Курс: «Разработка программного обеспечения на Java»

Тема: Java Collection Framework

План

1. Введение в JCF
2. Интерфейсы JCF:
    1. Collection
    2. Comparable и Comparator
    3. Enum
    4. Iterator
    5. List
    6. ListIterator
    7. Map
    8. Map.Entry
    9. Set
3. Классы JCF:
    1. ArrayList
    2. Collections
    3. HashMap
    4. HashSet
    5. LinkedList
    6. BitSet
    7. TreeMap
    8. TreeSet

1. Введение в JCF

Как может показаться на первый взгляд, единственным средством хранения последовательности объектов являются массивы. При работе с ними вы скорее всего заметили, что они имеют ряд недостатков. К примеру, необходимость постоянно следить за размерностью массива, невозможно расширить его или удалить лишний элемент не прибегая к перевыделению памяти.

Однако, в языке Java существует мощный встроенный фреймворк (JCF), который предоставляет готовые структуры данных для удобного хранения и гибкой работы с информацией, называемые коллекциями.

Java Collections Framework представляет собой набор интерфейсов и классов, которые помогают в хранении и эффективной обработки данных. Коллекции имеют большое количество полезных функций в сравнении с массивами.

JCF состоит из двух родительских интерфейсов: Collection и Map. Эти интерфейсы формируют два основных типа хранения данных: последовательные наборы элементов и наборы элементов «ключ - значение», также называемые словарями.

2. Интерфейсы JCF

1. Collection

Collection - это базовый интерфейс коллекции, которая содержит набор элементов. В данном интерфейсе определены основные методы для работы с данными: вставка, удаление, поиск, соответственно: add (или addAll), remove (или removeAll, clear), поиск contains.

2. Comparable и Comparator

Для сравнения и сортировки объектов класса, которые являются элементами коллекции, эти объекты должны реализовывать интерфейс Comparable<E>. При применении данного интерфейса, его следует типизировать текущим классом.

Интерфейс Comparable содержит метод compareTo(E item), который сравнивает текущий объект с объектом, переданным в качестве параметра.

В интерфейсе Comparator<E> объявлен метод compare(Object obj1, Object obj2), который предназначен для упорядочивания объектов коллекции. Его удобно использовать при сортировке упорядоченных списков или массивов объектов. В отличие от метода equals, который возвращает true или false, compareTo возвращает число (отрицательное, положительное или число 0), если:

- значения равны – 0;
- вызываемый объект меньше параметра – < 0;
- вызываемый объект больше параметра – > 0.

Пример:

```
class Student implements Comparator<Student>, Comparable<Student>{
  private String name;
  private int age;

  Student(){
  }

  Student(String name, int age){
    this.name = name;
    this.age = age;
  }

  public String getName(){
```

```
    return name;
  }

  public int getAge(){
    return age;
  }

  public int compareTo(Student s){
    return (this.name).compareTo(s.name);
  }

  public int compare(Student s1, Student s2){
    return s1.age - s12.age;
  }
}

public class Main{
  public static void main(String args[]){
    ArrayList<Student> students = new ArrayList<Student>();

    students.add(new Student("Max", 25));
    students.add(new Student("Andrew", 21));
    students.add(new Student("Lena", 24));
    students.add(new Student("Vlad", 30));

    for(Student s: students)
      System.out.print(s.getName() + ", ");

    // Sorts the array list using comparator
    Collections.sort(students, new Student());
    System.out.println();
    for(Student a: students)
      System.out.print(a.getName() +"  : "+
              a.getAge() + ", ");
  }
}
```

3. Enum

При решении некоторых задач бывает необходимо определять строгое количество значений, которые могут использоваться в программе, например, месяца или дни недели. Для этого можно использовать перечисления (enum).

