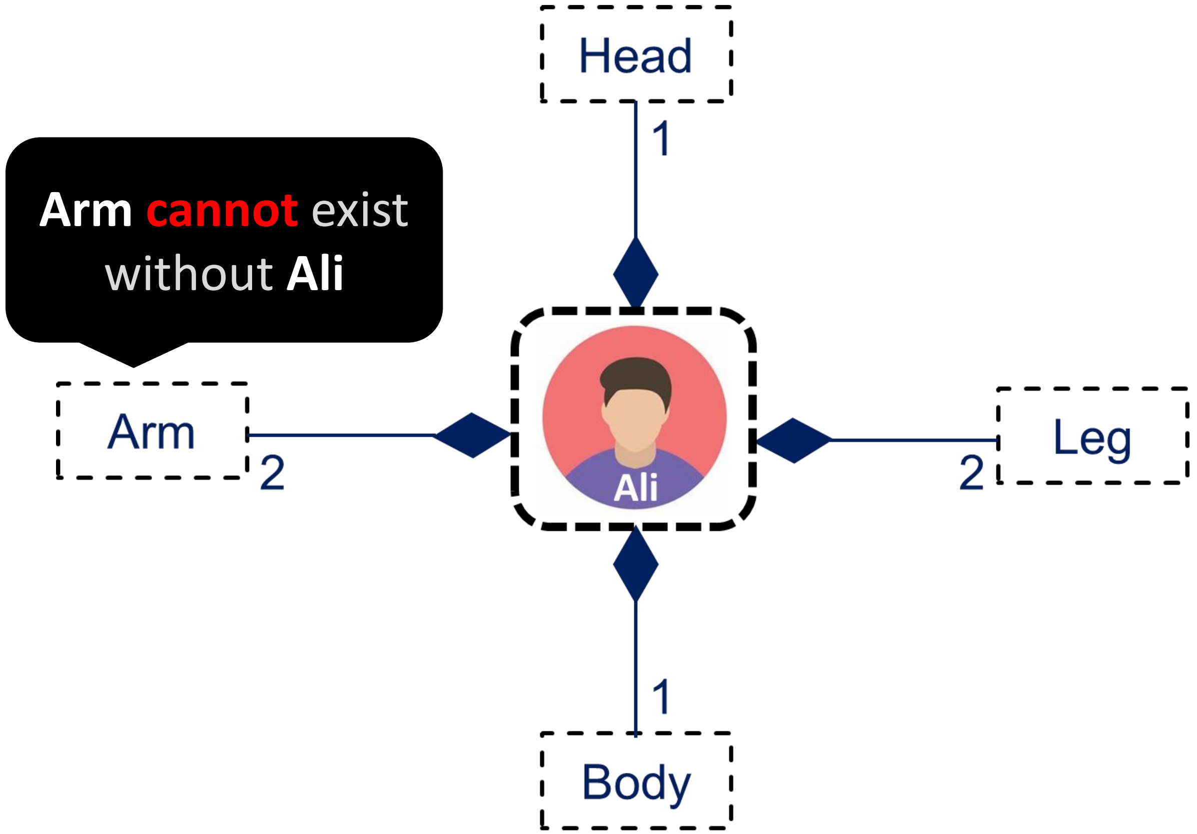


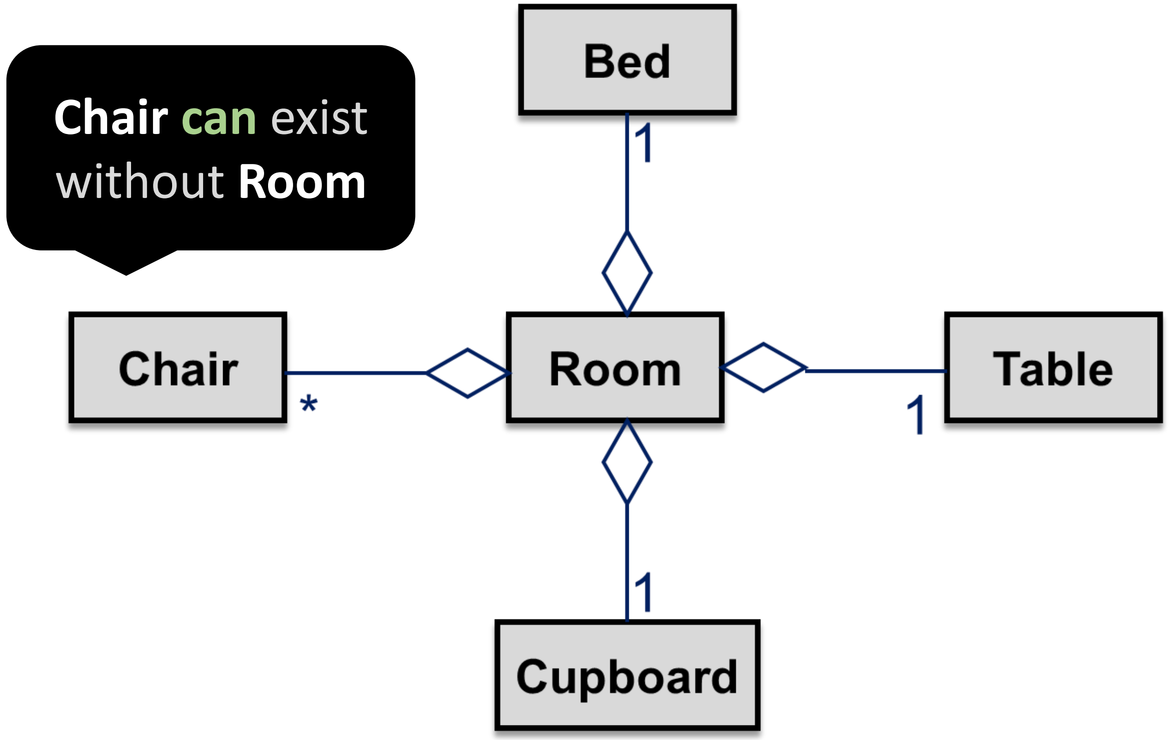
Lets Recall

OOP: “its all about code reuse”

