Ради простоты приведенных выше примеров в коллекции, как правило, сохранялись

объекты встроенных типов, в том числе int, string и char. Но ведь в коллекции

можно хранить не только объекты встроенных типов. Достоинство коллекций в том и

состоит, что в них допускается хранить объекты любого типа, включая объекты определяемых

пользователем классов.

Рассмотрим сначала простой пример применения класса необобщенной коллекции

ArrayList для хранения информации о товарных запасах. В этом классе инкапсулируется

класс Inventory.

(***glava25\_26***)

class Inventory

{

string name;

double cost;

int onhand;

public Inventory(string n, double c, int h)

{

name = n;

cost = c;

onhand = h;

}

public override string ToString()

{

return

String.Format("{0, -10}Cost: {1,6:C} Quantity: {2}",

name, cost, onhand);

}

}

class InventoryList

{

static void Main()

{

ArrayList inv = new ArrayList();

inv.Add(new Inventory("Pliers", 5.95, 3));

inv.Add(new Inventory("Screws", 8.29, 2));

inv.Add(new Inventory("Hammers", 3.50, 4));

inv.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("List of goods:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

}

}

Обратите внимание на то, что в данном примере программы не потребовалось никаких

специальных действий для сохранения в коллекции объектов типа Inventory.

Благодаря тому что все типы наследуют от класса object, в необобщенной коллекции

можно хранить объекты любого типа. Именно поэтому в необобщенной коллекции

нетрудно сохранить объекты определяемых пользователем классов. Безусловно, это

также означает, что такая коллекция не типизирована.

Для того чтобы сохранить объекты определяемых пользователем классов в типизированной

коллекции, придется воспользоваться классами обобщенных коллекций.

В качестве примера ниже приведен измененный вариант программы из предыдущего

примера. В этом варианте используется класс обобщенной коллекции List<T>, а результат

получается таким же, как и прежде.

(***glava25\_26.1***)

class InventoryList

{

static void Main()

{

List<Inventory> inv = new List<Inventory>();

inv.Add(new Inventory("Pliers", 5.95, 3));

inv.Add(new Inventory("Screws", 8.29, 2));

inv.Add(new Inventory("Hammers", 3.50, 4));

inv.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("List of goods:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

}

}

Данный пример отличается от предыдущего лишь передачей типа Inventory в

качестве аргумента типа конструктору класса List<T>. А в остальном оба примера

рассматриваемой здесь программы практически одинаковы. Это, по существу, означает,

что для применения обобщенной коллекции не требуется никаких особых усилий,

но при сохранении в такой коллекции объекта конкретного типа строго соблюдается

типовая безопасность.

Тем не менее для обоих примеров рассматриваемой здесь программы характерна

еще одна особенность: они довольно кратки. Если учесть, что для организации динамического

массива, где можно хранить, извлекать и обрабатывать данные товарных

запасов, потребуется не менее 40 строк кода, то преимущества коллекций сразу же

становятся очевидными. Нетрудно догадаться, что рассматриваемая здесь программа

получится длиннее в несколько раз, если попытаться закодировать все эти функции

коллекции вручную. Коллекции предлагают готовые решения самых разных задач

программирования, и поэтому их следует использовать при всяком удобном случае.

У рассматриваемой здесь программы имеется все же один не совсем очевидный

недостаток: коллекция не подлежит сортировке. Дело в том, что в классах ArrayList

и List<T> отсутствуют средства для сравнения двух объектов типа Inventory. Но из

этого положения имеются два выхода. Во-первых, в классе Inventory можно реализовать

интерфейс IComparable, в котором определяется метод сравнения объектов данного

класса. И во-вторых, для целей сравнения можно указать объект типа IComparer.

Оба подхода рассматриваются далее по очереди.

**Реализация интерфейса Icomparable**

Если требуется отсортировать коллекцию, состоящую из объектов определяемого

пользователем класса, при условии, что они не сохраняются в коллекции класса

SortedList, где элементы располагаются в отсортированном порядке, то в такой

коллекции должен быть известен способ сортировки содержащихся в ней объектов.