Пример:

```
public class Main {
   public static void main(String[] args) {
      Day day = Day.MONDAY;
                        System.out.println("Список дел на понедельник:\n" +
ToDo.getMessageFromDay(day));
   }
}

class ToDo {
   public static String getMessageFromDay(Day day) {
      String s;
      switch (day) {
         case SUNDAY:
            s = "-";
            break;
         case MONDAY:
            s = "#1 - Подключить OAuth 2.0 авторизацию\n";
            s += "#2 - Интегрировать соц. сети Facebook и Twitter\n";
            s += "#3 - Встреча с Романом";
            break;
         case TUESDAY:
            s = "#1 - Поехать в Киев\n";
            s += "#2 - Провести презентацию продукта\n";
            break;
         case WEDNESDAY:
            s = "#1 - Подключение API\n";
            break;
         case THURSDAY:
          s = "#1 - Тестирование нового функционала на смартфонах и планшетах\n";
            s = "#2 - Публикация приложения в Google Play\n";
            break;
         case FRIDAY:
            s = "#1 - Сдать недельный отчет";
            break;
         case SATURDAY:
            s = "-";
            break;
         default:
            s = null;
            break;
      }
```

```
        return s;
    }
}

enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

4. Iterator

Благодаря интерфейсу Iterator возможно пройтись по всем элементам некоторого объекта. Одним из важных условий реализации итератора является то, что итератор должен гарантировать нераскрытие внутреннего устройства объекта.

Интерфейс содержит следующие методы:

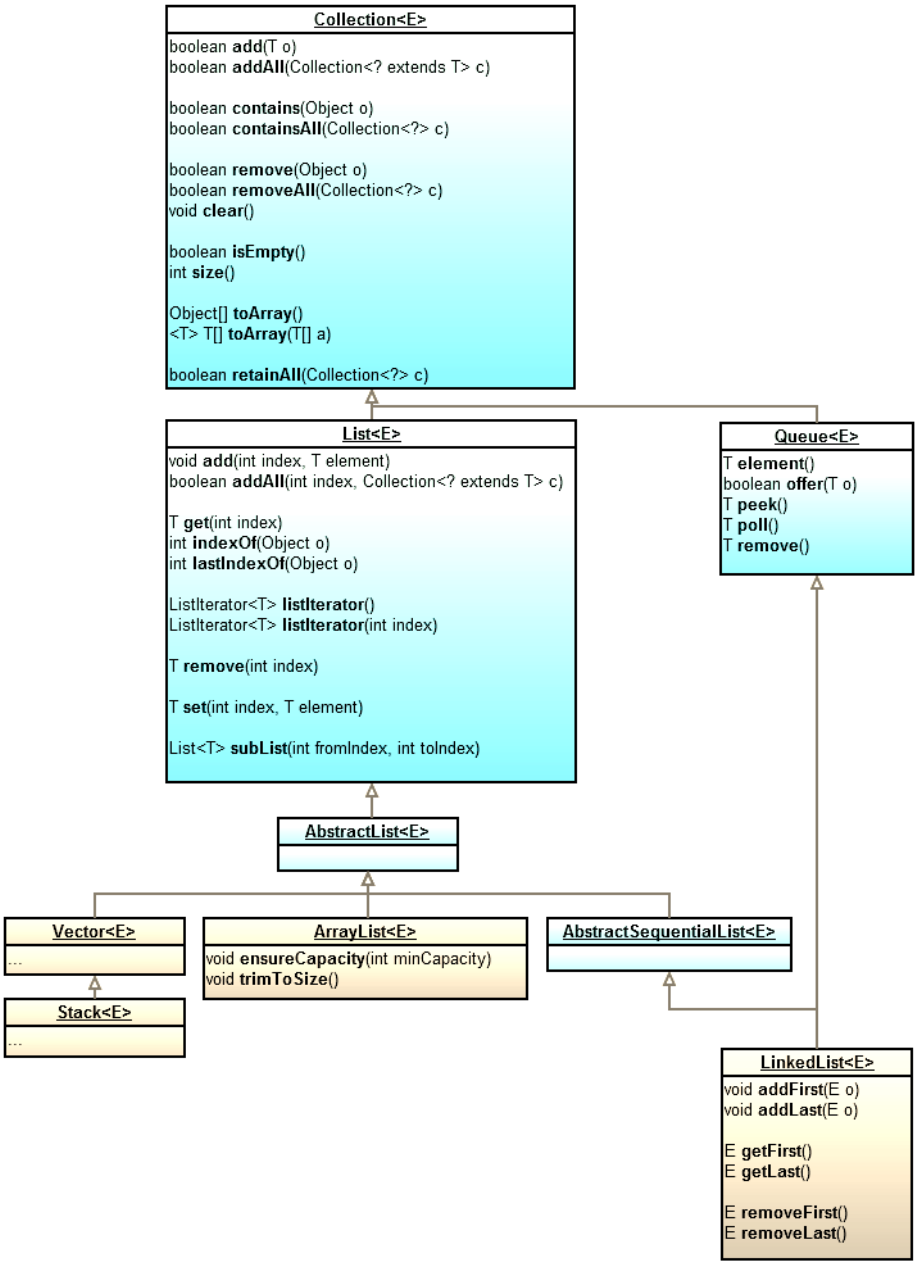
- hasNext() - возвращает true\false в зависимости от того, есть ли еще элементы в коллекции.
- next() - возвращает следующий элемент в коллекции, пока не достигнут последний элемент.
- remove() - удаляет элемент, который был возвращен последним вызовом метода next.()

5. List

Интерфейс List представляет собой упорядоченную коллекцию. List принято называть списком. Список может содержать повторяющиеся элементы. Программист имеет возможность управлять индексом вставки элемента в коллекцию и может получить доступ к элементам списка по индексу. В дополнение к стандартным функциям интерфейса Collection, List содержит также: доступ по позиции, поиск, наличие ListIterator, получения диапазона элементов.

Интерфейс List реализуют несколько классов, которые следует использовать в том или ином случае, они отличаются способом хранения элементов, поэтому имеют разные показатели производительности при вставке и получении элемента коллекции.

Иерархия классов List:



6. ListIterator

Для работы со списком можно использовать ListIterator, который позволяет работать с последовательными свойствами списка.

ListIterator позволяет вставлять и заменять элементы и производить перебор элементов в двух направлениях.

Пример объявление интерфейса:

```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void remove();
    void set(E e);
    void add(E e);
}
```

7. Map

Интерфейс Map реализует такую структуру данных, благодаря которой элемент коллекции ассоциируются с определенным ключом. Поэтому доступ к этому элементу осуществляется не по позиции, а по этому ключу. В качестве ключа может выступать объект любого класса.

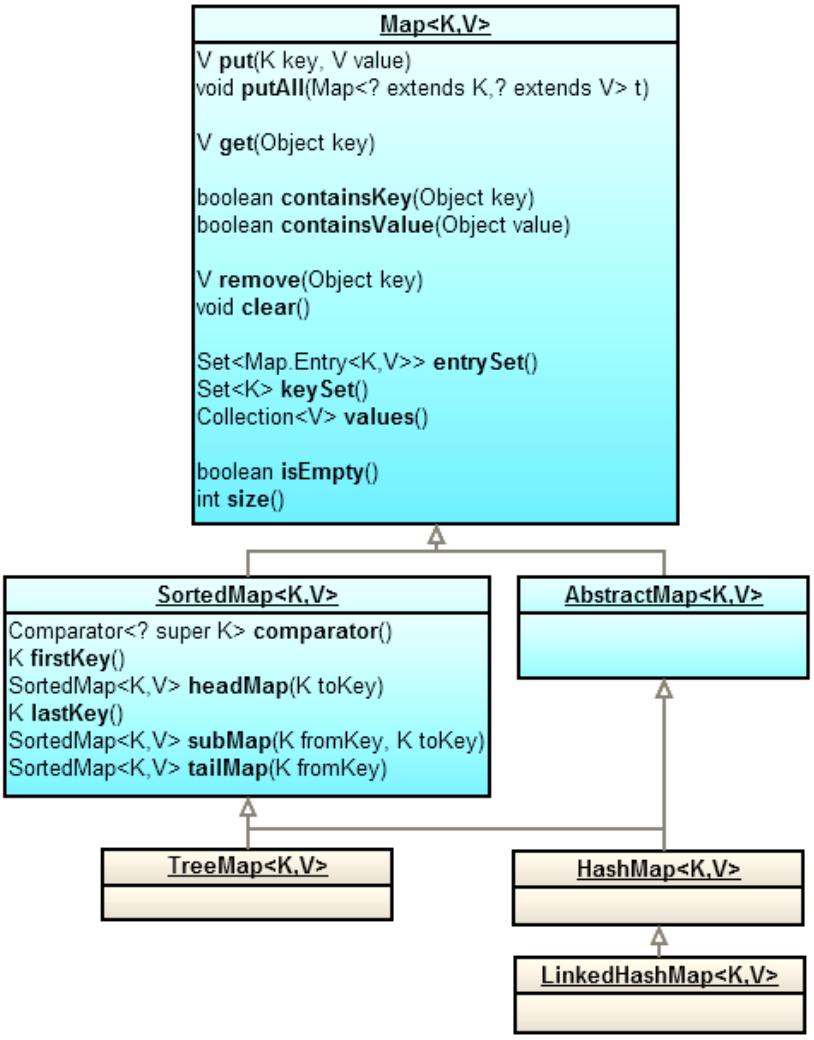
interface Map<K, V> – K определяет тип ключа, V – тип значения.

Рассмотрим основные методы Map:

- put(K, V) – Добавляет ключ и значение в коллекцию. Если такой ключ уже имеется, то новый объект заменяет предыдущий, который связан с этим ключом.
- get(K) – Возвращает объект, соответствующий указанному ключу или значение null, если словарь не содержит указанный ключ.
- putAll(Map<? extends K, ? extends V> entries) – Добавляет все элементы заданного словаря к текущему. Типы данных должны быть соответствующими.
- containsKey(Object key) – Возвращает значение true, если в словаре имеется указанный ключ.
- containsValue(Object value) – Возвращает значение true, если в словаре имеется указанное значение.

Также, как и List, интерфейс Map имеет несколько конечных реализаций в виде классов HashMap, TreeMap и LinkedHashMap, которые можно использовать «с коробки».

Иерархия классов Map:



8. Map.Entry

Map.Entry – это интерфейс, который используется для хранение конкретной пары «ключ-значение» в коллекции Map. Одним из вариантов использования Map.Entry является проход по всем элементам словаря. Метод Map.entrySet() возвращает набор ключ-значений, потому самым эффективным способом пройтись по всем значениям Map будет:

```
for(Entry entry: Map.entrySet()) {
  //получить ключ
  K key = entry.getKey();
  //получить значение
  V value = entry.getValue();
}
```

9. Set

Коллекция Set схожа с коллекцией List, за исключением того факта, что в ней невозможно хранить повторяющиеся элементы (object1.equals(object2) == true) и не более одного null, из-за этой особенности Set называют множеством.

Методы интерфейса:

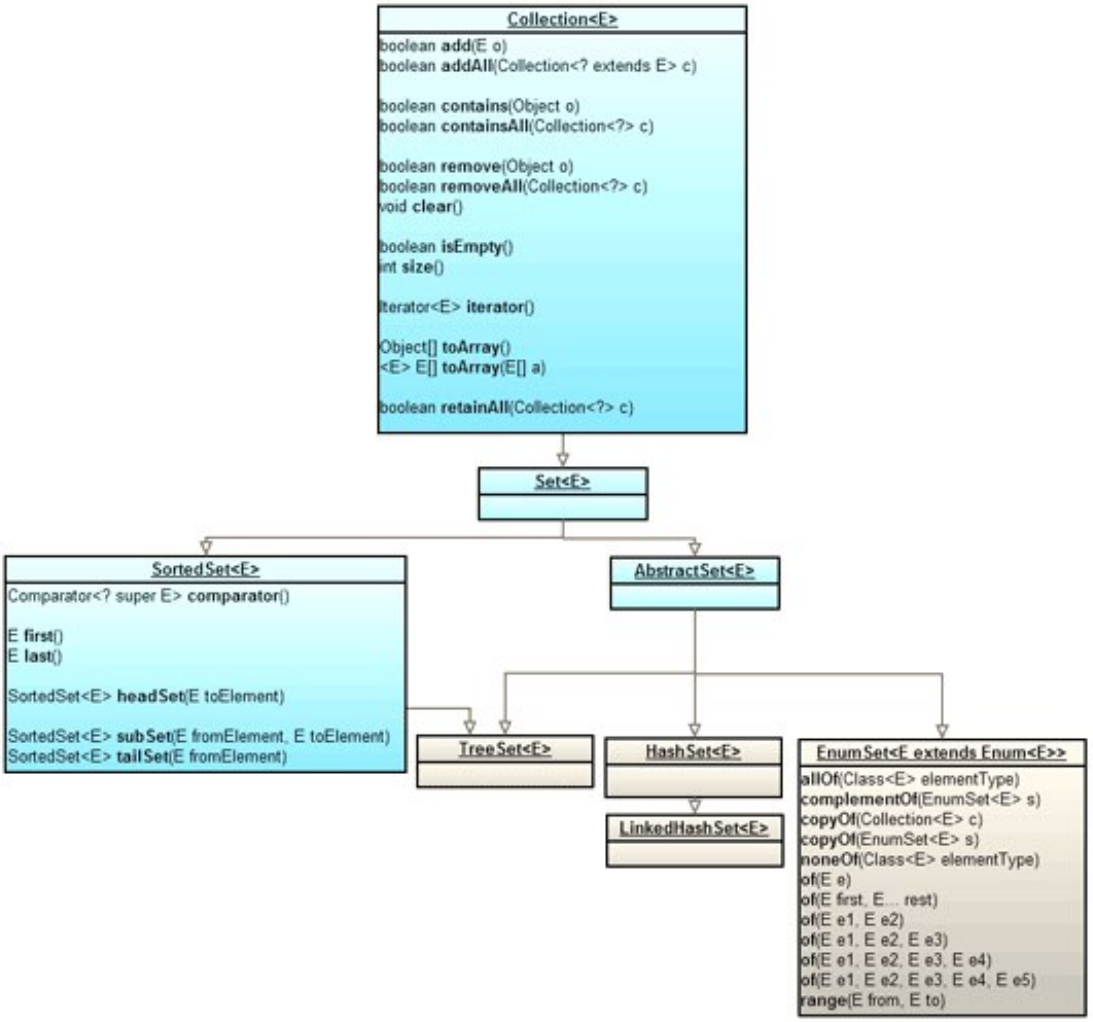
- add(E e) — добавление элемента в коллекцию. Возвращает true, если элемент добавлен
- addAll(Collection c) — добавление всех элементов другой коллекции
- clear() — удаление всех элементов множества
- contains(Object o) — возвращает true, если элемент присутствует в коллекции
- containsAll(Collection c) — возвращает true, если все элементы содержатся в коллекции
- equals(Object o) — сравнение коллекций
- isEmpty() — возвращает true если в коллекции отсутствуют элементы
- iterator() — возвращает итератор коллекции
- remove(Object o) — удаляет указанных объект из множества
- removeAll(Collection c) — удаление элементов, принадлежащих переданной коллекции
- retainAll(Collection c) — удаление элементов, не принадлежащих переданной коллекции
- size() — количество элементов множества
- toArray() — возвращает массив, содержащий элементы коллекции
- toArray(T[] a) —возвращает массив объектов такого типа, который был указан в параметре.

Несколько замечаний:

- Некоторые реализации Set имеют ограничения на элементы, которые они могут содержать. Например, в некоторых множествах не разрешено добавление null. При попытке добавления null произойдет исключение NullPointerException или ClassCastException.
- Две коллекции Set равны в том случае, если содержат одинаковые элементы. Если расположение элементов отличается, но их значения равны, тогда метод equals() всё равно вернет true.
- Если добавление элемента завершилось неудачей (например, такой элемент уже существует в коллекции), то возвращается false. Иначе - true.