Composition

Aggregation



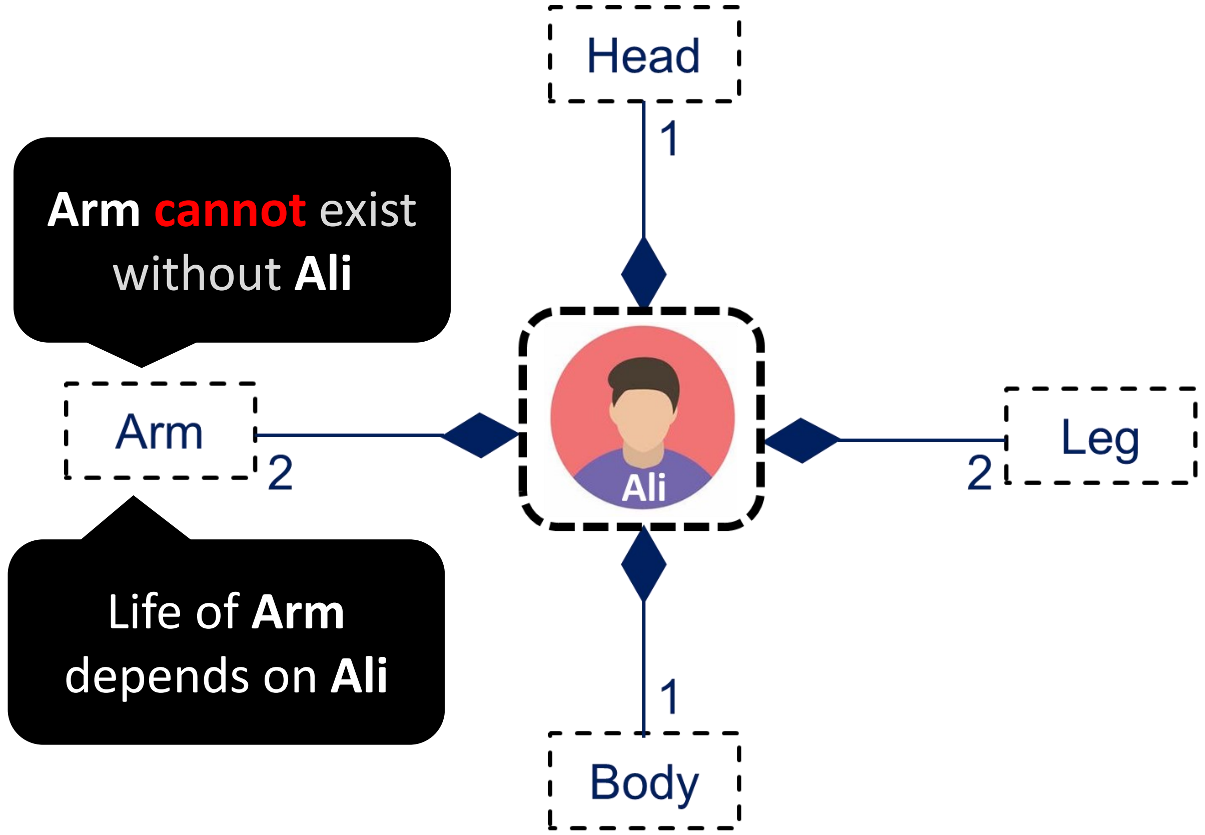


Lets Recall

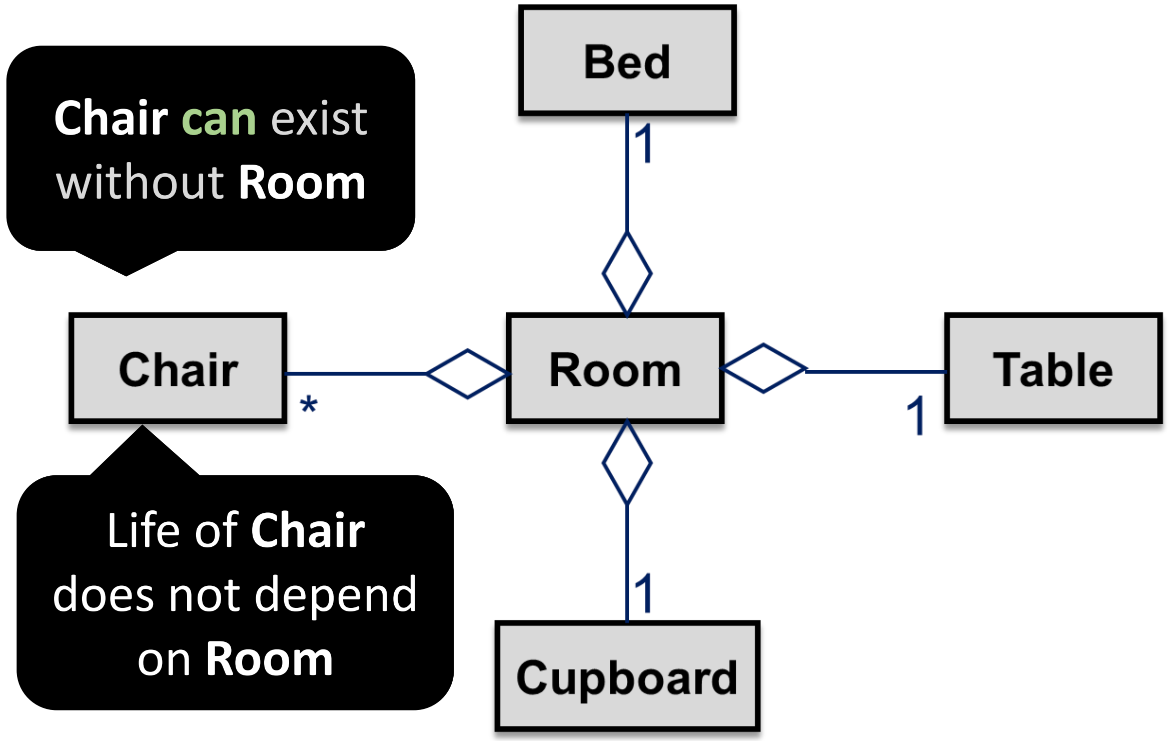
OOP: “its all about code reuse”

Composition

Aggregation



Arm cannot exist without Ali
Head
1
Arm
2
Ali
Leg
2
1
Body
Life of **Arm** depends on **Ali**



