Java SE
4. Collections

© 2011 NetCracker Technology Corp. Confidential.

Java Collections Framework

© 2011 NetCracker Technology Corp. Confidential.

Java Collections Framework

- Коллекции (контейнеры) - хранилища, поддерживающие разнообразные способы накопления и упорядочивания объектов с целью обеспечения возможностей эффективного доступа
- В отличие от массивов могут поддерживать дополнительную функциональность и быть более эффективными в определенных ситуациях
- Примерный аналог контейнеров и итераторов STL
- В Java коллекции разделены на интерфейсы, абстрагирующие общие принципы работы с коллекциями, и классы, реализующие конкретную функциональность
- Большинство из них размещено в пакете **java.util.*** и его подпакетах

© 2011 NetCracker Technology Corp. Confidential.

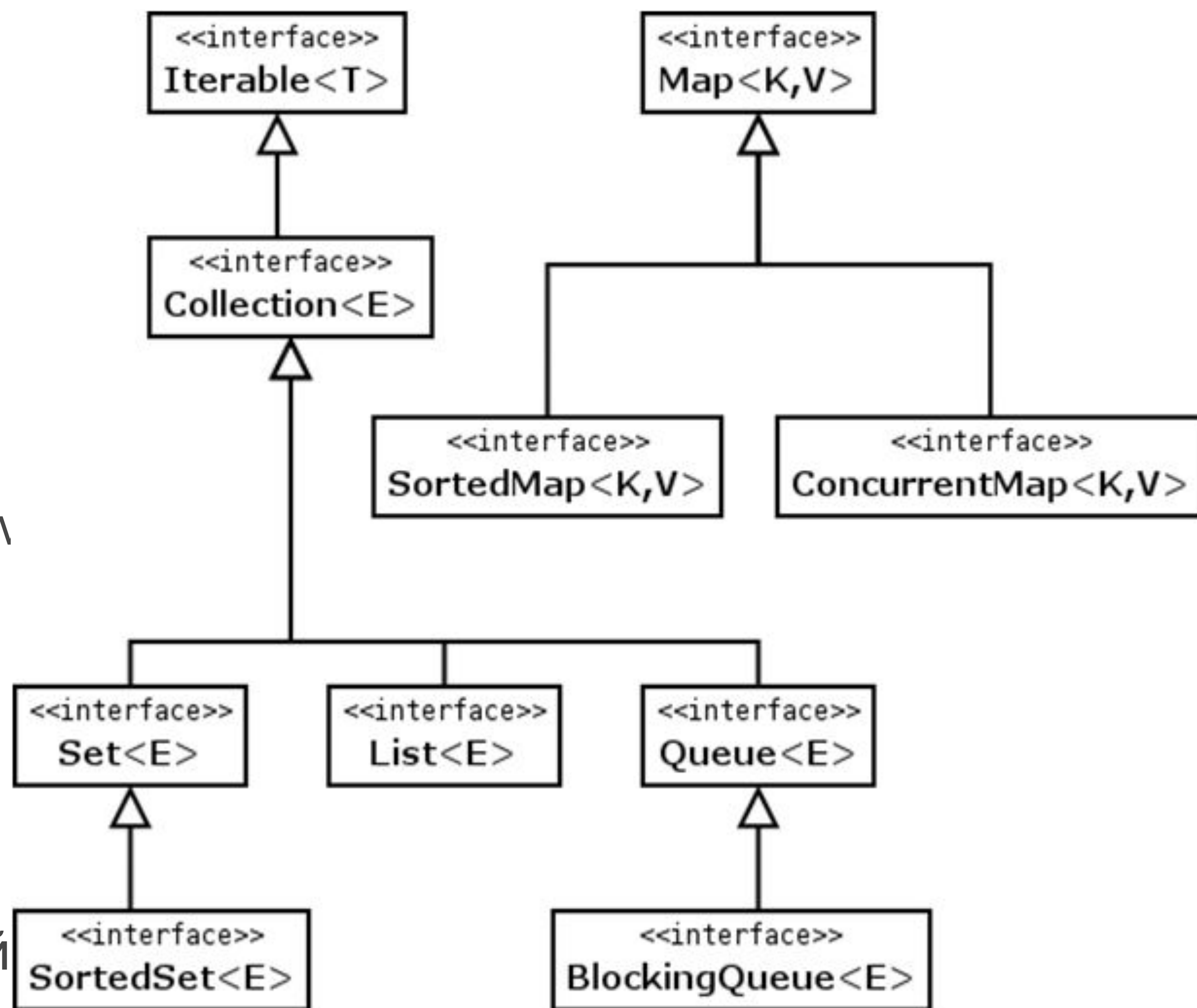
Java Collections Framework

- **Collection Interfaces** – представляют собой описания фундаментальных типов контейнеров и возможных операций над ними.
- **General-purpose Implementations** – Самые часто используемые реализации эти интерфейсов.
- **Legacy Implementations** – устаревшие контейнеры, существовавшие еще до появления Collection Framework и переписанные для реализации Collection-интерфейсов.
- **Wrapper Implementations** – сами по себе не хранят данных, но добавляют функциональность к другим коллекциям.
- **Convenience Implementations** – высокопроизводительные тривиальные реализации для простых случаев.
- **Abstract Implementations** – частичные реализации как основа для собственных коллекций.
- **Arrays/Collections Utilities** – набор вспомогательных утилитных методов для работы с коллекциями и

© 2011 NetCracker Technology Corp. Confidential.