С этой целью можно, в частности, реализовать интерфейс IComparable для объектов

сохраняемого типа. Интерфейс IComparable доступен в двух формах: обобщенной и

необобщенной. Несмотря на сходство применения обеих форм данного интерфейса,

между ними имеются некоторые, хотя и небольшие, отличия, рассматриваемые ниже.

**Реализация интерфейса IComparable для необобщенных коллекций**

Если требуется отсортировать объекты, хранящиеся в необобщенной коллекции,

то для этой цели придется реализовать необобщенный вариант интерфейса

IComparable. В этом варианте данного интерфейса определяется только один метод

CompareTo(), который определяет порядок выполнения самого сравнения. Ниже

приведена общая форма объявления метода CompareTo().

int CompareTo(object obj)

В методе CompareTo() вызывающий объект сравнивается с объектом *obj.* Для сортировки

объектов по нарастающей конкретная реализация данного метода должна

возвращать нулевое значение, если значения сравниваемых объектов равны; положительное

— если значение вызывающего объекта больше, чем у объекта *obj*; и отрицательное

— если значение вызывающего объекта меньше, чем у объекта *obj. А* для

сортировки по убывающей можно обратить результат сравнения объектов. Если же

тип объекта *obj* не подходит для сравнения с вызывающим объектом, то в методе

CompareTo() может быть сгенерировано исключение ArgumentException.

В приведенном ниже примере программы демонстрируется конкретная реализация

интерфейса IComparable. В этой программе интерфейс IComparable вводится

в класс Inventory, разработанный в двух последних примерах из предыдущего раздела.

В классе Inventory реализуется метод CompareTo() для сравнения полей name

объектов данного класса, что дает возможность отсортировать товарные запасы по наименованию.

Как показано в данном примере программы, коллекция объектов класса

Inventory подлежит сортировке благодаря реализации интерфейса IComparable в

этом классе.

(***glava25\_26.2***)

class Inventory : IComparable

{

string name;

double cost;

int onhand;

public Inventory(string n, double c, int h)

{

name = n;

cost = c;

onhand = h;

}

public override string ToString()

{

return

String.Format("{0, -10}Cost: {1,6:C} Quantity: {2}",

name, cost, onhand);

}

//interface IComparable

public int CompareTo(object obj)

{

Inventory b;

b = (Inventory)obj;

return name.CompareTo(b.name);

}

}

class IComparableDemo

{

static void Main()

{

ArrayList inv = new ArrayList();

inv.Add(new Inventory("Pliers", 5.95, 3));

inv.Add(new Inventory("Screws", 8.29, 2));

inv.Add(new Inventory("Hammers", 3.50, 4));

inv.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("List of goods:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

//sort list

inv.Sort();

Console.WriteLine("List of goods sorted:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

}

}

**Реализация интерфейса IComparable для обобщенных коллекций**

Если требуется отсортировать объекты, хранящиеся в обобщенной коллекции,

то для этой цели придется реализовать обобщенный вариант интерфейса

IComparable<T>. В этом варианте интерфейса IComparable определяется приведенная

ниже обобщенная форма метода CompareTo().

int CompareTo(Т other)

В методе CompareTo() вызывающий объект сравнивается с другим объектом

*other.* Для сортировки объектов по нарастающей конкретная реализация данного

метода должна возвращать нулевое значение, если значения сравниваемых объектов

равны; положительное — если значение вызывающего объекта больше, чем у объекта

другого *other*; и отрицательное — если значение вызывающего объекта меньше, чем

у другого объекта *other. А* для сортировки по убывающей можно обратить результат

сравнения объектов. При реализации обобщенного интерфейса IComparable<T> имя

типа реализующего класса обычно передается в качестве аргумента типа.

