**Introduction**

Object casting consists of converting an object from one type to another. Traditionally, in computer programming, the casting is done between a variable of one class with a variable declared from a class derived from the first class. Here is an example:

using System;

public interface ICounter<T>

{

int Count { get; }

T Get(int index);

}

public interface IPersons<T> : ICounter<T>

{

void Add(T item);

}

public class People<T> : IPersons<T>

{

private int size;

private T[] persons;

public People()

{

size = 0;

persons = new T[10];

}

public int Count { get { return size; } }

public void Add(T pers)

{

persons[size] = pers;

size++;

}

public T Get(int index) { return persons[index]; }

}

// A class named Employee

public class Employee

{

public long EmployeeNumber { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public double HourlySalary { get; set; }

public Employee(long number = 0, string fName = "John",

string lName = "Doe", double salary = 12.05D)

{

EmployeeNumber = number;

FirstName = fName;

LastName = lName;

HourlySalary = salary;

}

public override string ToString()

{

base.ToString();

return string.Format("================================\n" +

"Employee Record\n" +

"--------------------------------\n" +

"Employee #: {0}\nFirst Name: {1}\n" +

"Last Name: {2}\nHourly Salary: {3}",

EmployeeNumber, FirstName,

LastName, HourlySalary);

}

}

// A class named Manager derived from Employee

public class Manager : Employee

{

public string Title { get; set; }

public override string ToString()

{

base.ToString();

return string.Format("================================\n" +

"Manager Information\n" +

"--------------------------------\n" +

"Employee #: {0}\nFirst Name: {1}\n" +

"Last Name: {2}\nHourly Salary: {3}\n" +

"Title: {4}\n", EmployeeNumber, FirstName,

LastName, HourlySalary, Title);

}

}

public class Exercise

{

public static int Main()

{

Manager mgr = new Manager();

mgr.EmployeeNumber = 249730;

mgr.FirstName = "Alex";

mgr.LastName = "Joyner";

mgr.HourlySalary = 32.85;

mgr.Title = "Regional Manager";

Console.WriteLine(mgr.ToString());

// Casting a Manager object to an Employee

Employee empl = (Employee)mgr;

Console.WriteLine(empl);

Console.WriteLine("================================");

return 0;

}

}

This would produce:

================================

Manager Information

--------------------------------

Employee #: 249730

First Name: Alex

Last Name: Joyner

Hourly Salary: 32.85

Title: Regional Manager

================================

Employee Record

--------------------------------

Employee #: 249730

First Name: Alex

Last Name: Joyner

Hourly Salary: 32.85

================================

Press any key to continue . . .

On the surface, casting appears as an easy operation. After all, it should be obvious to compare a son to his father. In computer programming, the operation is traditionally denied in most languages to compare a parent to a child. The following code will cause an error:

public class Exercise

{

public static int Main()

{

Employee empl = null;

empl = new Employee();

empl.EmployeeNumber = 253055;

empl.FirstName = "Joseph";

empl.LastName = "Denison";

empl.HourlySalary = 12.85;

Console.WriteLine(empl);

// Casting an Employee object to a Manager

Manager mgr = (Manager)empl;

Console.WriteLine(mgr);

Console.WriteLine("================================");

return 0;

}

}

This would produce:

================================

Employee Record

--------------------------------

Employee #: 253055

First Name: Joseph

Last Name: Denison

Hourly Salary: 12.85

================================

Unhandled Exception: System.InvalidCastException: Unable to cast object of type

'Employee' to type 'Manager'.

at Exercise.Main() in C:\Temporary Projects\Exercise.cs:line 77

Press any key to continue . . .

The error is based on the fact that a parent class has no way of identifying what child it has, because at the time a class is created, it is not aware of child classes that would be derived from it. This operation is still not directly allowed.