Иерархия интерфейсов

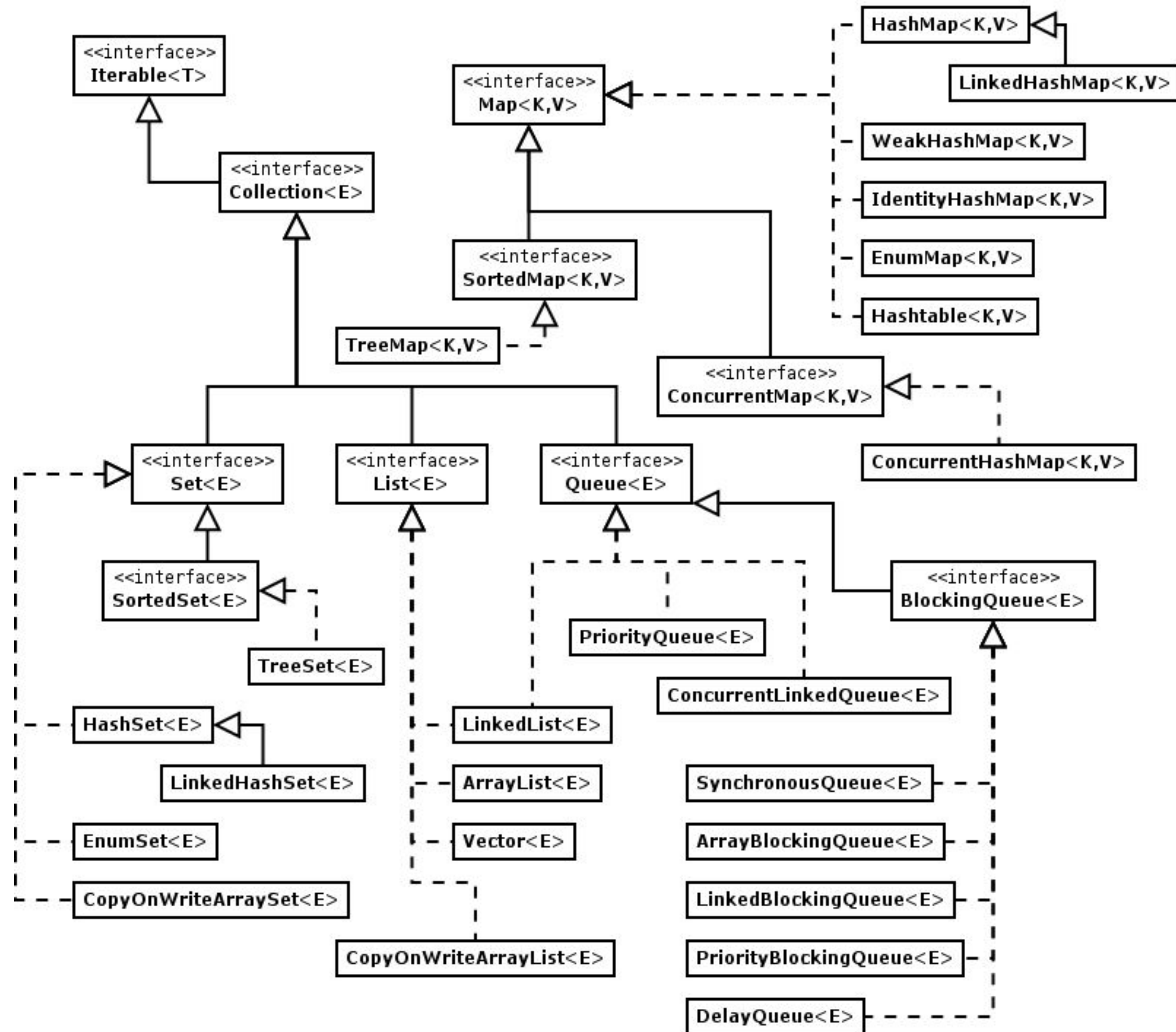
- Интерфейсы представляют собой наиболее общие описания фундаментально различных типов коллекций
- **List** – упорядоченный список с позиционным доступом
- **Set** – коллекция без дубликатов
- **Queue** – очередь элементов или стэк
- **Map** – ассоциативный массив



© 2011 NetCracker Technology Corp. Confidential.

