BTH001
Object Oriented Programming
Lesson 02
Relationship

Relationship

- Objects often relates to other objects – they **associate** to other objects. This is expressed on class level.
- *borrows, owns, use, has, consist-of, …* describes associations
- Inheritance – a relationsship on classtype level (later)

Composition and Aggregation

Are special kinds of associations

Composition:

A university has/consists-of (owns) departments – the departments will not exist without the university

A game has (owns) a character and a ball – the character and the ball will not exist without the game

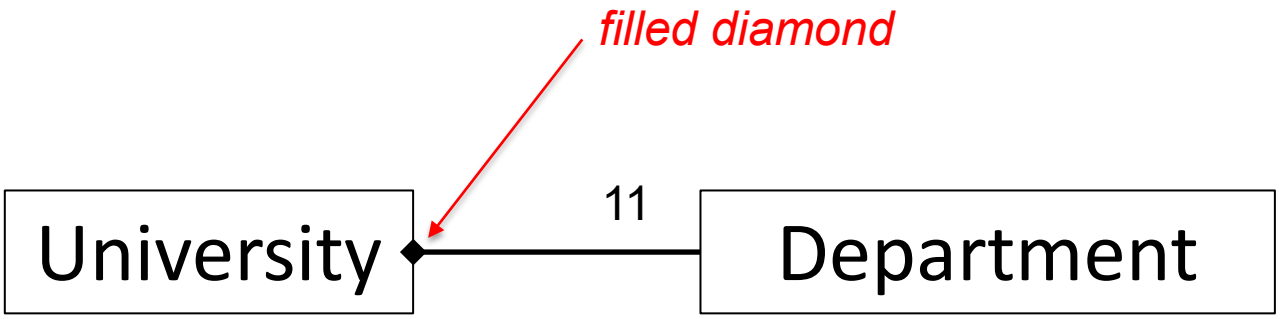
Aggregation:

A department has professors – the professors will exist even if the department is shutdown

A character has a ball (borrowed from the game) – the ball will exist even if the character isn´t alive

Composition

Class diagram describing a university has 11 departments



The Department class

```
class Department
{
private:
     string name;
     int nrOfEmployees;
     ….
public:
       int getNrOfEmployees();
       …..
};
```

The Department class is not aware of the University class

The University class

```
class University
{
private:
        string name;
        Department departments[11];
        ….
public:
        int totalNrEmployees();


};
```

A university has 11 departments

Communication

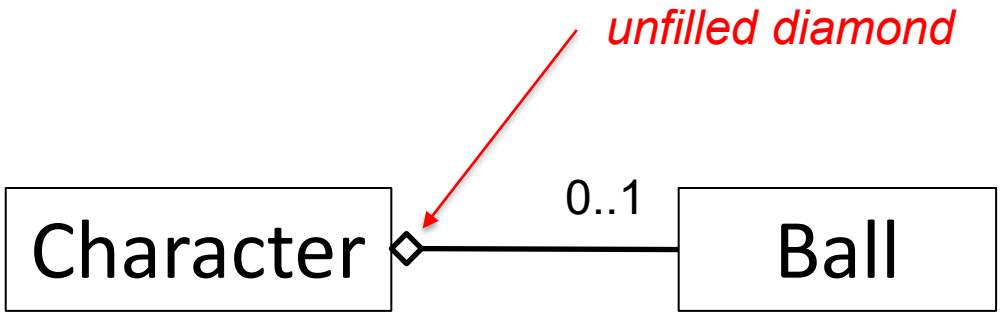
When a call of totalNrOfEmployees() is made for an University object, it will communicate with all Department objects to be able to calculate the total number of employees.

Communication continued

```
int University::totalNrEmployees() const
{
   int total = 0;
   for (int i=0; i<11; i++)
   {
       total += departments[i].getNrOfEmployees();
   }
   return total;
}
```

Aggregation

Class diagram describing a character has (borrows) a Ball



The Character class

- The Character class has a member variable that makes it possible to ”hold” a Ball object

- The Character object can communicate with the Ball object

The Character class

```
class Character
{
private:
      int nrOfLives;
      float health;
      Ball* ballPtr;
      ….
public:
      void receiveBall(Ball* aBall);
      void releaseBall();
      void moveBall();
      ….

};
```

”Someone else” (the Game) is responsible for the Ball object (creating, handling and deleting). A Character object has access to (not necessarily all the time) a Ball object (the pointer named ballPtr) which implies that the character is able to communicate with the ball if it has access to it.

Communication

```
void Character::receiveBall(Ball* aBall)
{
   this->ballPtr = aBall;
}
void Character::releaseBall()
{
   this->ballPtr = nullptr;
}
void Character::moveBall()
{
   if (this->ballPtr != nullptr)
       this->ballPtr->move();
}
```

Composition vs Aggregation

An university has 11 departments (composition)

A character has a ball (borrowed from the game) (aggregation)

The difference between aggregation and composition is that

- the departments will **not** exist without the university

but

- the ball will exist without the character

Another example