**Covariance**

Remember that you can create a list of objects using a generic collection class. Here are examples:

using System;

public interface ICounter<T>

{

int Count { get; }

T Get(int index);

}

public interface IPersons<T> : ICounter<T>

{

void Add(T item);

}

public class People<T> : IPersons<T>

{

private int size;

private T[] persons;

public People()

{

size = 0;

persons = new T[10];

}

public int Count { get { return size; } }

public void Add(T pers)

{

persons[size] = pers;

size++;

}

public T Get(int index) { return persons[index]; }

}

// A class named Employee

public class Employee

{

public long EmployeeNumber { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public double HourlySalary { get; set; }

public Employee(long number = 0, string fName = "John",

string lName = "Doe", double salary = 12.05D)

{

EmployeeNumber = number;

FirstName = fName;

LastName = lName;

HourlySalary = salary;

}

public override string ToString()

{

base.ToString();

return string.Format("================================\n" +

"Employee Record\n" +

"--------------------------------\n" +

"Employee #: {0}\nFirst Name: {1}\n" +

"Last Name: {2}\nHourly Salary: {3}",

EmployeeNumber, FirstName,

LastName, HourlySalary);

}

}

// A class named Manager derived from Employee

public class Manager : Employee

{

public string Title { get; set; }

public override string ToString()

{

base.ToString();

return string.Format("================================\n" +

"Manager Information\n" +

"--------------------------------\n" +

"Employee #: {0}\nFirst Name: {1}\n" +

"Last Name: {2}\nHourly Salary: {3}\n" +

"Title: {4}\n", EmployeeNumber, FirstName,

LastName, HourlySalary, Title);

}

}

public class Exercise

{

public static int Main()

{

IPersons<Employee> staff = new People<Employee>();

Employee empl = null;

empl = new Employee();

empl.EmployeeNumber = 253055;

empl.FirstName = "Joseph";

empl.LastName = "Denison";

empl.HourlySalary = 12.85;

staff.Add(empl);

empl = new Employee();

empl.EmployeeNumber = 204085;

empl.FirstName = "Raymond";

empl.LastName = "Ramirez";

empl.HourlySalary = 9.95;

staff.Add(empl);

empl = new Employee();

empl.EmployeeNumber = 970044;

empl.FirstName = "James";

empl.LastName = "Macies";

empl.HourlySalary = 14.25;

staff.Add(empl);

Console.WriteLine("=--= Employees =--=");

for(int i = 0; i < staff.Count; i++)

Console.WriteLine(staff.Get(i));

IPersons<Manager> leaders = new People<Manager>();

Manager mgr = null;

mgr = new Manager();

mgr.EmployeeNumber = 249730;

mgr.FirstName = "Alex";

mgr.LastName = "Joyner";

mgr.HourlySalary = 32.85;

mgr.Title = "Regional Manager";

leaders.Add(mgr);

mgr = new Manager();

mgr.EmployeeNumber = 579047;

mgr.FirstName = "Robert";

mgr.LastName = "Farley";

mgr.HourlySalary = 28.55;

mgr.Title = "Shift Supervisor";

leaders.Add(mgr);

Console.WriteLine("=--= Managers =--=");

for (int i = 0; i < leaders.Count; i++)

Console.WriteLine(leaders.Get(i));

Console.WriteLine("================================");

return 0;

}

}

This would produce:

=--= Employees =--=

================================

Employee Record

--------------------------------

Employee #: 253055

First Name: Joseph

Last Name: Denison

Hourly Salary: 12.85

================================

Employee Record

--------------------------------

Employee #: 204085

First Name: Raymond

Last Name: Ramirez

Hourly Salary: 9.95

================================

Employee Record

--------------------------------

Employee #: 970044

First Name: James

Last Name: Macies

Hourly Salary: 14.25

=--= Managers =--=

================================

Manager Information

--------------------------------

Employee #: 249730

First Name: Alex

Last Name: Joyner

Hourly Salary: 32.85

Title: Regional Manager

================================

Manager Information

--------------------------------

Employee #: 579047

First Name: Robert

Last Name: Farley

Hourly Salary: 28.55

Title: Shift Supervisor

================================

Press any key to continue . . .