Иерархия реализаций

- Эта схема показывает только основные реализации, входящие в JDK. В сторонних библиотеках можно найти сотни других реализаций с самыми разными свойствами



© 2011 NetCracker Technology Corp. Confidential.

java.util.Collection

© 2011 NetCracker Technology Corp. Confidential.

public interface Collection<E> extends Iterable<E>

- Является образующим для интерфейсов коллекций
- Определяет базовую функциональность любой коллекции
- Подразумевает добавление, удаление, выбор элементов в коллекции
- Допускает дубликаты и пустые элементы
- Позволяет получить итератор для обхода коллекции
- Не все методы, заявленные в интерфейсе, должны реализовываться классами. Часть методов может выбрасывать **UnsupportedOperationException**

public interface Collection<E> extends Iterable<E>

```
public interface Collection<E> extends Iterable<E> {
    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);         //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    // Array operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```


© 2011 NetCracker Technology Corp. Confidential.

java.util.Set

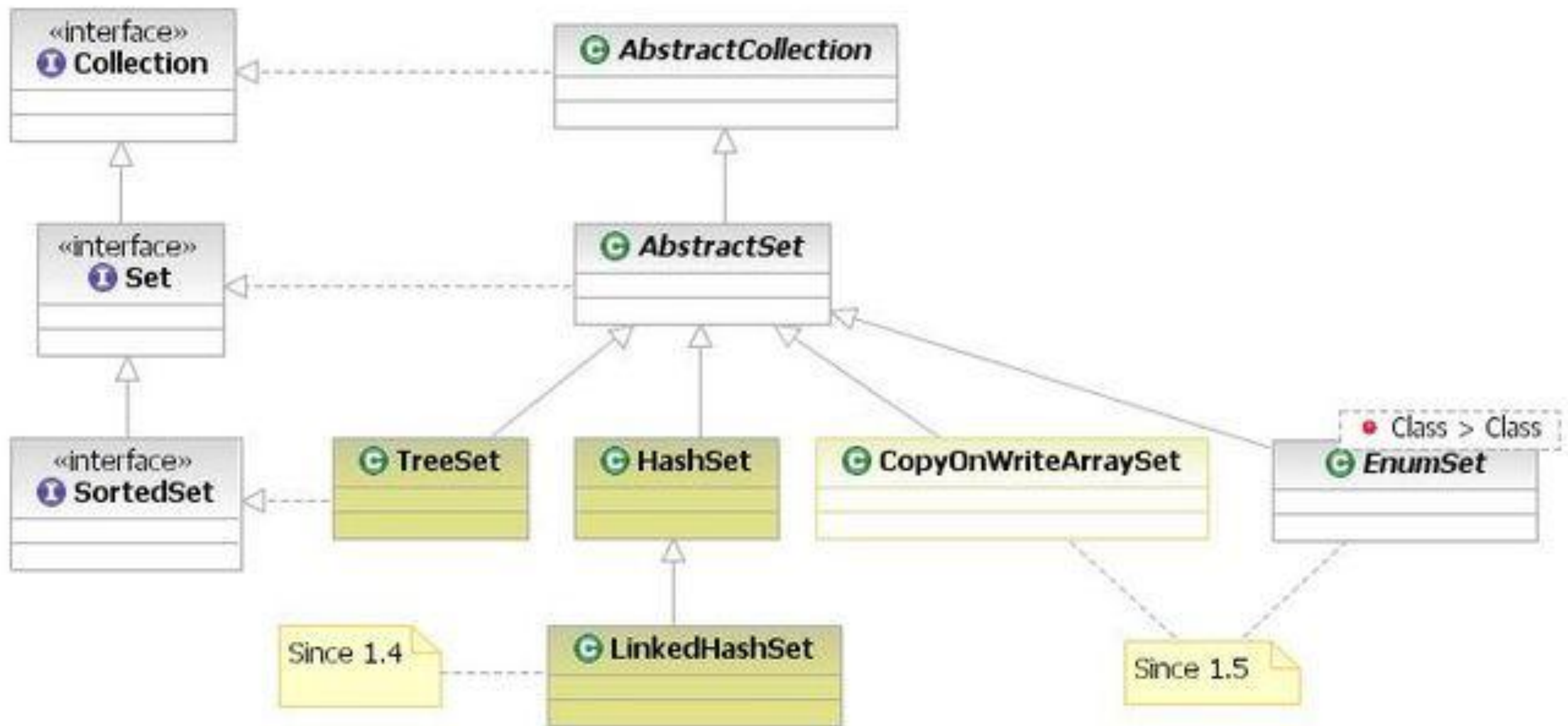
© 2011 NetCracker Technology Corp. Confidential.

public interface **Set<E>** extends Collection<E>