У данного интерфейса есть три готовых реализации: HashSet, LinkedHashSet и TreeSet.

Иерархия классов Set:



3. Классы JCF

1. ArrayList

Класс ArrayList является наиболее часто используемой коллекцией, его преимущество перед массивом в том, что список динамически расширяемый и имеет ряд удобных методов для работы с элементами.

Конструкторы класса:

- ArrayList() – создание пустого списка
- ArrayList(Collection <? extends E> c) – создание списка и с копированием значений из другой коллекции
- ArrayList (int capacity) – создает списка с заданным параметром capacity (емкость). Емкость коллекции, как правило, больше текущего размера списка, при превышении этого показателя происходит перераспределение памяти, которе влияет на производительность. Чем емкость больше, тем реже будет проходить перераспределение, но не следует определять большую емкость, если количество элементов коллекции будет минимальным. Стандартная реализация автоматически увеличивает емкость при добавлении новых элементов.

Рассмотрим основные методы ArrayList:

- get(int index) – возвращает объект из списка по индексу
- add(E obj) – добавляет в элемент в конец списка
- add(int index, E obj) – добавляет элемент списка в позицию index
- addAll(Collection<? extends E> c) – добавляет в список элементы другой коллекции col. Есть перегруженная версия с добавлением в конкретную позицию.
- indexOf(Object obj) – возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1
- size() – возвращает размер списка
- remove(int index) – удаляет объект из списка по индексу, возвращая при этом удаленный объект
- set(int index, E obj): присваивает значение объекта obj элементу, который находится по индексу index
- sort(Comparator<? super E> comp): сортирует список с помощью компаратора comp
- substring(int start, int end): получает набор элементов, которые находятся в списке между индексами start и end

Пример использования:

```
public class Main {

    public static void printList(ArrayList list) {
        System.out.print("Items: ");
        for (int i = 0; i < list.size(); i++)
            System.out.print(list.get(i) + " ");
        System.out.println();
    }

    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Item1");
        list.add("Item2");
        list.add("Item3");
        list.add("Item4");
        list.add("Item5");

        printList(list);
        System.out.println("size() = " + list.size());
        System.out.println("get(2) = " + list.get(2));
        System.out.println("add(\"NewItem6\")");
        list.add("NewItem6");
        System.out.println("add(0,\"NewItem0\")");
        list.add(0, "NewItem0");
        System.out.println("size() = " + list.size());
        printList(list);
    }
}
```

2. Collections

Класс Collections содержит ряд дополнительных методов, предназначенных для манипулирования коллекциями.

Рассмотрим некоторые из них:

- addAll(Collection<? super T> c, T… a) – добавляет в коллекцию элементы соответствующего типа
- copy(List<? super T> dest, List<? extends T> src) – копирует все элементы из одного списка в другой

- replaceAll(List<T> list, T oldValue, T newValue) – заменяет все заданные элементы новыми
- void reverse(List<?> list) – переворачивает список
- void rotate(List<?> list, int distance) – сдвигает список циклически на заданное число элементов
- disjoint(Collection<?> c1, Collection<?> c2) – возвращает true, если коллекции не содержат одинаковых элементов;
- emptyList(), emptyMap(), emptySet() – возвращают пустой список, словарь и множество соответственно
- fill(List<? super T> list, T obj) – заполняет список заданным элементом
- frequency(Collection<?> c, Object o) – возвращает количество вхождений в коллекцию заданного элемента
- checkedList(), checkedMap(), heckedSet() и т.п. – проверяет коллекции на наличие «посторонних» объектов
- max(Collection<? extends T> coll) и min(Collection<? extends T> coll) – возвращают минимальный и максимальный элемент соответственно, есть перегруженная версия с компаратором сравнения

3. HashMap

Класс HashMap реализует интерфейс Map, который был рассмотрен ранее. Его особенность в том, что он использует хеш-таблицу для хранения элементов, это обеспечивает быстрое время выполнения запросов get() и put() при больших наборах. Порядок элементов не будет соответствовать порядку добавления. Ключи и значения могут быть любых типов, в том числе и null. Все ключи должны быть уникальны, а значения могут повторяться.

Что такое хэш? Это битовая строка фиксированной длины, или если проще, это целое число, которое можно получить вызвав соответствующий метод объекта. Метод работает так, что для конкретного объекта будет вычислено некоторое число, причем для одного и того же объекта, это число будет одинаковое, а другие объекты будут иметь другой хеш-код (возможны повторения из-за ограниченности значений типа int).

Рассмотрим пример работы со HashMap (реализация простейшего русско-английского словаря для разработчиков):

```
public class Main {
   public static void main(String[] args) {
      HashMap<String, String> dictionary = new HashMap<>();

      dictionary.put("метод", "method");
      dictionary.put("объект", "object");
```

```
        dictionary.put("класс", "class");
        dictionary.put("интерфейс", "interface");
        dictionary.put("инкапсуляция", "encapsulation");
        dictionary.put("поле", "field");
        dictionary.put("наследование", "inheritance");
        dictionary.put("переменная", "variable");
        dictionary.put("константа", "constant");

 System.out.println("Слово: интерфейс, перевод: " + dictionary.get("интерфейс"));
System.out.println("Слово: переменная, перевод: " + dictionary.get("переменная"));
        System.out.println("Слово: объект, перевод: " + dictionary.get("объект"));
    }

}
```

4. HashSet

Этот класс реализует интерфейс Set, который поддерживает хранение элементов в виде хэш-таблицы. Этот подход не дает гарантий относительно итеративного порядка набора, в частности это не гарантирует, что порядок останется постоянным в течение некоторого времени. Этот класс разрешает использовать элемент null. HashSet имеет постоянную производительность для основных операций: add, remove, contains и size.

Пример работы с HashSet:

```
public class Main {
    private static HashSet<String> countries = new HashSet<>();

    public static void addCountry(String name) {
        countries.add(name);
    }
    public static void printCountries() {
        Iterator<String> i = countries.iterator();
        while (i.hasNext()) {
            System.out.print(i.next() + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        System.out.println("Сколько стран вы знаете?");
        addCountry("Украина");
```

```
        addCountry("США");
        addCountry("Россия");
        addCountry("Канада");
        addCountry("Польша");
        addCountry("Украина");
        addCountry("Турция");
        addCountry("Франция");
        addCountry("Украина");
        addCountry("Бельгия");
        addCountry("США");
        addCountry("Великобритания");

        System.out.println("Вы знаете " + countries.size() + " стран: ");
        printCountries();
    }