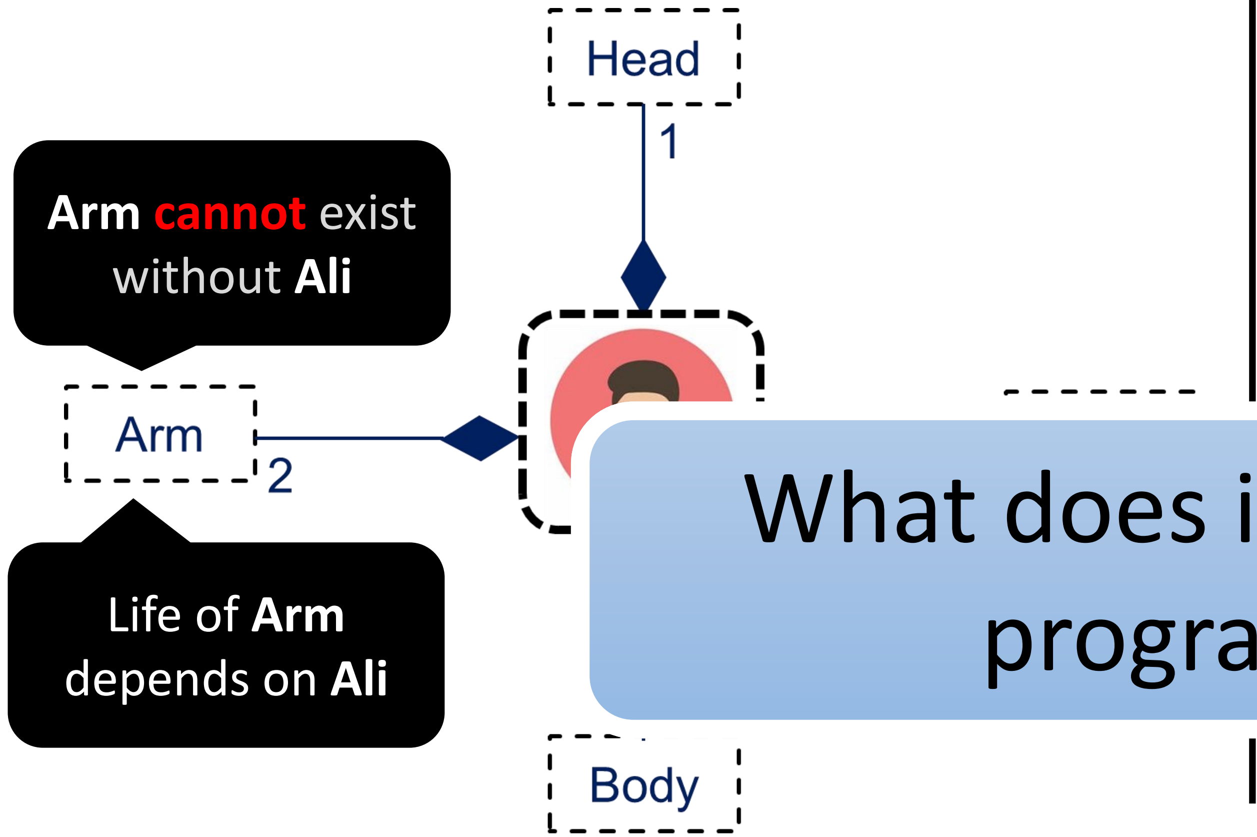
Chair can exist without Room
Bed
1
Chair
*
Room
Table
1
1
Cupboard
Life of **Chair** does not depend on **Room**