- **Set** — коллекция, не содержащая дубликатов
- Может содержать не более одного Null-значения
- Все остальные свойства могут варьироваться в зависимости от конкретной реализации
- **Set** не добавляет дополнительных методов к интерфейсу **Collection** и является маркерным интерфейсом с дополнительной документацией
- Как правило **Set** не поддерживает порядок элементов, но некоторые реализации это позволяют

© 2011 NetCracker Technology Corp. Confidential.

Основные реализации и дочерние интерфейсы Set

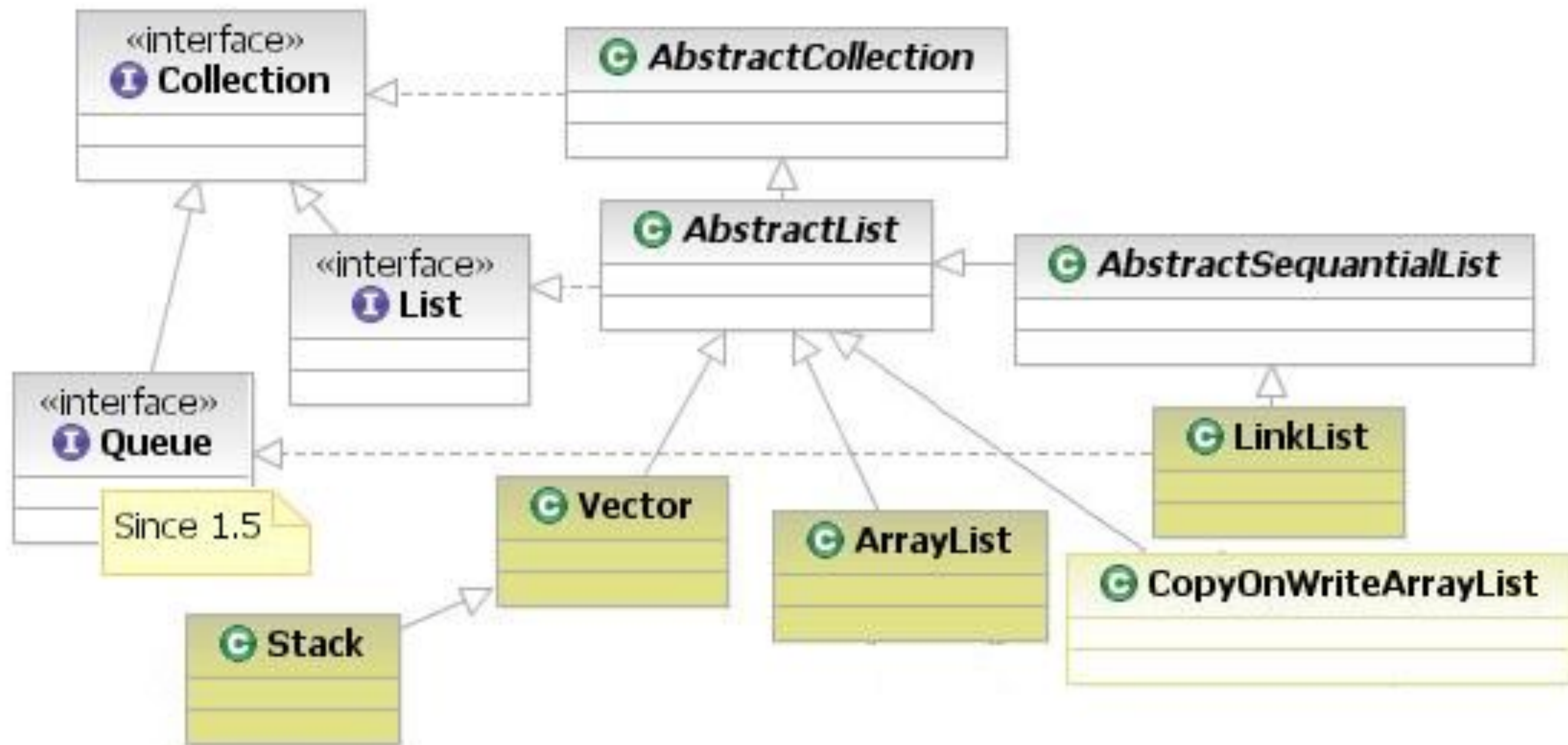


- **HashSet** – неупорядоченное множество, реализованное через хэш-таблицу
- **TreeSet** – отсортированное множество на красно-черных деревьях
- **LinkedHashSet** – множество, сохраняющее порядок добавления элементов
- **CopyOnWriteArraySet** – потоково-безопасная реализация

java.util.List

© 2011 NetCracker Technology Corp. Confidential.

Основные реализации и дочерние интерфейсы List



- List — упорядоченная по времени добавления коллекция
- В отличие от других коллекций List позволяет делать позиционный доступ к элементам по индексам
- Некоторые старые коллекции были переделаны, чтобы также реализовывать этот интерфейс (Vector, Stack)

© 2011 NetCracker Technology Corp. Confidential.

public interface **List<E>** extends Collection<E>

