Лабораторная работа № 6

Работа с коллекциями и интерфейс Comparator.

Цель работы: научиться использовать функционал Collection Framework на языке Java.

Справочная информация

В пакет java.util входит одна из самых эффективных подсистем Java - каркас коллекций Collections Framework. Этот каркас представляет собой сложную иерархию интерфейсов и классов, реализующих современную технологию управления группами объектов. Он заслуживает пристального внимания всех программирующих на Java. Начиная с версии JDK 9, пакет java.util входит в состав модуля java.base.

Каркас коллекций Collections Framework в Java стандартизирует способы управления группами объектов в прикладных программах. Коллекции не входили в исходную версию языка Java, но были внедрены в версии J2SE 1.2. До появления каркаса коллекций, предназначенного для хранения групп объектов и манипулирования ими, в Java предоставлялись такие специальные классы, как *Dictionary*, *Vector*, *Stack* и *Properties*. И хотя эти классы были достаточно удобны, им недоставало общей, объединяющей основы. Так, класс *Vector* отличался способом своего применения от класса *Properties*. Такой первоначальный специализированный подход не был рассчитан на дальнейшее расширение и адаптацию. Для разрешения этого и ряда других затруднений и были внедрены коллекции.

Каркас коллекций был разработан для достижения нескольких целей. Во-первых, он должен был обеспечивать высокую производительность. Реализация основных коллекций (динамических массивов, связных списков, деревьев и хеш-таблиц) отличается высокой эффективностью. Программировать один из таких "механизмов доступа к данным" вручную приходится крайне редко. Во-вторых, каркас должен был обеспечивать единообразное функционирование коллекций с высокой степенью взаимодействия. В-третьих, коллекции должны были допускать простое расширение и/или адаптацию. В этом отношении весь каркас коллекций построен на едином наборе стандартных интерфейсов. Некоторые стандартные реализации этих интерфейсов (например, в классах *LinkedList*, *HashSet* и *TreeSet*) можно использовать в исходном виде. Но при желании можно реализовать и свои коллекции. Для удобства программистов

предусмотрены различные реализации специального назначения, а также частичные реализации, которые облегчают создание собственных коллекций. И наконец, в каркас коллекций были внедрены механизмы интеграции стандартных массивов.

Алгоритмы составляют другую важную часть каркаса коллекций. Алгоритмы оперируют коллекциями и определены в виде статических методов в классе *Collections*. Таким образом, они доступны во всех коллекциях и не требуют реализации их собственной версии в каждом классе коллекции. Алгоритмы предоставляют стандартные средства для манипулирования коллекциями. Другим элементом, тесно связанным с каркасом коллекций, является интерфейс *Iterator*, определяющий итератор, который обеспечивает общий, стандартизованный способ поочередного доступа к элементам коллекций. Иными словами, итератор предоставляет способ перебора содержимого коллекций. А поскольку каждая коллекция предоставляет свой итератор, то элементы любого класса коллекций могут быть доступны с помощью методов, определенных в интерфейсе *Iterator*. Таким образом, код, перебирающий в цикле элементы множества, можно с минимальными изменениями применить, например, для перебора элементов списка. В версии JDK 8 внедрена другая разновидность итератора, называемая итератором- разделителем. Если коротко, то итераторы-разделители обеспечивают параллельную итерацию. Итераторы-разделители поддерживаются в интерфейсе *Spliterator* и ряде вложенных в него интерфейсов, которые, в свою очередь, поддерживают примитивные типы данных. В версии JDK 8 внедрены также интерфейсы итераторов, предназначенные для применения вместе с примитивными типами данных. К их числу относятся интерфейсы *PrimitiveIterator* и *PrimitiveIterator*.*OfDouble*.

Помимо коллекций, в каркасе Collections Framework определен ряд интерфейсов и классов отображений, в которых хранятся пары “ключ – значение”: Несмотря на то что отображения входят в состав каркаса коллекций, строго говоря, они не являются коллекциями. Тем не менее для отображения можно получить представление коллекции. Такое представление содержит элементы отображения, хранящиеся в коллекции. Таким образом, содержимое отображения можно при желании обрабатывать как коллекцию. Механизм коллекций был усовершенствован для некоторых классов, изначально определенных в пакете *java.util* таким образом, чтобы интегрировать их в новую систему. Но несмотря на то что внедрение коллекций изменило архитектуру многих первоначальных служебных классов, они не стали от этого не рекомендованными к употреблению. Коллекции просто предлагают лучшее решение некоторых задач.

Компараторы.

Классы *TreeSet* и *TreeMap* сохраняют элементы в отсортированном порядке. Однако понятие "порядок сортировки" точно определяет применяемый ими компаратор. По умолчанию эти классы сохраняют элементы, используя то, что в Java называется естественным упорядочением, т.е. ожидаемым упорядочением, когда после A следует B, а после 1 - 2 и т.д. Если же элементы требуется упорядочить иным образом, то при создании множества или отображения следует указать компаратор типа *Comparator*. Это дает возможность точно управлять порядком сохранения элементов в отсортированных коллекциях.

Интерфейс *Comparator* является обобщенным и объявляется приведенным ниже образом, где *T* обозначает тип сравниваемых объектов.

```
interface Comparator<T>
```

До версии JDK 8 в интерфейсе Comparator определялись только два метода: compare() и equals().Метод compare(),общая форма которого приведена ниже, сравнивает два элемента по порядку.

```
int compare (T объект 1, T объект)
```

Здесь параметры *объект1* и *объект2* обозначают сравниваемые объекты. Обычно этот метод возвращает нулевое значение, если объекты равны; положительное значение, если *объект1* больше, чем *объект2*, а иначе - отрицательное значение. Этот метод может сгенерировать исключение типа `ClassCastException`, если типы сравниваемых объектов несовместимы. Реализуя метод `compare()`, можно изменить порядок расположения объектов. Например, чтобы отсортировать объекты в обратном порядке, можно создать компаратор, который обращает результат их сравнения.

Порядок выполнения работы:

1. Создать минимум 4 класса, которые содержат 3-4 переменные. У всех классов должна быть общая связь между собой. Например:
   Студент (ФИО, группа), Кафедра (название, специальность, группа), Расписание (группа, предмет), Оценки (ФИО студента, группа, оценка, предмет). Где связи Студент.ФИО/Студент.группа – Оценки.ФИО/ Оценки.группа, Студент.группа/Кафедра.группа, Студент.группа/ Расписание.группа.
2. Заполнить данные по всем классам (кодом/из файлов/БД) и хранить их в коллекциях. Минимум по три экземпляра каждого класса.
3. Вывести полную информацию на экран, учитывая связи. Например:
   Студент Иванов Иван Иванович.
   Группа ПИН-120.

Кафедра ПИн.
Специальность 09.03.04.
Предмет «Архитектура вычислительных систем».
Оценка «отлично».

4. Сделать сортировку с использованием компаратора для одного из классов. Вывести результат на экран. Например, список студентов с сортировкой по ФИО.
5. Реализовать в программе возможность поиска информации по конкретному признаку. Например, по ФИО студента.

Варианты тематики классов:

1. Грузоперевозки.
2. Персональный компьютер.
3. Банк.
4. Автосервис.
5. Место отдыха.
6. Строительная фирма.
7. Медицинское учреждение.
8. Швейная мастерская.
9. Интернет провайдер.
10. Деканат
11. Библиотека
12. Отдел кадров