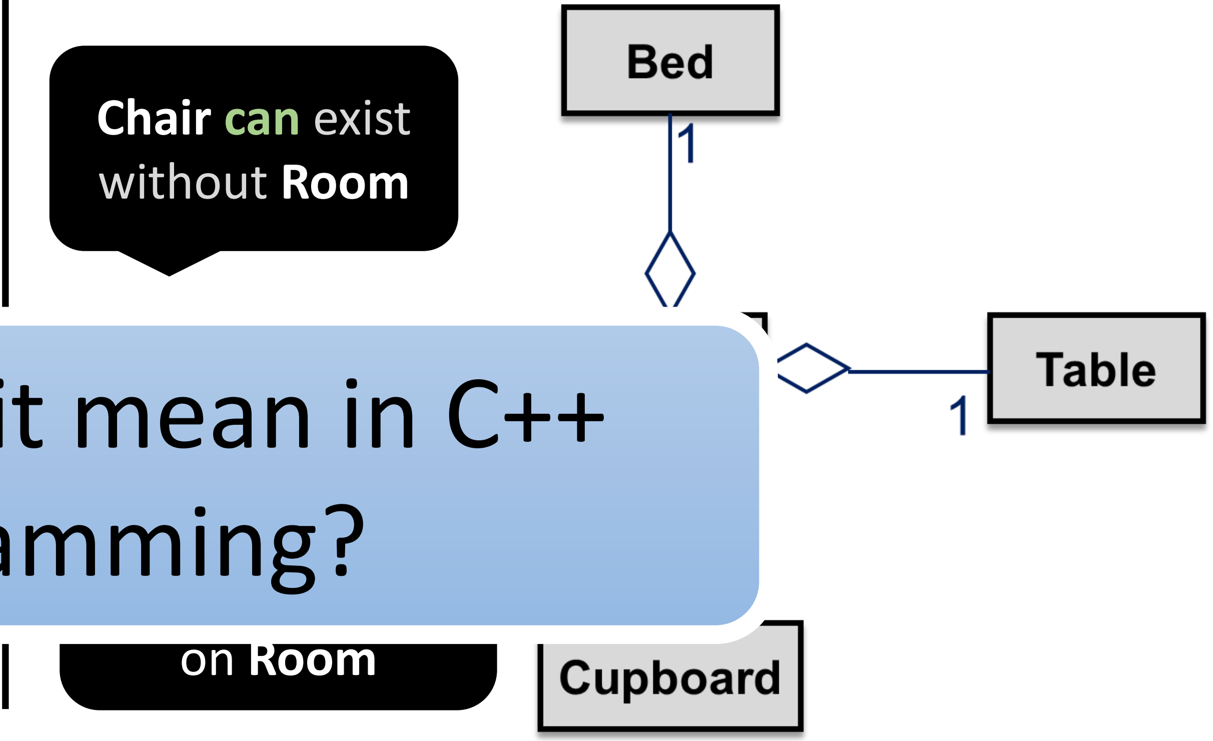
Lets Recall

OOP: “its all about code reuse”

Composition

Aggregation





What does it mean in C++ programming?

In C++

Composition

Aggregation

```
class Bar {
};


class Foo {
Private:
    Bar bar1;
}


void main {

    Foo foo1;
}
```

In C++

Composition

Aggregation

```
class Bar {
};


class Foo {
Private:
    Bar bar1;
}


void main {

    Foo foo1;
}
```

Foo own **Bar** object and responsible for **Bar** lifetime. When **Foo** dies, so does **Bar**

In C++

Composition

```
class Bar {
};

class Foo {
Private:
    Bar bar1;
}

void main {

    Foo foo1;
}
```

Foo own **Bar** object and responsible for **Bar** lifetime. When **Foo** dies, so does **Bar**

Aggregation

```
class Bar {
};

class Foo {
Private:
    Bar* bar1;
    Foo(*Bar X)
    {
        bar1=X;
    }
}

void main {
    Bar a
    Foo foo1(&a);
}
```

In C++

Composition

```
class Bar {
};

class Foo {
Private:
    Bar bar1;
}

void main {

    Foo foo1;
}
```

Foo own **Bar** object and responsible for **Bar** lifetime. When **Foo** dies, so does **Bar**

Aggregation

```
class Bar {
};
class Foo {
Private:
    Bar* bar1;
Public:
    Foo(*Bar X)
    {
        bar1=X;
    }
}

void main {
    Bar a
    Foo foo1(&a);
}
```

Foo has an object which it borrowed from someone else. When **Foo** dies, **Bar** may live on.

In C++

Composition

```
class Bar {
};

class Foo {
Private:
    Bar bar1;
}

void main {

    Foo foo1;
}
```

Constructors of the sub-objects are always executed before the constructors of the master class

Foo own **Bar** object and responsible for **Bar** lifetime. When **Foo** dies, so does **Bar**

Aggregation

```
class Bar {
};
class Foo {
Private:
    Bar* bar1;
Public:
    Foo(*Bar X)
    {
        bar1=X;
    }
}

void main {
    Bar a
    Foo foo1(&a);
}
```

Foo has an object which it borrowed from someone else. When **Foo** dies, **Bar** may live on.