Since we have already learned how to return a generic interface from a method, if you want, instead of creating the records locally, you can define a method that generates them, and then return an interface from that method. Here are examples:

using System;

*. . . No Change*

public class Exercise

{

private static IPersons<Employee> GetEmployees()

{

IPersons<Employee> staff = new People<Employee>();

Employee empl = null;

empl = new Employee();

empl.EmployeeNumber = 253055;

empl.FirstName = "Joseph";

empl.LastName = "Denison";

empl.HourlySalary = 12.85;

staff.Add(empl);

empl = new Employee();

empl.EmployeeNumber = 204085;

empl.FirstName = "Raymond";

empl.LastName = "Ramirez";

empl.HourlySalary = 9.95;

staff.Add(empl);

empl = new Employee();

empl.EmployeeNumber = 970044;

empl.FirstName = "James";

empl.LastName = "Macies";

empl.HourlySalary = 14.25;

staff.Add(empl);

return staff;

}

private static IPersons<Manager> GetManagers()

{

IPersons<Manager> leaders = new People<Manager>();

Manager mgr = null;

mgr = new Manager();

mgr.EmployeeNumber = 249730;

mgr.FirstName = "Alex";

mgr.LastName = "Joyner";

mgr.HourlySalary = 32.85;

mgr.Title = "Regional Manager";

leaders.Add(mgr);

mgr = new Manager();

mgr.EmployeeNumber = 579047;

mgr.FirstName = "Robert";

mgr.LastName = "Farley";

mgr.HourlySalary = 28.55;

mgr.Title = "Shift Supervisor";

leaders.Add(mgr);

return leaders;

}

public static int Main()

{

IPersons<Employee> staff = GetEmployees();

Console.WriteLine("=--= Employees =--=");

for (int i = 0; i < staff.Count; i++)

Console.WriteLine(staff.Get(i));

IPersons<Manager> leaders = GetManagers();

Console.WriteLine("=--= Managers =--=");

for (int i = 0; i < leaders.Count; i++)

Console.WriteLine(leaders.Get(i));

Console.WriteLine("================================");

return 0;

}

}

When creating objects from your generic collection class, you may be interested in casting one list from one type to a list of another type. Consider the following example:

public class Exercise

{

private static IPersons<Employee> GetEmployees()

{

IPersons<Employee> staff = new People<Employee>();

*. . . No Change*

return staff;

}

private static IPersons<Manager> GetManagers()

{

IPersons<Manager> leaders = new People<Manager>();

*. . . No Change*

return leaders;

}

public static int Main()

{

ICounter<Manager> leaders = GetManagers();

ICounter<Employee> staff = (ICounter<Employee>)leaders;

Console.WriteLine("=--= Employees =--=");

for (int i = 0; i < staff.Count; i++)

Console.WriteLine(staff.Get(i));

Console.WriteLine("=--= Managers =--=");

for (int i = 0; i < leaders.Count; i++)

Console.WriteLine(leaders.Get(i));

Console.WriteLine("================================");

return 0;

}

}

This would produce the following error:

Unhandled Exception: System.InvalidCastException: Unable to cast object of type

'People`1[Manager]' to type 'ICounter`1[Employee]'.

at Exercise.Main() in C:\Exercise1\Exercise.cs:line 143

Press any key to continue . . .

In realty, you can manage casting only among compatible objects. That is, you can perform casting from objects of classes that implement the same interface but there are rules you must follow.

If you want value casting to be performed among the objects of its implementers, when creating the generic interface, precede the type parameter with the **out** keyword. Here is an example:

public interface ICounter<out T>

{

int Count { get; }

T Get(int index);

}

public interface IPersons<T> : ICounter<T>

{

void Add(T item);

}

 After doing this, the casting would work. Here is an example:

public class Exercise

{

private static IPersons<Employee> GetEmployees()

{

IPersons<Employee> staff = new People<Employee>();

*. . . No Change*

return staff;

}

private static IPersons<Manager> GetManagers()

{

IPersons<Manager> leaders = new People<Manager>();

*. . . No Change*

return leaders;

}

public static int Main()

{

ICounter<Manager> leaders = GetManagers();

ICounter<Employee> staff = (ICounter<Employee>)leaders;

Console.WriteLine("=--= Employees =--=");

for (int i = 0; i < staff.Count; i++)

Console.WriteLine(staff.Get(i));

Console.WriteLine("=--= Managers =--=");

for (int i = 0; i < leaders.Count; i++)

