Motivation: Two or more classes with similar features.

**Before:**

class Employee

{

public string Name { get; private set; }

public string ID { get; private set; }

public int AnnualCost { get; private set; }

public Employee(string name, string id, int cost)

{

Name = name;

ID = id;

AnnualCost = cost;

}

}

class Department

{

public string Name { get; private set; }

public IEnumerable<Employee> Staff { get; private set; }

public int TotalAnnualCost

{

get { return Staff.Sum(s => s.AnnualCost); }

}

public int HeadCount

{

get { return Staff.Count(); }

}

public Department(string name)

{

Name = name;

Staff = new List<Employee>();

}

public void AddStaff(Employee employee)

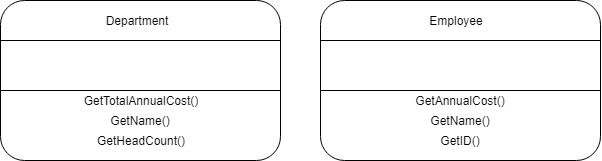
{

((IList<Employee>)Staff).Add(employee);

}

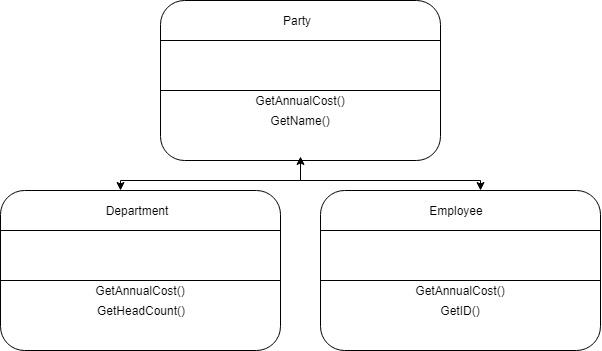
}

**Before Diagram:**



**Mechanics:** In this example there are ID, Name and Annual cost in **Employee** class. In **Department** class it has also a Name, list of employees of the department staff, totalAnnualCost and some other things like Department Head Count, functionality to add employees. So, the common features are name and the constructor logic that sets the name. Here TotalAnnualCost and AnnualCost are the same things. So, I am going to move these things to superclass name **Party** class. And make the **Employee** and **Department** as child class.

**After Diagram:**



**After:**

/// <summary>

/// SuperClass

/// </summary>

abstract class Party

{

public string Name { get; private set; }

public abstract int AnnualCost { get; }

protected Party(string name)

{

Name = name;

}

}

class Employee : Party

{

public string ID { get; private set; }

private int \_annualCost;

public override int AnnualCost

{

get { return \_annualCost; }

}

/// <summary>

/// Constructor

/// </summary>

/// <param name="name"></param>

/// <param name="id"></param>

/// <param name="cost"></param>

public Employee(string name, string id, int cost)

: base(name)

{

ID = id;

\_annualCost = cost;

}

}

class Department : Party

{

public IEnumerable<Party> Parties { get; private set; }

public override int AnnualCost

{

get { return Parties.Sum(s => s.AnnualCost); }

}

/// <summary>

/// Count Employee

/// </summary>

public int HeadCount

{

get { return Parties.Count(); }

}

/// <summary>

/// Constructor

/// </summary>

/// <param name="name"></param>

public Department(string name)

: base(name)

{

Parties = new List<Party>();

}

/// <summary>

/// Add Staff to the Party

/// </summary>

/// <param name="party"></param>

public void AddParty(Party party)

{

((IList<Party>)Parties).Add(party);

}

}