In C++

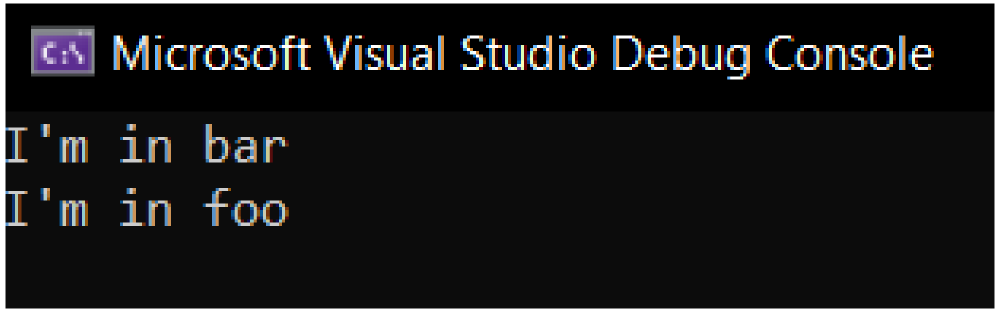
Composition

```
class bar
{
public:
    bar()
    {
        cout << "I'm in bar" << endl;
    }
};

class foo
{
    bar b;
public:
    foo()
    {
        cout << "I'm in foo" << endl;
    }
};
```

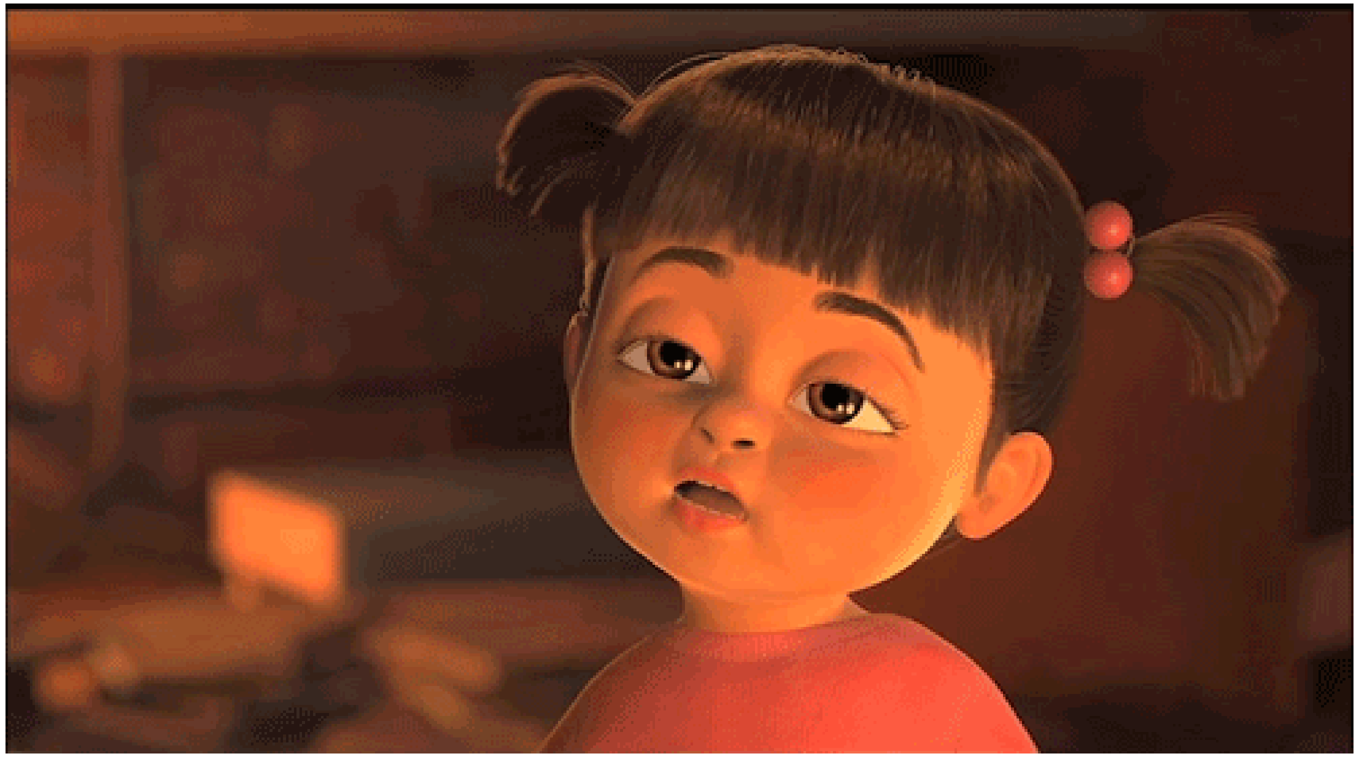
```
int main()
{

    foo f;
}
```



```
I'm in bar
I'm in foo
```

Thanks a lot



If you are taking a Nap, **wake up**.......Lecture Over