Console.WriteLine(leaders.Get(i));

Console.WriteLine("================================");

return 0;

}

}

In this cast, we performed the cast by preceding the the variable with the name of the interface and the other class. Actually, you can omit it:

using System;

public interface ICounter<out T>

{

int Count { get; }

T Get(int index);

}

public interface IPersons<T> : ICounter<T>

{

void Add(T item);

}

public class People<T> : IPersons<T>

{

private int size;

private T[] persons;

public People()

{

size = 0;

persons = new T[10];

}

public int Count { get { return size; } }

public void Add(T pers)

{

persons[size] = pers;

size++;

}

public T Get(int index) { return persons[index]; }

}

public class Employee

{

public long EmployeeNumber { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public double HourlySalary { get; set; }

public Employee(long number = 0, string fName = "John",

string lName = "Doe", double salary = 12.05D)

{

EmployeeNumber = number;

FirstName = fName;

LastName = lName;

HourlySalary = salary;

}

public override string ToString()

{

base.ToString();

return string.Format("================================\n" +

"Employee Record\n" +

"--------------------------------\n" +

"Employee #: {0}\nFirst Name: {1}\n" +

"Last Name: {2}\nHourly Salary: {3}",

EmployeeNumber, FirstName,

LastName, HourlySalary);

}

}

// A class named Manager derived from Employee

public class Manager : Employee

{

public string Title { get; set; }

public override string ToString()

{

base.ToString();

return string.Format("================================\n" +

"Manager Information\n" +

"--------------------------------\n" +

"Employee #: {0}\nFirst Name: {1}\n" +

"Last Name: {2}\nHourly Salary: {3}\n" +

"Title: {4}\n", EmployeeNumber, FirstName,

LastName, HourlySalary, Title);

}

}

public class Exercise

{

private static IPersons<Employee> GetEmployees()

{

IPersons<Employee> staff = new People<Employee>();

Employee empl = null;

empl = new Employee();

empl.EmployeeNumber = 253055;

empl.FirstName = "Joseph";

empl.LastName = "Denison";

empl.HourlySalary = 12.85;

staff.Add(empl);

empl = new Employee();

empl.EmployeeNumber = 204085;

empl.FirstName = "Raymond";

empl.LastName = "Ramirez";

empl.HourlySalary = 9.95;

staff.Add(empl);

empl = new Employee();

empl.EmployeeNumber = 970044;

empl.FirstName = "James";

empl.LastName = "Macies";

empl.HourlySalary = 14.25;

staff.Add(empl);

return staff;

}

private static IPersons<Manager> GetManagers()

{

IPersons<Manager> leaders = new People<Manager>();

Manager mgr = null;

mgr = new Manager();

mgr.EmployeeNumber = 249730;

mgr.FirstName = "Alex";

mgr.LastName = "Joyner";

mgr.HourlySalary = 32.85;

mgr.Title = "Regional Manager";

leaders.Add(mgr);

mgr = new Manager();

mgr.EmployeeNumber = 579047;

mgr.FirstName = "Robert";

mgr.LastName = "Farley";

mgr.HourlySalary = 28.55;

mgr.Title = "Shift Supervisor";

leaders.Add(mgr);

return leaders;

}

public static int Main()

{

ICounter<Manager> leaders = GetManagers();

ICounter<Employee> staff = leaders;

Console.WriteLine("=--= Employees =--=");

for (int i = 0; i < staff.Count; i++)

Console.WriteLine(staff.Get(i));

Console.WriteLine("=--= Managers =--=");

for (int i = 0; i < leaders.Count; i++)

Console.WriteLine(leaders.Get(i));

Console.WriteLine("================================");

return 0;

}

}

This technique of managing value casting is called covariance.