}
```

Как видно из результатов выполнения программы, несколько стран повторяются, но при выводе элементов коллекции все уникальные строки существуют в единственном экземпляре.

5. LinkedList

Данный список представляет собой структуру данных в виде связанного списка. Он наследуется от класса AbstractSequentialList и реализует интерфейсы List, Dequeue и Queue. Список позволяет добавлять любые элементы в том числе и null.

Элементы списка представляют собой объекты класса Entry, которые хранят ссылки на предыдущий и следующий элементы, включая header, который является «начальным» псевдо-элементом списка.

Методы LinkedList, которые не были рассмотрены ранее:

- addFirst() – добавляет элемент в начало списка
- addLast() – добавляет элемент в конец списка
- removeFirst() – удаляет первый элемент из начала списка
- removeLast() – удаляет последний элемент из конца списка
- getFirst() – получает первый элемент
- getLast() – получает последний элемент

LinkedList хорошо подходит для реализации очереди или стека. Он работает заметно быстрее чем ArrayList, только при вставке или удалении элементов вначале списка. Для всех остальных случаев рекомендуется использовать ArrayList.

6. BitSet

BitSet позволяет создать битовый вектор, размер которого изменяется динамически. Один бит может принимать значение true или false. По умолчанию, все биты равны false. Для хранения набора выделяется объем памяти, необходимый для хранения вектора вплоть до старшего бита, который устанавливался или сбрасывался в программе — все превышающие его биты считаются равными false.
При создании экземпляра BitSet можно определить исходный размер вектора или использовать конструктор по умолчанию.

Рассмотрим методы класса:

- set(int i) – устанавливает бит в позиции i, присваивая ему значение true.
- clear(int i) – сбрасывает бит в позиции i, присваивая ему значение false.
- get(int i) – возвращает значение бита в позиции i.
- and(BitSet bset) – выполняет операцию логического И текущего и передаваемого набора и присваивает результат данному набору.
- or(BitSet other) – выполняет операцию логического ИЛИ текущего и передаваемого набора и присваивает результат данному набору.
- xor(BitSet other) – выполняет операцию исключающего логического ИЛИ текущего и передаваемого набора и присваивает результат данному набору.
- size() – возвращает позицию старшего бита в наборе, который может быть установлен или сброшен без необходимости увеличения набора.

7. TreeMap

Класс TreeMap является структурой данных, которая представляет информацию в виде дерева. Данный класс расширяет класс AbstractMap и реализует интерфейс NavigatebleMap. В TreeMap объекты хранятся в отсортированном порядке по возрастанию, поэтому эту коллекция достаточно производительная в момент доступа и извлечения элементов. Этот класс является хорошим выбором для хранения больших объемов отсортированной информации, которая должна быть быстро найдена. Компаратор не позволяет использовать ключ null, будет выброшено исключение, зато значение может быть null.

TreeMap основана на структуре данных Красно-Черное дерево, что гарантирует скорость доступа log(n) для операций containsKey, get, put и remove.

Пример (после запуска вы убедитесь, что элементы отсортированы по ключу):

```
public class Main {
   public static void main(String[] args) {
      TreeMap<String, String> treeMap = new TreeMap<>();
      treeMap.put("Украина", "Киев");
      treeMap.put("Япония", "Токио");
      treeMap.put("Польша", "Варшава");
      treeMap.put("США", "Вашингтон");
      treeMap.put("Канада", "Оттава");
      treeMap.put("Англия", "Лондон");
      treeMap.put("Польша", "Варшава");

      for (Map.Entry e : treeMap.entrySet()) {
         System.out.println(e.getKey() + " " + e.getValue());
      }
   }
}
```

8. TreeSet

Класс TreeSet создаёт множество, которое для хранения элементов применяет дерево. Объекты сохраняются в отсортированном порядке по возрастанию, что очень важно, если вам необходимо хранить элементы в упорядоченном виде, т.к. HashSet не гарантирует упорядоченный порядок элементов.

Рассмотрим предыдущий пример с использованием TreeSet для хранения стран в сортированном виде:

```
public class Main {
   public static void main(String[] args) {
      TreeSet<String> treeMap = new TreeSet<>();
      treeMap.add("Украина");
      treeMap.add("Япония");
      treeMap.add("США");
      treeMap.add("Канада");
      treeMap.add("Англия");
      treeMap.add("Польша");

      System.out.println(treeMap.toString());
   }
}
```