Приведенный ниже пример программы является вариантом предыдущего примера,

измененным с целью реализовать и использовать обобщенный интерфейс

IComparable<Т>. Обратите внимание на применение класса обобщенной коллекции

List<T> вместо класса необобщенной коллекции ArrayList.

(***glava25\_26.3***)

class Inventory : IComparable<Inventory>

{

string name;

double cost;

int onhand;

public Inventory(string n, double c, int h)

{

name = n;

cost = c;

onhand = h;

}

public override string ToString()

{

return

String.Format("{0, -10}Cost: {1,6:C} Quantity: {2}",

name, cost, onhand);

}

//interface IComparable

public int CompareTo(Inventory obj)

{

return name.CompareTo(obj.name);

}

}

class IComparableDemo

{

static void Main()

{

List<Inventory> inv = new List<Inventory>();

inv.Add(new Inventory("Pliers", 5.95, 3));

inv.Add(new Inventory("Screws", 8.29, 2));

inv.Add(new Inventory("Hammers", 3.50, 4));

inv.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("List of goods:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

//sort list

inv.Sort();

Console.WriteLine("List of goods sorted:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

}

}

**Применение интерфейса Icomparer**

Для сортировки объектов определяемых пользователем классов зачастую проще

всего реализовать в этих классах интерфейс IComparable. Тем не менее данную задачу

можно решить и с помощью интерфейса IComparer. Для этой цели необходимо сначала

создать класс, реализующий интерфейс IComparer, а затем указать объект этого

класса, когда потребуется сравнение.

Интерфейс IComparer существует в двух формах: обобщенной и необобщенной.

Несмотря на сходство применения обеих форм данного интерфейса, между ними имеются

некоторые, хотя и небольшие, отличия, рассматриваемые ниже.

**Применение необобщенного интерфейса Icomparer**

В необобщенном интерфейсе IComparer определяется только один метод,

Compare().

int Compare(object x, object y)

В методе Compare() сравниваются объекты x и у. Для сортировки объектов по нарастающей

конкретная реализация данного метода должна возвращать нулевое значение,

если значения сравниваемых объектов равны; положительное — если значение

объекта х больше, чем у объекта у; и отрицательное — если значение объекта х меньше,

чем у объекта *у.* А для сортировки по убывающей можно обратить результат сравнения

объектов. Если же тип объекта х не подходит для сравнения с объектом у, то в

методе CompareTo() может быть сгенерировано исключение ArgumentException.

Объект типа IComparer может быть указан при конструировании объекта класса

SortedList, при вызове метода ArrayList.Sort(IComparer), а также в ряде других

мест в классах коллекций. Главное преимущество применения интерфейса IComparer

заключается в том, что сортировке подлежат объекты тех классов, в которых интерфейс

IComparable не реализуется.

Приведенный ниже пример программы является вариантом рассматривавшегося

ранее необобщенного примера программы учета товарных запасов, переделанного с

целью воспользоваться интерфейсом IComparer для сортировки перечня товарных запасов.

В этом варианте программы сначала создается класс CompInv, в котором реализуется

интерфейс IComparer и сравниваются два объекта класса Inventory. А затем

объект класса CompInv указывается в вызове метода Sort() для сортировки перечня

товарных запасов.

(***glava25\_26.4***)

class CompInv : IComparer

{

//realize interface

public int Compare(object x, object y)

{

Inventory a, b;

a = (Inventory)x;

b = (Inventory)y;

return string.Compare(a.name, b.name, StringComparison.Ordinal);

}

}

class Inventory

{

public string name;

double cost;

int onhand;

public Inventory(string n, double c, int h)

{

name = n;

cost = c;

onhand = h;

}

public override string ToString()

{

return

String.Format("{0, -10}Cost: {1,6:C} Quantity: {2}",

name, cost, onhand);

}

}

class IComparableDemo

{

static void Main()

{

CompInv comp = new CompInv();

ArrayList inv = new ArrayList();

inv.Add(new Inventory("Pliers", 5.95, 3));

inv.Add(new Inventory("Screws", 8.29, 2));

inv.Add(new Inventory("Hammers", 3.50, 4));

inv.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("List of goods:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

//sort list

inv.Sort(comp);

Console.WriteLine("List of goods sorted:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

}

}

**Применение обобщенного интерфейса IComparer<T>**

Интерфейс IComparer<T> является обобщенным вариантом интерфейса IComparer.