**Introduction to Built-In Covariance Interfaces**

Covariance is a technique used in collection classes and makes it possible to cast one collection indirectly to another, using their common parent interface. To assist you with covariance, the .NET Framework provides some already created classes you can use in your applications. They are:

* **IEnumerable**: This interface is primarily used to enumerate the members of a collection. Of course, it must be a generic collection class that implements this interface. An example would be the **List<>** class. Here is an example:
* using System;
* using System.Collections.Generic;
* public class Employee
* {
* public long EmployeeNumber { get; set; }
* public string FirstName { get; set; }
* public string LastName { get; set; }
* public double HourlySalary { get; set; }
* public Employee(long number = 0, string fName = "John",
* string lName = "Doe", double salary = 12.05D)
* {
* EmployeeNumber = number;
* FirstName = fName;
* LastName = lName;
* HourlySalary = salary;
* }
* public override string ToString()
* {
* base.ToString();
* return string.Format("================================\n" +
* "Employee Record\n" +
* "--------------------------------\n" +
* "Employee #: {0}\nFirst Name: {1}\n" +
* "Last Name: {2}\nHourly Salary: {3}",
* EmployeeNumber, FirstName,
* LastName, HourlySalary);
* }
* }
* // A class named Manager derived from Employee
* public class Manager : Employee
* {
* public string Title { get; set; }
* public override string ToString()
* {
* base.ToString();
* return string.Format("================================\n" +
* "Manager Information\n" +
* "--------------------------------\n" +
* "Employee #: {0}\nFirst Name: {1}\n" +
* "Last Name: {2}\nHourly Salary: {3}\n" +
* "Title: {4}\n", EmployeeNumber, FirstName,
* LastName, HourlySalary, Title);
* }
* }
* public class Exercise
* {
* private static List<Employee> GetEmployees()
* {
* List<Employee> staff = new List<Employee>();
* Employee empl = null;
* empl = new Employee();
* empl.EmployeeNumber = 253055;
* empl.FirstName = "Joseph";
* empl.LastName = "Denison";
* empl.HourlySalary = 12.85;
* staff.Add(empl);
* empl = new Employee();
* empl.EmployeeNumber = 204085;
* empl.FirstName = "Raymond";
* empl.LastName = "Ramirez";
* empl.HourlySalary = 9.95;
* staff.Add(empl);
* empl = new Employee();
* empl.EmployeeNumber = 970044;
* empl.FirstName = "James";
* empl.LastName = "Macies";
* empl.HourlySalary = 14.25;
* staff.Add(empl);
* return staff;
* }
* private static List<Manager> GetManagers()
* {
* List<Manager> leaders = new List<Manager>();
* Manager mgr = null;
* mgr = new Manager();
* mgr.EmployeeNumber = 249730;
* mgr.FirstName = "Alex";
* mgr.LastName = "Joyner";
* mgr.HourlySalary = 32.85;
* mgr.Title = "Regional Manager";
* leaders.Add(mgr);
* mgr = new Manager();
* mgr.EmployeeNumber = 579047;
* mgr.FirstName = "Robert";
* mgr.LastName = "Farley";
* mgr.HourlySalary = 28.55;
* mgr.Title = "Shift Supervisor";
* leaders.Add(mgr);
* return leaders;
* }
* public static int Main()
* {
* IEnumerable<Manager> managers = GetManagers();
* IEnumerable<Employee> employees = managers;
* foreach (Employee empl in employees)
* Console.WriteLine(empl);
* foreach (Manager man in managers)
* Console.WriteLine(man);
* Console.WriteLine("================================");
* return 0;
* }

}

This would produce:

================================

Manager Information

--------------------------------

Employee #: 249730

First Name: Alex

Last Name: Joyner

Hourly Salary: 32.85

Title: Regional Manager

================================

Manager Information

--------------------------------

Employee #: 579047

First Name: Robert

Last Name: Farley

Hourly Salary: 28.55

Title: Shift Supervisor

================================

Manager Information

--------------------------------

Employee #: 249730

First Name: Alex

Last Name: Joyner

Hourly Salary: 32.85

Title: Regional Manager

================================

Manager Information

--------------------------------

Employee #: 579047

First Name: Robert

Last Name: Farley

Hourly Salary: 28.55

Title: Shift Supervisor

================================

Press any key to continue . . .

* **IEnumerator**: This interface is used to iterate through a collection that uses memory and must release it after use
* **IQueryable**: This interface is used to perform queries in a LINQ collection
* **IGrouping**: This interface also is used in LINQ queries to group members that respond to a criterion