```java
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);    //optional
    boolean add(E element);         //optional
    void add(int index, E element); //optional
    E remove(int index);            //optional
    boolean addAll(int index, Collection<? extends E> c); //optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
}
```


© 2011 NetCracker Technology Corp. Confidential.

Основные реализации List

- **Vector** - Legacy-коллекция, адаптированная к интерфейсу List. Синхронизированная и безопасная в многопоточной среде
- **Stack** - Наследник вектора, реализующий LIFO структуру данных
- **ArrayList** - Самая распространенная реализация на базе массива
- **LinkedList** - Реализация на базе связного списка, также этот класс реализует интерфейс **Queue** и может выступать в качестве очереди
- **CopyOnWriteArrayList** – Потоково безопасная реализация, создающая копию массива данных при каждой операции записи

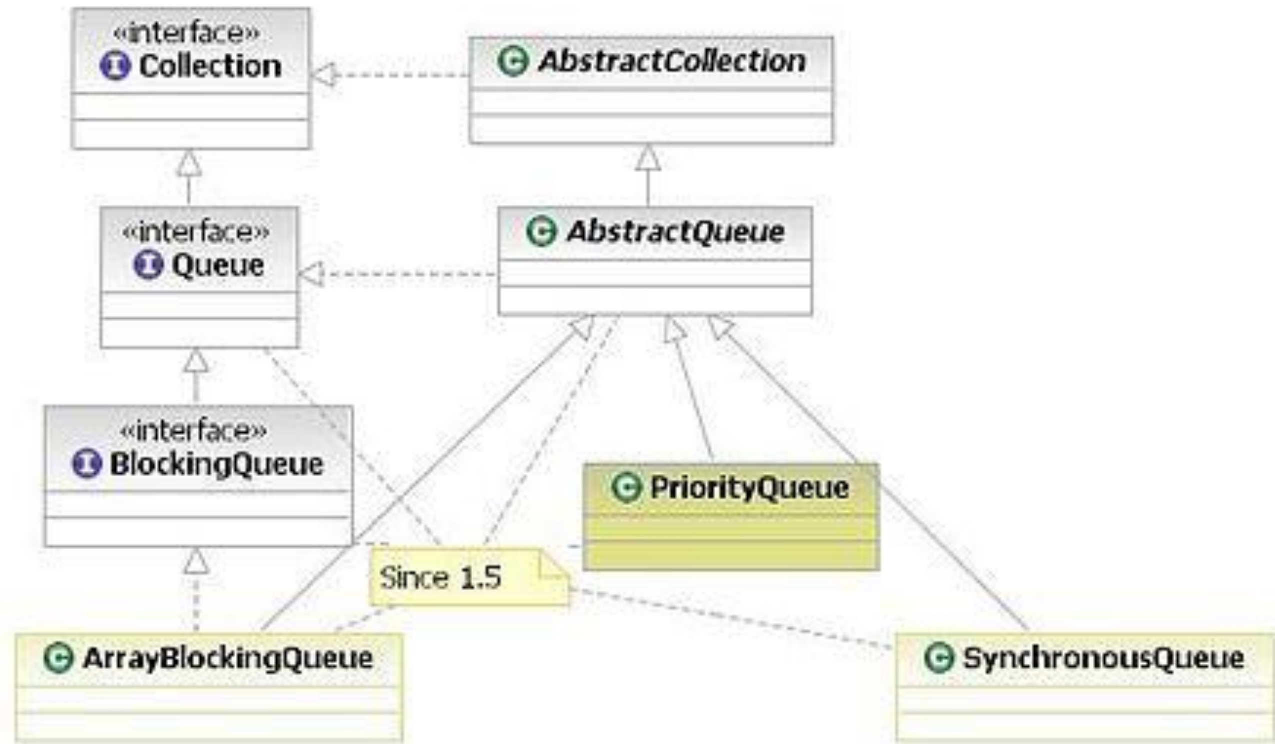
© 2011 NetCracker Technology Corp. Confidential.

java.util.Queue

© 2011 NetCracker Technology Corp. Confidential.

Основные реализации и дочерние интерфейсы Queue

- **PriorityQueue** - упорядочивает элементы на основе естественного порядка или реализации Comparator, переданной в конструктор при создании очереди



- **ArrayBlockingQueue** хранит элементы в порядке FIFO; синхронизированная реализация.
- **SynchronousQueue** - каждая операция добавления будет блокирована до соответствующей операции чтения и наоборот. Фактически это блокирующая ячейка под единственный элемент

© 2011 NetCracker Technology Corp. Confidential.

Queue API

- Queue представляет много дополнительных методов для работы с данными помимо стандартных