В нем определяется приведенный ниже обобщенный вариант метода Compare().

int Compare(Т х, Т у)

В этом методе сравниваются объекты *х* и у и возвращается нулевое значение, если

значения сравниваемых объектов равны; положительное — если значение объекта *х*

больше, чем у объекта у; и отрицательное — если значение объекта *х* меньше, чем у

объекта у.

Ниже приведена обобщенная версия предыдущей программы учета товарных запасов,

в которой теперь используется интерфейс IComparer<T>. Она дает такой же

результат, как и необобщенная версия этой же программы.

(***glava25\_26.5***)

class CompInv<T> : IComparer<T> where T : Inventory

{

//realize interface

public int Compare(T x, T y)

{

return string.Compare(x.name, y.name, StringComparison.Ordinal);

}

}

class Inventory

{

public string name;

double cost;

int onhand;

public Inventory(string n, double c, int h)

{

name = n;

cost = c;

onhand = h;

}

public override string ToString()

{

return

String.Format("{0, -10}Cost: {1,6:C} Quantity: {2}",

name, cost, onhand);

}

}

class IComparableDemo

{

static void Main()

{

CompInv<Inventory> comp = new CompInv<Inventory>();

List<Inventory> inv = new List<Inventory>();

inv.Add(new Inventory("Pliers", 5.95, 3));

inv.Add(new Inventory("Screws", 8.29, 2));

inv.Add(new Inventory("Hammers", 3.50, 4));

inv.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("List of goods:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

//sort list

inv.Sort(comp);

Console.WriteLine("List of goods sorted:");

foreach (Inventory i in inv)

Console.WriteLine(" " + i);

Console.WriteLine();

}

}

**Применение класса StringComparer**

В простых примерах из этой главы указывать явно способ сравнения символьных

строк совсем не обязательно. Но это может потребоваться в тех случаях, когда строки

сохраняются в отсортированной коллекции или когда строки ищутся либо сортируются

в коллекции. Так, если строки должны быть отсортированы с учетом настроек

одной культурной среды, а затем их приходится искать с учетом настроек другой культурной

среды, то во избежание ошибок, вероятнее всего, потребуется указать способ

сравнения символьных строк. Аналогичная ситуация возникает и при хешировании

коллекции. Для подобных (и других) случаев в конструкторах классов некоторых коллекций

предусмотрена поддержка параметра типа IComparer. С целью явно указать

способ сравнения символьных строк этому параметру передается в качестве аргумента

экземпляр объекта класса StringComparer.

Класс StringComparer был подробно описан в главе 21 при рассмотрении вопросов

сортировки и поиска в массивах. В этом классе реализуются интерфейсы IComparer,

IComparer<String>, IEqualityComparer, а также IEqualityComparer<String>.

Следовательно, экземпляр объекта типа StringComparer может быть передан параметру

типа IComparer в качестве аргумента. В классе StringComparer определяется

несколько доступных только для чтения свойств, возвращающих экземпляр

объекта типа StringComparer, который поддерживает различные способы сравнения

символьных строк. Как пояснялось в главе 21, к числу этих свойств относятся

следующие: CurrentCulture, CurrentCultureIgnoreCase, InvariantCulture,

InvariantCultureIgnoreCase, Ordinal, а также OrdinalIgnoreCase. Все эти свойства

можно использовать для явного указания способа сравнения символьньгх строк.

В качестве примера ниже показано, как коллекция типа SortedList<TKey,

TValue> конструируется для хранения символьных строк, ключи которых сравниваются

порядковым способом.

SortedList<string, int> users =

new SortedList<string, int>(StringComparer.Ordinal);