```
public interface Queue<E> extends Collection<E> {
    E element();
    E peek();
    boolean offer(E e);
    E remove();
    E poll();
}
```

- Они позволяют выполнять вставку или получение элемента с разным поведением в ошибочных ситцуациях

Queue – доступные операции и обработка ошибок		
Действие	Бросает исключение	Возвращает специальное значение
Вставка	add(e)	offer(e)
Удаление	remove()	poll()
Просмотр	element()	peek()

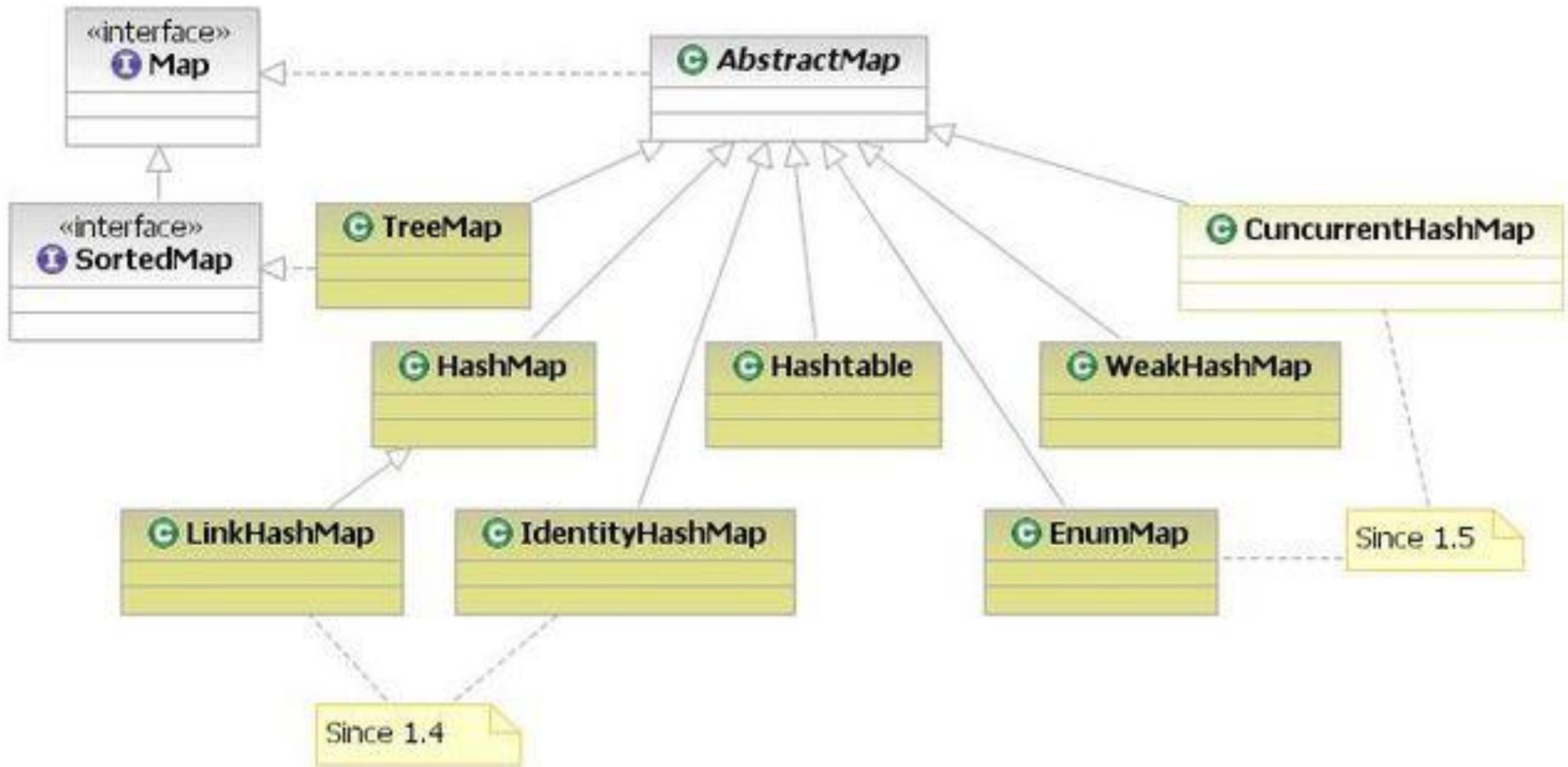
NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

java.util.Map

© 2011 NetCracker Technology Corp. Confidential.

Основные реализации и дочерние интерфейсы Map



- **Map<K, V>** – ассоциативный массив, коллекция пар ключ-значение
- Одному ключу не может соответствовать более одного значения. Так называемых Multimap в Java Collection Framework

© 2011 NetCracker Technology Corp. Confidential.

public interface **Map<K,V>**

- Map предоставляет много вариантов перебора содержимого
  - Через коллекцию ключей - **keySet()**
  - Через коллекцию значений – **values()**
  - Через коллекцию пар, так называемых **Map.Entry**
- Разные реализации могут допускать или не допускать Null-значения
- Примитивный тип не может выступать в роли ключа или значения

```
public interface Map<K,V> {

    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    // Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Entry<K,V>> entrySet();

    // Интерфейс для пар ключ-значение
    public interface Entry<K,V> {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Map API example

```
//map to hold student grades
Map<String, Integer> map = new HashMap<String, Integer>();

map.put("Evan", 100);
map.put("Robert", 90);
map.put("Wayne", 92);
map.put("Michael", 98);
map.put("John", 88);

System.out.println(map);    // {Robert=90, Wayne=92, Evan=100, John=88, Michael=98}

System.out.println(map.get("Evan"));            //  100
System.out.println(map.get("Michael"));         //  98
System.out.println(map.keySet());               // [Robert, Wayne, Evan, John, Michael]
System.out.println(map.isEmpty());              // false
System.out.println(map.size());                 // 5
```

- Типизация Map при помощи Generics позволяет быть уверенным в том, что все ключи имеют одинаковый тип и все значения также имеют одинаковый тип, возможно не совпадающий с типом ключей

© 2011 NetCracker Technology Corp. Confidential.

Основные реализации Map

- **HashMap** - Самая распространенная реализация, основана на хэш-таблице
- **ConcurrentHashMap** - Реализация для работы в многопоточной среде, причем доступ на чтение будет неблокирующим
- **Hashtable** - Legacy-коллекция с синхронизированным доступом
- **WeakHashMap** - Эта реализация будет удалять записи, на ключи которых нет ссылок за пределами коллекции
- **LinkedHashMap** – Гарантирует, что элементы коллекции будут возвращаться в том же порядке, что и были в нее добавлены
- **TreeMap** – Ключи в этой коллекции будут отсортированы согласно Cormparator'у или реализации Comparable
- **IdentityHashMap** - Эта реализация использует для сравнения элементов равенство ссылок вместо вызова **equals()**.

© 2011 NetCracker Technology Corp. Confidential.

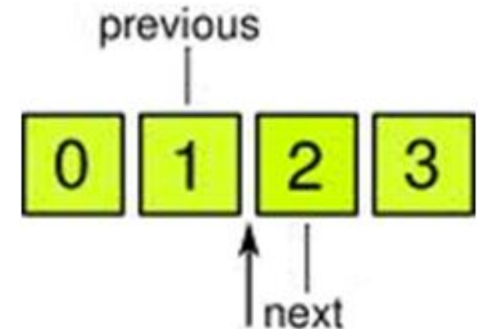
Итераторы

© 2011 NetCracker Technology Corp. Confidential.

Iterator

- Iterator – специальный объект для последовательного обхода коллекции
- Является реализацией одноименного шаблона проектирования
- Iterator можно получить из любой коллекции вызовом метода iterator()
- Для абстрактной коллекции это единственный доступный способ обхода
- Цикл for each использует итератор неявным образом
- Интерфейс итератора:

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```



© 2011 NetCracker Technology Corp. Confidential.

Iterator – пример использования

```
class Wallet {
    private ArrayList<Integer> bills = new ArrayList<Integer>();

    public void addBill(Integer aBill) {
        bills.add(new Integer(aBill));
    }

    public Integer getMoneyTotal() {
        Integer total = 0;
        Iterator<Integer> iterator = bills.iterator();
        while (iterator.hasNext()) {
            total += iterator.next();
        }
        return total;
    }
}
```

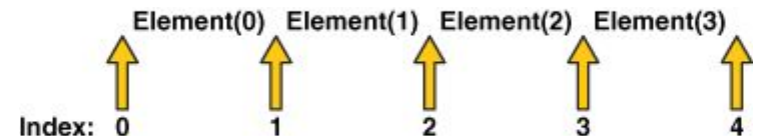
NetCracker®
Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

ListIterator

- Расширяет стандартный итератор дополнительной функциональностью:
  - В отличие от простого итератора позволяет двигаться не только вперед по коллекции, но и назад
  - Метод **set()** перезапишет предыдущий элемент
  - Метод **add()** добавит новый элемент в коллекцию непосредственно перед указателем итератора

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



© 2011 NetCracker Technology Corp. Confidential.

Сравнение и сортировка элементов коллекций

© 2011 NetCracker Technology Corp. Confidential.

Comparator

- **Comparator** – интерфейс, описывающий алгоритм сравнения двух объектов.
- Он может быть передан во многие коллекции и структуры данных для упорядочивания данных
- Несколько компараторов могут определять разные правила сортировки одних и тех же данных

```java
public class Data {

    private String name;
    private String value;

    public Data(String name, String value) {
        this.name = name;
        this.value = value;
    }

    public String getName() {
        return name;
    }

    public String getValue() {
        return value;
    }
}
```

```java
public class DataComparator
        implements Comparator<Data> {

    public int compare(Data o1, Data o2) {
        int length1 = o1.getName().length();
        int length2 = o2.getName().length();
        if (length1 > length2) {
            return 1;
        } else if (length1 > length1) {
            return -1;
        } else {
            return 0;
        }
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Comparable

- В качестве альтернативы сами объекты с данными могут реализовывать интерфейс **Comparable**, таким образом предоставляя API для сортировки себя
- Если метод **compareTo()** возвращает положительное число, то данный объект считается больше аргумента
- Если результат – отрицательное число, то данный объект меньше аргумента
- В случае равенства возвращается ноль
- Эта реализация должна соответствовать реализации **equals()** – обе они должны показывать равенство в одним и

```java
public class Data  implements Comparable<Data>{

    private String name;
    private String value;

    public Data(String name, String value) {
        this.name = name;
        this.value = value;
    }

    public String getName() {
        return name;
    }

    public String getValue() {
        return value;
    }

    public int compareTo(Data other) {
        int length1 = name.length();
        int length2 = other.getName().length();
        if (length1 > length2) {
            return 1;
        } else if (length1 > length1) {
            return -1;
        } else {
            return 0;
        }
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Примеры использования Comparator

- Если класс сам реализует **Comparable**, то отсортировать коллекцию можно следующим образом:

```
List<Data> list = new ArrayList<Data>();
list.add(new Data("aaa","aaa"));
list.add(new Data("bb","bb"));
Collections.sort(list);
```

- Если правила сортировки описаны во внешнем **Comparator**'е, то сортировка выглядит так:

```
List<Data> list = new ArrayList<Data>();
list.add(new Data("aaa","aaa"));
list.add(new Data("bb","bb"));
Collections.sort(list, new DataComparator());
```

- Некоторые коллекции умеют сортировать уже в момент добавления данных без необходимости отдельно вызывать метод сортировки:

```
Set<Data> list = new TreeSet<Data>();
list.add(new Data("aaa","aaa"));
list.add(new Data("bb","bb"));
//данные в коллекции уже отсортированы
```


© 2011 NetCracker Technology Corp. Confidential.

Collator

- Сортировка в лексикографическом порядке должна принимать во внимание не только алфавит, но и язык оригинала текста
- Эта информация не может быть в общем случае получена из текста, так что она указывается отдельно в виде наследника абстрактного класса **Collator**
- **Collator** является реализацией **Comparator**, то есть может быть

```
// Сравниваем строки в локали по умолчанию
Collator myCollator = Collator.getInstance();
if( myCollator.compare("abc", "ABC") < 0 )
    System.out.println("abc is less than ABC");
else
    System.out.println("abc is greater than or equal to ABC");
```

```
// Сравниваем строки в китайской локали
Collator collator = Collator.getInstance(Locale.CHINA);
collator.setStrength(Collator.PRIMARY);
if( collator.compare("abc", "ABC") == 0 ) {
    System.out.println("Strings are equivalent");
}
```


© 2011 NetCracker Technology Corp. Confidential.

Утилитные классы

© 2011 NetCracker Technology Corp. Confidential.

java.util.Collections

- Этот утилитный класс предоставляет набор статических методов для типовых операций над коллекциями
  - Сортировка
  - Перемешивание элементов
  - Разворот коллекции
  - Заполнение
  - Двоичный поиск
  - Определение частоты вхождений
  - Пересечение
  - Нахождение минимума и максимума
  - etc.

© 2011 NetCracker Technology Corp. Confidential.

java.util.Arrays

- Предоставляет утилитные методы для работы с массивами:
  - Бинарный поиск
  - Полное и частичное копирование
  - Преобразование к реализации интерфейса List
  - **equals()**, работающий по элементам массива
  - **deepToString()**, вызывающий **toString()** у всех элементов массива
  - Заполнение массива одинаковыми значениями
  - Сортировка

```
byte[] array = new byte[]{1,2,3,4,5};
int position = Arrays.binarySearch(array, (byte) 4);   //3
```

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

Другие реализации интерфейсов Collection API

- Обертки и адаптеры для добавления некоторой функциональности к уже существующим коллекциям
  - Синхронизирующие обертки
  - Обертки, запрещающие модификацию

```java
List<Data> list = new ArrayList<Data>();
List<Data> synchronizedList = Collections.synchronizedList(list);
List<Data> immutableList = Collections.unmodifiableList(list);
```

- «Convenience implementations» - минималистичные реализации коллекций для использования в вырожденных или специфических случаях
  - Arrays.asList()
  - Немодифицируемые коллекции из единственного элемента
  - Пустые Set, List и Map

```java
List<Data> emptyList = Collections.emptyList();
List<Data> singletonList = Collections.singletonList(new Data("lol", "wut"));
```


© 2011 NetCracker Technology Corp. Confidential.

Q&A

© 2011 NetCracker Technology Corp. Confidential.

Thank you!

© 2011 NetCracker Technology Corp. Confidential.